CONVERGENT POSE AND TWIST ESTIMATION FOR VELOCITY-DENIED MOBILE ROBOTS BASED ON A CASCADING, DUAL-FRAME, MOTION-TRACKING ESTIMATOR

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF HAWAIʻI AT MĀNOA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

MECHANICAL ENGINEERING

DECEMBER 2019

By

Brennan E. Yamamoto

Thesis Committee

A Zachary Trimble, Chairperson
Peter Berkelman
Reza Ghorbani
Zhuoyuan Song
Margo Edwards

i. Acknowledgements

It is difficult to express how privileged I am for the opportunity to pursue a PhD in a field of my passion, under comfortable financial circumstances, and alongside so many awesome people. There are countless people to thank for their contributions to my success.

First, thank you to my incredible advisor of seven years, A Zachary Trimble, to whom I attribute most of my graduate school success. Thank you for your time, expertise, patience, honesty, willingness, dedication, patience, and flexibility—and did I mention patience? How you managed to transform a cold, confused, and uncertain undergrad like myself into someone PhD-capable is a feat I will probably never fully appreciate. I could not have asked for a more fitting advisor to oversee my graduate study; working with you has truly been honor.

Thank you to my thesis committee: Peter Berkelman, for directing much of my exploration though the wonderful field of robotics; Margo Edwards, for providing our lab with multiple key opportunities to explore maritime unmanned systems, and for shaping much of my personal ideology towards the future of maritime systems in Hawaii; Reza Ghorhani, a longtime advisor, mentor, and friend (and serving on both my MS and PhD committees!); and Zhuoyuan Song, a super-approachable mentor who is always eager to assist with my research progress. Thank you also to Miguel Nunes and Ben Jones for their willingness to serve on my committee (if not for unlucky circumstances outside of their control).

Thank you to all the faculty members in the department of mechanical engineering, and the wider College of Engineering for all the expertise, teachings, and support you provided me throughout my academic career at UH. In particular, I would like to thank Brian Bingham, for originally stirring my interest in mobile robotics (and still managing to support me even after leaving UH!); Song Choi, for supporting me over the past *ten years*, be it through ECUH, Kanaloa, scholastic robotics events, volunteer events, career events, scholarship events…the list is endless; David Ma, for supporting much of my energy harvesting research; Mehrdad Nejhad, for the many intersecting research projects over the years; Miguel Nunes, for the teaching support and research advice; Wayne Shiroma, a longtime ethical mentor and friend; and Yi Zuo, for always enthusiastically providing words of encouragement and honest advice.

Thank you to the college of engineering and mechanical engineering department staff for their support over the years. In particular, I would like to thank Lewis Moore for all of the improvements made to the mechanical engineering shops (and putting up with us nocturnal students!); Jean Imada, for your unwavering support of the CoE student body and student organizations; Charlene Sato and Lori Yuu in the CoE office, and Jamie Wong and Gail Yamamoto in the CoE fiscal office, for being endlessly accommodating of me; and Joanne Yee and Shayna Yee for their excellent attentiveness and administrative support.

Thank you also to Rafael Gacel-Sinclair and the UH Office of Innovation and Commercialization/Office of Technology Transfer for working around my last-minute research schedule to produce a provisional patent in time for my dissertation.

Thank you also to my fellow faculty and staff at Leeward Community College for giving me the opportunity to teach in while pursuing my degree. In particular, I would like to thank Jennifer McFatridge, Bryson Padasdao, Nicholas Takebayashi, Jennie Thompson, and Teri-Ann Tsukenjo for supporting me for the short time I was there.

Thank you to the current and past undergraduate students who have donated blood, sweat, and tears to team Kanaloa. I hope you were able to learn as much from me as I have you. These students are: Aricia Argyris, Ileana Argyris, Amy Bentz, Karla Cortez, Steven Cory, Jonathan De Leon, Raina Ann Duenas, Richard Eidswick, Nicole Clare Hortizuela, Michael Huang, Kelan Ige, Christianne Izumigawa, Minshik Kang, Steven Kim, Paulo Lemus, Michael Loui, Aaron Nagamine, Nathan Park, Kobe Taylor, Eric Welton, Allison Wong, Darren Wong, Kai Jones, Thomas Moriyasu, Andrew Ngyuen, Johnny Pham, Ryan Roque, Russell Tolentino, Chauncey Tom, Peter Agcanas, Jason Chan, Allan Ching, Ryan Ganiron, Peiyi Kwok, Cheyenne Langamin, Benjamin Liang, Jordan Romanelli, Ronnie Soriano, Yoshio Yoshizumi, Qicheng Zhou, Raymond Andrade, Paul Baessler, Kekoa Data, Blaine Furman, Hunter Garrett, Dominic Gaspar, Danny Hong, Serena Kobayashi, Yong Cong Li, Charmaine Lum, Lisa Mahan, Kealoha Moody, Kai Outlaw-Spruell, Dane Sobol, Daniel Truong, Jordan Dalessandro, Matthew Bowers, Haley DeTure, Colin Lambert, Emily Pang, Paula Rae Penullar, Tyler Wilfahrt, Yuuma Yamamoto, Alexandra Makaiau, Marisa Matsuo, Judy Phan, and Mai Van. Thank you also to Mark Rongstad, Anthony Sylvester III, and Zhuoyuan Song for advising the team in various capacities, and graduate students Bradley Beeksma, Michael Borusinski, Curran Meek, Brenden Minei, Tyson Seto-Mook, and Ryan Theriot for contributing in various capacities.

Thank you to all of my fellow RIP lab mates from throughout the years: Daniel Wong (M.S. 2014), Evan Akuna (B.S. 2015), Reika Arta (M.S. 2016), Laura Fitzpatrick (M.S. 2016), Shane Brown (B.S. 2017), Tina Li (B.S. 2017), Aaron Nagamine (B.S. 2017), Allison Wong (B.S. 2019), Grant Takara (M.S. 2018), and Kai Jones (M.S. candidate). Thank you for being an awesome group of people to bounce ideas with, vent frustrations to, grab food with, and being there to lend a helping hand.

Thank you to the Applied Research Lab and the Office of Naval Research for most directly funding my research. Thank you also to the ARCS Foundation Honolulu Chapter, for awarding me a research scholarship *twice* during my PhD. Thank you also to the following companies and institutions for their support for team Kanaloa in various capacities: the Associated Students at the University of Hawaii (ASUH), the Association for Unmanned Vehicle Systems International (AUVSI) and RoboNation, the IEEE Hawaii Chapter, Makai Ocean Engineering, Marine Technology Society Hawaii Chapter, Mathworks, Nvidia, the UH Student Activity and Program Fee Board (SAPFB), and Velodyne LiDAR.

This paragraph is dedicated to the hundreds of classmates I have befriended over these past 11 years at UH. I regret that this list is too long, and I would risk too much attempting to recount so many amazing people. Thank you for the struggle; the many allnighters, study-sessions, assignments, projects, exams, and whatever other shenanigans engineering students find themselves in. Without you, I likely would have lost interest in engineering a very long time ago.

Thank you to my ÉMF friends, whose companionship and camaraderie makes everything I do worthwhile. Alexander Ching and Bryce Fukunaga; thanks for being the best friends a guy could ask for. Let's keep making crazy ÉMF videos.

Finally, it is not without the unwavering love and support of my extended and immediate family that this thesis could have been completed. Julienne and Joelle, thank you for always keeping me grounded, honest and humble, and giving me something greater to strive for. Mom and dad, thank you for your unending sacrifice (one which I too often take for granted), positive moral compass, and constant confidence in me.

ii. Abstract

Advancements in micro-fabrication and micro-electromechanical systems, increased market demand, and economies of scale have lowered the cost for global navigation satellite system (GNSS) and inertial measurement unit (IMU) sensor systems to levels cost-relevant for average consumers. The combination of GNSS+IMU can provide basic robot localization information, but cannot measure linear velocity, which is essential for autonomous mobile robot operation. Unfortunately, linear velocity sensors like wheeled odometers, air speed, optical flow, or doppler velocity log sensors are situationally applicable and/or cost prohibitive for many robot applications; these robots can be entitled "velocity-denied". In this work I propose a state estimation algorithm based on a cascading, dual-frame, motion-tracking estimator that is capable of providing accurate pose (positions) and twist (velocities) estimates for velocity-denied robot platforms, by probabilistically estimating the unobserved velocity state based on the time-varying information extracted from the measured position and acceleration states. Because this state estimator is based on a kinematic, motion-tracking state-transition model, it does not require dynamical information about the robot platform or the forces acting on it. I first demonstrate this state estimator algorithm on simulated mobile robot data and then on real data collected from a GNSS+IMU robot sensor system. I show that this state estimation algorithm consistently maintains a dead-reckoning pose accuracy of $< 1\,m$ of the post-interpolated pose measurements, and provides hidden a linear velocity accuracy of $< \pm 1$ m/s.

iii. Table of Contents

iv. List of Tables

v. List of Figures

vi. List of Abbreviations

AHRS is attitude, heading, reference system

GNSS is the Global Navigation Satellite System

IMU is the Inertial Measurement Unit

MARG is magnetic, angular rate, and gravity

ROS is the Robot Operating System

SNAME is the Society of Naval Architects and Marine Engineers

vii. List of Symbols

A is the state matrix of a linear dynamical system expressed in canonical state-space form

b is the bias vector due to ocean current and unmodeled dynamics *or* the damping coefficient of the robot

B is the input matrix of a linear dynamical system expressed in canonical state-space form

$\vec{B}$ is the input matrix of a Kalman filter

dt is the discrete time step

C is the Coriolis matrix of the vehicle *or* the output matrix of a linear dynamical system expressed in canonical state-space form

D is the hydrodynamic damping matrix of the vehicle *or* the map frame z-axis pointing towards down in a north-east-down system (only used in reference nomenclature) *or* the feedthrough matrix of a linear dynamical system expressed in canonical state-space form

E is the map frame y-axis pointing towards true east in a north-east-down system (only used in reference nomenclature)

$\vec{F}$ is the state-transition matrix of a Kalman filter

F is the generalized force of a robot thruster and/or acting on the robot *or* the state matrix of a discrete-time linear dynamical system expressed in canonical state-space form

G sliding covariance gain *or* the input matrix of a discrete-time linear dynamical system expressed in canonical state-space form

$\vec{H}$ is the observation matrix of a Kalman filter *or* the output matrix of a discrete-time linear dynamical system expressed in canonical state-space form

I is the rotational moment of inertia of the robot *or* an identity matrix

J is the feedthrough matrix of a discrete-time linear dynamical system expressed in canonical state-space form

K is the hydrodynamic moment acting about the robot body frame x-axis (only used in reference nomenclature)

${}_{B}^{A}\widehat{K}$ is the unit vector defining the axis of rotation of frame {B} with respect to frame {A} for the angle-axis frame representation

l_i is the magnitude of the intermediate transition vector between two reference states

m is the mass of the robot

M is the mass/inertia matrix of the vehicle *or* the hydrodynamic moment acting about the robot body frame y-axis (only used in reference nomenclature)

n is the number of measurements in a sequence of numbers over which to take the mean

N is the hydrodynamic moment acting about the robot body frame z-axis (only used in reference nomenclature) *or* the map frame x-axis pointing towards true north in a north-east-down system (only used in reference nomenclature)

p is the angular velocity about the robot body frame x-axis (only used in reference nomenclature)

q is the angular velocity about the robot body frame y-axis (only used in reference nomenclature)

$\vec{P}$ is the error covariance matrix of a Kalman filter

$\vec{Q}$ is the process noise covariance matrix of a Kalman filter

r is the angular velocity about the robot body frame z-axis (only used in reference nomenclature)

$\vec{R}$ is the measurement noise covariance matrix a Kalman filter

${}^{A}_{B}R$ is the rotation matrix of frame {B} with respect to frame {A} for the rotation matrix frame representation

t is time

u is the linear velocity along the robot body frame x-axis (only used in reference nomenclature)

$\vec{u}$ is the input vector of a Kalman filter *or* the input vector of a linear dynamical system expressed in canonical state-space form

v is the linear velocity along the robot body frame y-axis (only used in reference nomenclature)

$var(x)$ is the variance of the variable x

w is the linear velocity along the robot body frame z-axis (only used in reference nomenclature)

x is the linear position along the x-axis, or a generalized state variable

$\dot{x}$ is the time derivative of x, i.e. the linear velocity along the x-axis

$\ddot{x}$ is the time derivative of $\dot{x}$, i.e. the acceleration along the x-axis

X is the force component acting along the robot body frame x-axis (only used in reference nomenclature)

y is the linear position along the y-axis

$\dot{y}$ is the time derivative of y, i.e. the linear velocity along the y-axis

$\ddot{y}$ is the time derivative of $\dot{y}$, i.e. the linear acceleration along the y-axis

Y is the force component acting along the robot body frame y-axis (only used in reference nomenclature)

z is the linear position along the z-axis

$\dot{z}$ is the time derivative of z, i.e. the linear velocity along the z-axis

$\ddot{z}$ is the time derivative of $\dot{z}$, i.e. the linear acceleration along the z-axis

$\vec{z}$ is the measurement vector of a Kalman filter

Z is the force component acting along the robot body frame z-axis (only used in reference nomenclature)

s is a generalized sequence of numbers

S is a generalize skew-symmetric matrix

Δt is the elapsed time

$t_{1/2}$ is the half life of an exponential function

t_{smooth} is the time spent in the smoothing region of the adaptive sliding adaptive covariance method

t_{scale} is the time spent in the scaling region of the adaptive sliding adaptive covariance method

${}^{A}_{B}\vec{\Omega}$ is the angular velocity vector of frame {B} with respect to frame {A}

ϕ is the angular position about the x-axis

$\dot{\phi}$ is the time derivative of ϕ, i.e. the Euler angular velocity about the x-axis

$\ddot{\phi}$ is the time derivative of $\dot{\phi}$, i.e. the Euler angular acceleration about the x-axis

θ is the angular position about the y-axis

$\dot{\theta}$ is the time derivative of θ, i.e. the Euler angular velocity about the y-axis

$\ddot{\theta}$ is the time derivative of $\dot{\theta}$, i.e. the Euler angular acceleration about the y-axis

${}^{A}_{B}\theta$ is the angle of rotation about ${}^{A}_{B}\hat{K}$ of frame {B} with respect to frame {A} for the angle-axis representation

ψ is the angular position about the z-axis

$\dot{\psi}$ is the time derivative of ψ, i.e. the Euler angular velocity about the z-axis; also identical to the angular velocity Ω_z in two dimensions

$\ddot{\psi}$ is the time derivative of $\dot{\psi}$, i.e. the Euler angular acceleration about the z-axis

τ is the vector of the controlling force input

τ_{wind} is the vector of the wind-induced force

1. Introduction

"Mobile robotics" refers to the class of robots capable of motion of its base platform within its environment. Advanced mobile robots must not only move, but also *sense* their motion to some degree. The methods employed, and information extracted from a mobile robot sensing system will vary significantly based on the desired application; however, all mobile robots propose some answer to the question of *robot localization*; what is my pose (positions) and twist (velocities) relative to my surroundings?

Advancements in micro-fabrication and micro-electromechanical systems, increased market demand, and economies of scale have lowered the cost for global navigation satellite system (GNSS) and inertial measurement unit (IMU) sensor systems to levels cost-relevant for average consumers. Developing mobile robots that can convergently estimate pose (position) and twist (twist) using *only* these two sensors is highly attractive for cost-sensitive autonomous mobile robots; however, the combination of GNSS+IMU is incapable of measuring linear velocity, which is essential for autonomous robot operation. For this reason, I call robots employing this sensor system "velocity-denied". Velocity-denied robot operation is a relevant problem because linear velocity sensors like wheeled odometers, air speed, optical flow, or doppler velocity log sensors are situationally applicable and/or cost prohibitive; if the robot linear velocity can be estimated using more universally available position and acceleration sensors like GNSS and IMUs, a significant portion of mobile robots stand to benefit. There are many currently-existing well-documented and/or open-source state estimation algorithms (which will be discussed in detail in this work); however, these systems require a generalized linear velocity sensor, and will not work when velocity-denied. In this work, I present a state estimation algorithm based on the discrete Kalman filter, that is capable of convergently estimating the pose and twist of a velocity denied mobile robot.

In chapter two of this work, I will describe the process of state estimation for robot localization, the current state-of-the-art, and elaborate on why the velocity-denied mobile robot problem is a relevant one. In the second section, I describe the two most common probabilistic state observers for robot localization: the discrete Kalman filter and the particle filter; and explain why I chose the Kalman filter is more appropriate for this problem. In the third section, I discuss the existing, well-documented/open-source solutions for state estimation for mobile robots and identify why these solutions are insufficient for the needs of velocity-denied robot platforms. I then describe the Kalman filtering process, the quality of the measurements our GNSS+IMU sensor system has access to, and the constant-acceleration state-transition kinematic motion model I employ in this work. Finally, I will discuss the fundamental problems that need to be addressed in a state estimation solution that will convergently estimate pose and twist for a velocity-denied mobile robot.

In chapter three, I describe the revised state estimation algorithm for robot localization based on a cascading, dual-frame, motion-tracking state estimator, that is capable of estimating pose (positions) and twist (velocities) for robot systems without a linear velocity sensor. The novel contributions of this algorithm can be categorized into four areas:

- A discrete-time interpretation of the robot body reference frame. This allows us to fuse position, velocity, and acceleration sensor data in the robot body frame.
- Adaptive covariance profiling for sensor quantization mitigation and improved estimator prediction capability.
- Unobserved linear velocity estimation based on the moving-mean forward-difference of GNSS position data, and trapezoidal method of accelerometer (IMU) data.
- Cascading estimators in the robot body frame and then in the map frame, virtually eliminating error due to frame transformation.

In chapter four, I evaluate my revised state estimation algorithm against simulated sensor data with a known ground truth. From this, I draw several key learnings. I then take the same state estimation algorithm and use it against real sensor data collected from a moving platform. Using this data, I show that the state estimation algorithm provides pose predictions in between GNSS sensor updates with a mean error ranging from 0.4 to 1.2m across all of the tested trials, and a linear velocity error between 0.1 to 0.8 m/s using the corrective linear velocity open loop bias predicted using simulated data.

2. Existing state estimation algorithms for robot localization

2.1. The need for state estimation and sensor fusion in robot localization

The practice of state estimation belongs to a much wider field of Bayesian statistics [1], which can generally be described as the system for describing the uncertainty of knowledge using mathematical probabilities [2]. In state estimation, noise-prone measurements are modeled as *probabilities* that are mathematically coupled to other probabilistic state variables that evolve in parallel over time. Instead of treating imperfect measurements as direct reflections of a state variable, probabilistic measurements increase our *confidence* (or certainty) in a state variable. We can further improve our confidence in a state variable by combining these probabilistic measurements with a dynamical model that can predict the next state variable based on its past states. Throughout this paper, I refer to the mathematical tool that calculates the probabilistic system states as a *state observer*, and its outputs (i.e. improved "measurements") as *state estimates*. Finally, when employing a state observer to provide state estimates using information from multiple sensors simultaneously, I call this *sensor fusion*.

To better understand sensor fusion and the benefits it provides, Mitchell [3] describes multiple ways sensor fusion can improve the measurement process:

- *Representation*. The state estimate after fusion is described at an improved level or granularity over the original data set.
- *Certainty* and *accuracy*. The probability distribution $p(x)$ and the standard deviation σ of the state estimate after fusion is improved over the original data set.
- *Completeness.* The state estimate after fusion describes information in a way that allows for a more complete view on the environment than the original data set does.

Sensor fusion systems can also be classified by their *metrology*, i.e. how the data from multiple sensors is aggregated. Boudjemaa and Forbes [4] classify sensor fusion *metrology* by:

- *Fusion across sensors.* Fusing information from multiple sensors measuring the same quantity.
- *Fusion across attributes*. Fusing information from multiple sensors measuring different quantities associated with the same experimental situation.
- *Fusion across domains*. Fusing information from multiple sensors measuring the same quantity, but over different ranges or domains.
- *Fusion across time*. Fusing current information with historical information (from an earlier measurement).

Durrant-Whyte [5] describe a similar classification of sensor fusion systems based on their *configuration*:

- *Competitive* information occurs when multiple sensors supply information in the same location and degrees of freedom.
- *Complementary* information occurs when multiple sensors supply different information about the same geometric feature in different degrees of freedom.
- *Cooperative* information occurs when one sensor relies on another for information prior to observation.

From these descriptions, it is easy to see why sensor fusion is valuable; a well-implemented state estimator offers improved measurement performance and can provide information that would otherwise be unavailable from the unfused sensor outputs alone. Unsurprisingly, sensor fusion for robot localization is a highly studied problem in mobile robotics [6]. In this work, I employ an improved sensor fusion algorithm for robot localization that is designed to estimate pose (positions) and twist (velocities) using *only* a low cost GNSS and IMU sensor system. Because the GNSS+IMU sensor system is incapable of directly measuring linear velocity, most mobile robots require a generalized linear velocity sensor for full state estimation; however, linear velocity sensors are not applicable and/or too cost-prohibitive for many mobile robot applications. In the improved algorithm, I extract the probability distribution (estimate) of the linear velocity states based only on sensor data from related states.

In order to validate the improvements of our improved state estimation algorithm, I must first establish the attributes and performance of a "general" state estimation algorithm for robot localization. That said, it is difficult to claim that the performance of an algorithm is a reasonable "general" representation, for a few reasons:

- The full sensor fusion "algorithm" has many steps, and with variables that vary for each application. This makes it challenging to claim that any form of this algorithm is generally applicable to all, or even general cases.
- Accessible scientific literature is largely focused on the theoretical aspects of filtering and signal processing. This information is an important part of this work; however, I are also concerned about the application and performance of these theoretical concepts to practical mobile robot platforms. Due to the nature of scientific publication, papers that discuss the application of already-vetted theoretical processes are few, and rarely discuss the detail necessary to fully reproduce their system.
- Based on the state-estimation performance of commercially-available inertial navigation systems, it is possible that solutions to the challenges described have already been commercially addressed; however, the majority of these solutions are "black-boxed" (proprietary) and inaccessible to researchers. Because these systems combine multiple technologies, e.g. differential GNSS, real-time kinematic (RTK) and precise point positioning (PPP), multiple GNSS sensors, and onboard sensor fusion, it is impossible to comprehensively understand the contributive benefits of each technology without access to the underlying source code driving these systems.

For these reasons, it can be difficult to justify academic research effort into improved methods of filtering for lower-cost, more primitive sensor solutions; however, there are a few reasons scientific study is essential. Firstly, improvement to proprietary solutions are limited only to the small group of individuals who can access the source code. Secondly, researchers are forced to assume that black-boxed solutions are performing operations based on the claims of the manufacturer. As previously noted, when analyzing systems that employ multiple technologies, it is impossible to understand the contributive benefits of each individual technology. Thirdly, each example mobile robot can have significantly different operational requirements; with a black-boxed system, advanced users lose the ability to make any modifications to the underlying system operation to better fit their application. Some examples of useful modifications to fit operational requirements could be manually varying sensor covariances to account for environmental noise; making the system robust to loss of sensors (e.g. going into a tunnel, changing measurement domains, etc.); tweaking an estimation system to provide more smoothing at the cost of accuracy, or vice versa.

In this chapter, I aim to characterize this "general" state estimation algorithm for robot localization; this will allowme to identify the shortcomings in this algorithm if used with a velocity-denied sensor system like a GNSS+IMU. In section 2.2, I discuss the two preeminent probabilistic state observers for robot localization: the Kalman filter and particle filter. In section 2.3, I scrutinize open-source/well-documented state estimation algorithms on known robot platforms; this will tell us: (1) how state estimation algorithms are currently performed, (2) the variance in algorithms that are currently employed, and (3) the limitations/shortcomings of current state estimation algorithms, so that improvements can be proposed. In section 2.4, I describe the iterative Kalman filter, which is the underlying statistical state observer employed for most state estimation algorithms for robot localization. In section 2.5, I discuss the expected precision and refresh rate specifications of the low-cost GNSS and IMU sensors that I will employ in this work. In section 2.6, I discuss the constant-acceleration state-transition model, the kinematic motion model employed by most state observers for state prediction. Finally in section 2.7, I summarize my findings by outlining the major shortcomings in the current robot localization state estimators for a velocity-denied mobile robot using only a GNSS+IMU sensor system.

2.2. The Kalman filter and particle filter state observers

The probabilistic state observer is a mathematical tool whose goal is to estimate the probability distribution of system state variables based on a state-transition dynamical function, and measurements related to the state variables; it is the most important component in the full state estimation algorithm. Many statistical modeling techniques are available to solve these types of probabilistic time-series filtering problems; as described in the seminal 1999 paper by Roweis et al. [7], many common statistical techniques for multidimensional time series modeling can be seen as variants of the same underlying unsupervised learning model. These similar processes include principal component analysis, mixtures of gaussian clusters, vector quantization, independent component analysis, Kalman filter models, and hidden Markov models. Specifically for the task of mobile robot localization, the two most studied state observers are the Kalman filter and its extended and unscented variants [8]–[13], and nonparametric Markov chain filters, which are typically classified into grid-based [14]–[16], topology-based [17], [18], and particle-based Markov chain filters [19]–[23]. Generally speaking, the Kalman filter approach to robot localization describes the robot position and sensor readings as unimodal Gaussian distributions, which is robust and computationally cheap; however, the Kalman filter is incapable of representing multi-modal (nonlinear) probability distributions, and generally requires that the starting location of the robot is known [6], [21], [24]. Subsequent evolutions of the Kalman filter, such as the extended [10], [11] and unscented [12], [13] Kalman filter are capable to employing nonlinear state-transition equations and measurements, but still fundamentally propagate a single Gaussian probability distribution per state variable, limiting its representational power. The non-parametric filter approach to robot localization describes the probability distribution of the robot's surrounding environment; in grid-based approaches, the filter calculates the probability distribution over a grid of possible positions in the environment; in topology-based approaches the filter calculates the probability distribution for all sensed features in the environment; and in particle-based approaches, the filter calculates the probability distribution for a randomly-drawn set of samples in the environment [6].

While non-parametric filter approaches to the robot localization problem are relatively recent, the particle-based non-parametric filter has emerged as the preeminent method for probabilistic robot localization in applications where the starting location of the robot is unknown and extrinsic sensor information extracted from the environment are the primary source of information. This extrinsic sensor data is typically in the form of laser scans, sonar, and/or feature extraction from vision, i.e. GNSS-denied applications. This filter is generally referred to as the *particle filter*. Despite the many strengths of the particle filter, its philosophy of defining the probability distribution of the surrounding environment, rather than the distribution of the robot make it less relevant for the fusion of *intrinsic* sensors like the GNSS and IMU, which measure the internal state variables of the robot. Therefore, for sensor fusion of GNSS and IMU sensors, the Kalman filter was selected as the state observer for this work.

In the next section, I will review known open source and/or well-documented state estimators. Because I am focusing on systems employing Kalman filter-based state observers, I will omit known systems employing particle filter-based state observers. I will then return to a detailed explanation of the Kalman filter in a later section.

2.3. A survey of existing open-source or well-documented state estimation algorithms for robot localization

In this section, I will scrutinize existing examples of open-source or well-documented state estimators for robot localization, specifically for pose *and* twist estimation on robot platforms. These example state estimation algorithms can take multiple forms:

- Documented state estimation algorithm of a working robot example

- Open-source state estimation software for compatible micro-controllers and/or micro-processors for known robot systems
- Pure software packages that require the user to manually interop into their own compatible hardware systems
- Proposed models that could be used in a state estimator

To keep the search relevant for the problem statement in this work, I narrowed the search to state estimation algorithms that extract the probability distribution of pose (positions) and twist (velocities) of a robot using a Kalman filter. I am particularly interested in the sensors used by the robot, and how the state-transition equations incorporate data from these sensors (if applicable). This information will be relevant in later sections. The state estimation algorithms analyzed in this section are:

- State estimation software packages
  - ArduPilot [25], a family of open source flight control software. ArduPilot controllers include the ArduPilot Copter [26] for helicopter and multicopter systems, the ArduPilot Plane [27] for fixed wing aircraft, and ArduPilot Rover for ground vehicles and boats [28].
    - ArduPilot GitHub repository [29]
    - ArduPilot documentation [30]
    - ArduPilot state estimation documentation [31]–[33]
  - NavLab [34], a generic tool for robot navigation. Maintained by the Forsvarets Forskningsinstitutt (Norwegian Defense Research Establishment).
    - NavLab introduction paper [35]
  - Open-Source-Sensor-Fusion [36], an open-source sensor fusion/state estimation software for MEMS chips. Maintained by the MEMS Industry group.
    - Open-Source-Sensor-Fusion Kalman filter documentation [37]
  - OpenIMU [38], the sensor fusion/state estimation software for ACEINNA IMU systems. Maintained by ACEINNA.
    - OpenIMU GitHub repository [39]

        - OpenIMU documentation [40]
        - OpenIMU state estimation documentation [41]
    - PX4 [42] flight control software, an open source flight control software. Maintained by Dronecode Project Inc.
        - PX4 GitHub repository [43]
        - PX4 developer documentation [44]
        - PX4 state estimation documentation [45], [46]
        - PX4 user guide documentation [47]
    - ROS robot_localization node [48], a sensor fusion/state estimation software package for the robot operating system. Maintained by Tom Moore.
        - robot_localization GitHub repository [49]
        - robot_localization introduction paper [50]
        - robot_localization documentation [51]
- Surface vehicle Kalman filter-based estimators
    - Alam et al. surface vehicle [52]
    - Dhariwal et al. surface vehicle [53]
    - Papadopoulos et al. surface vehicle [54]
    - Subramanian et al. surface vehicle [55]
- Underwater vehicle Kalman filter-based estimators
    - Bozorg et al. underwater vehicle [56]
    - Ferreira et al. MARES underwater vehicle [57]
    - Gadre et al. underwater vehicle [58]
    - Karras et al. underwater vehicle [59]
    - Loebis et al. autonomous underwater vehicle [60]
    - Petillot et al. underwater vehicle [61]
    - Sabet et al. autonomous underwater vehicle [62]
- Wheeled vehicle Kalman filter-based estimators
    - Bayramoglu et al. two-wheel corridor robot [63]
    - Jetto et al. two-wheel robot [64]
    - Kiriy et al. golf course lawn mower robot [65]

- Kwon et al. two-wheel robot [66], [67]
- Lee et al. two-wheel robot [68]
- Messom et al. two-wheel ball-tracking robot [69]

2.3.1. State estimation software packages

ArduPilot is an open-source autopilot software, intended to run on specific (open-source) sensor+microcontroller platforms. There are software variants for fixed wing aircraft, multicopter drones, wheeled ground vehicles, and boats. It has excellent documentation [30], access to source code [29], and is actively maintained at the time of writing. The state estimation algorithm is documented on this webpage [31], and the code deriving the state equations can be found on this webpage [33]. The state estimation algorithm is based on an extended Kalman filter, and fuses information from an accelerometer, rate gyroscope, magnetometer/compass, GNSS/GPS, airspeed, barometric pressure measurements, optical flow, and laser rangefinders. The state-transition model is a kinematic motion model based on the derivative changes in each state. Not all sensors are required for the system to operate; however, a linear velocity sensor is necessary to correct the linear velocity state estimates, i.e. this state estimation algorithm will not provide a reasonable linear velocity estimate for a velocity-denied robot.

Navlab is a closed-source state estimation software. Despite its close-source nature there is enough information in [35] describing NavLab's underlying algorithms that it warrants consideration. Navlab includes multiple software components (simulation, real-time estimation, and data post processing), and has vetted real-world performance. Its discussion of the state estimation algorithm does not include the state-transition equations of the Kalman filter, but it does describe the required sensors. These include an accelerometer, rate gyroscope, magnetometer/compass, and generalized position, velocity, and depth/altitude measurements. Subsequent Navlab publications related to state estimation and navigation primarily discuss orientation estimation for inertial navigation systems [70], and the various methods of measuring heading/orientation [71]. The need for a generalized velocity sensor indicates that NavLab is not designed for pose and twist estimation for a velocity-denied robot.

In addition, the NavLab introduction paper [35] identifies the problem of sensor quantization, which occurs when employing multiple digital sensors that update at varying rates. This manifests as hard "jumps" in the estimator output when the slower updating sensor(s) produces an update; this can be particularly detrimental to filter performance if the state-transition equations rely on discrete differentiation of the estimator output. In [35], the hard jumps in the estimator outputs were removed by employing a post-processing optimal smoothing filter; however, this filter cannot be implemented in real time. The problem of real-time sensor quantization mitigation is discussed in depth on our work.

Open-Source-Sensor-Fusion is a state estimation software, intended to run on specific IMU platforms; it has excellent documentation, and easy access to source code [36], though its last update was in 2015, making it less frequently updated than some of the other algorithms analyzed in this work. This particular state estimation algorithm is designed only for fusion of IMU sensors (accelerometer, rate gyroscope, magnetometer) for estimating angular positions and velocities; therefore it is not a full robot localization solution on its own. This process, often called magnetic, angular rate, and gravity, or "MARG", or attitude, heading reference system, or "AHRS" estimation, was formally published as a complete filtering algorithm in [72], and is generally considered to be a solved problem. The majority of commercial IMU systems on the market today employ a similar built-in AHRS/MARG state estimator built in. Documentation on the open-source sensor fusion Kalman filter implementation can be found in [37].

For further reading on AHRS/MARG state estimators, this application report [73] of the well-documented, commercially-available-off-the-shelf (COTS) MSP430F5xx AHRS system by Texas Instruments details the application of a very similar system to hardware.

OpenIMU is an open-source state estimation software, intended to run on specific (open-source) sensor+microcontroller platforms; it has excellent documentation [40], easy access to source code [39], and is actively maintained at the time of writing. The state estimation algorithm is documented on this webpage [41]; it fuses sensor data from an accelerometer, rate gyroscope, magnetometer, GNSS/GPS, and a generalized velocity sensor for velocity measurements. The state-transition model is a kinematic motion model based on the derivative changes in each state. Similar to other state estimation algorithms scrutinized, not all sensors are required for the system to operate; however, a linear velocity sensor is necessary to correct the linear velocity state estimates, i.e. this state estimation algorithm will not provide a reasonable linear velocity estimate for a velocity-denied robot.

PX4 is an open-source autopilot software, intended to run on specific (open-source) sensor+microcontroller platforms; it has good documentation [44], [47], easy access to source code [43], and is actively maintained at the time of writing. The state estimation algorithm is documented on this webpage [45], as well as additional details on this repository [46]. PX4 fuses information from an IMU (accelerometer, rate gyroscope, magnetometer), height sensor (barometric pressure and/or GNSS/GPS), GNSS/GPS, rangefinder, airspeed, optical flow, and external vision sensor. The state-transition model is a kinematic motion model based on the derivative changes in each state. Similar to other state estimation algorithms scrutinized, not all sensors are required for the system to operate; however, a linear velocity sensor is necessary to correct the linear velocity state estimates, i.e. this state estimation algorithm will not provide a reasonable linear velocity estimate for a velocity-denied robot.

"robot_localization" is a state estimation software node/package that runs on the robot operating system (ROS) meta-operating system. It has good documentation [51], easy to access source code [49], and is actively maintained at the time of writing. Details about the state estimation algorithm are not well-documented; however, this 2016 paper [50] outlines the general operating, and performance of the robot_localization node. robot_localization fuses sensor data from position data from a GNSS, orientation data from a

IMU, and linear velocity data from a generalized velocity sensor (rotary encoder odometry data from a wheeled robot, and optical flow data from a low-altitude drone were cited examples). At the time of publication of [50] (2016), robot_localization does not utilize sensor data from the accelerometer, although this is expected future functionality. Similar to other state estimation algorithms scrutinized, not all sensors are required for the system to operate; however, a linear velocity sensor is necessary to correct the linear velocity state estimates, i.e. this state estimation algorithm will not provide a reasonable linear velocity estimate for a velocity-denied robot.

2.3.2. Surface vehicle Kalman filter-based estimators

Alam et al. [52] describes a state estimation algorithm based on the extended Kalman filter for a high speed surface and aerial vehicle. The state-transition model is a kinematic motion model based on the fusion of a GNSS and IMU sensor. The problem statement of this paper is similar to our work, as their goal is to extract position and velocity using only a GNSS and IMU sensor; however, this work measures velocity from the direct output of the GNSS sensor. Because low-cost GNSS sensors operating in single point precision (SPP) mode calculate velocity by directly differentiating two consecutive positions, this velocity measurement is very poor, i.e. meter-per-second-level of accuracy [74]. Their sensor fusion positioning system has a RMSE accuracy of 6.4m, as compared to a reference RTK-based GNSS positioning system. It is unclear what GNSS velocity system was employed, and the paper explicitly states that the velocity performance was not ultimately evaluated against any known ground truth.

Dhariwal et al. [53] describes a state estimation algorithm based on the extended Kalman filter for a surface vehicle. The state-transition model is a dynamic model based on the vehicle mass Coriolis, centripetal, damping, buoyancy and gravitational forces. The state-transition model fuses sensor data from a GNSS, IMU, and open loop velocity data based on straight-line testing of commanded inputs, equivalent thruster force, and conversion efficiency parameters. Tentative conclusions suggest waypoint accuracy of approximately 2 m, which is very close to the GNSS/GPS precision. This demonstrates very high dependence on position measurements from the GNSS. The accuracy of the linear velocity state estimates was not presented.

Subramanian et al. surface vehicle [55] describes a state estimation algorithm based on the extended Kalman filter for a surface vehicle. The purpose of this vehicle is to visually-track features on the shoreline using an omni-directional camera; these features are located relative to the vehicle pose. The state-transition model is a kinematic motion model based on the fusion of a GNSS, IMU, doppler velocity log, and visually tracked features in the environment from the camera. They found that their vision tracking method is reliable in a river environment but has trouble in environments with strong reflecting ripples. Objective analysis of the robot estimator localization performance was not presented.

2.3.3. Underwater vehicle Kalman filter-based estimators

Bozorg et al. [56] describes a state estimation algorithm based on the extended and unscented Kalman filter for an underwater vehicles. The state-transition model is a kinematic motion model based on the fusion of an inertial navigation system and doppler velocity log in simulation. They found that the extended Kalman filter outperformed the unscented Kalman filter on simulated data of an NPS AUV II underwater vehicle.

Ferreira et al. [57] describes a state estimation algorithm based on the extended Kalman filter and the particle filter for the MARES underwater vehicle; the performance of the two state observer methods are then compared. The state-transition model is a dynamic model based on the vehicle mass, Coriolis, drag, gravity, and thruster forces. It fuses sensor data from a magnetometer, rate gyroscopes, depth sensor, and two long baseline acoustic beacons for positioning. They found that the particle filter is a promising estimator that does not make assumptions about the system noise; however, the number of particles must be large to ensure robustness; they frequently experience divergence in their particle filter estimates. Conversely, the extended Kalman filter requires that assumptions about the noise are made, and requires initialization, but is generally more robust and less exigent in terms of computational requirements.

Gadre et al. [58] describes a state estimation algorithm based on the extended Kalman filter for an underwater vehicle. The state-transition model is a kinematic motion model based on the fusion of a magnetometer, depth sensor, open loop velocity measurement based on propeller velocity, and position based on an acoustic pulse. They found that the system was fully observable for all trajectories except for those that are segments of straight lines that pass through the origin (the location of the acoustic beacon).

Karras et al. [59] describes a state estimation algorithm based on the Kalman filter for an underwater vehicle. The state-transition model is a kinematic motion model based on the fusion of an IMU and a laser-based vision system that consists of two underwater laser pointers and a single CCD camera mounted on the ROV. They found that the laser-based vision system corrects the unbounded position error of the IMU, even after long periods of time where no measurements were taken.

Loebis et al. [60] describes a state estimation algorithm based on the extended Kalman filter with adaptive covariance tuning using fuzzy logic for an underwater vehicle. The state-transition model is a kinematic motion model based on the fusion of an IMU periodic GPS (when the vehicle surfaces). They found that the adaptive covariance tuning based on fuzzy logic improved the estimation accuracy of the extended Kalman filter.

Petillot et al. [61] describes a state estimation algorithm based on the Kalman filter for an underwater vehicle. The state-transition model is a kinematic motion model based on the fusion of an IMU and forward-facing multi-beam sonar for 2D obstacle avoidance, mapping via constructive solid geometry, and extraction of the dynamics of features in the surrounding environment. They demonstrated their algorithm on real sonar data in a real trial, finding that their system generates very smooth paths, can handle complex and changing workspaces, and can be used for improved motion estimation using the tracking module for sonar servoing, and simultaneous localization and mapping.

Sabet et al. [62] describes a process for more-accurately estimating the hydrodynamic coefficients of an underwater vehicle. These coefficients can be used to build the equations of motion of a state-transition model for the pose estimation of an underwater vehicle. This dynamical model is highly detailed, including three-dimensional coefficients for cross-flow drag, added mass, added mass cross-term, added mass cross-term and fin lift, rolling resistance, body and fin lift and Munk moment, body lift force and fin lift, fin lift force, fin lift moment, propeller thrust, and propeller torque. These equations of motion were then used as state-transition equations for extended and unscented Kalman filters. They found that the unscented Kalman filter is faster and more accurate than the extended Kalman filter for their state-transition equations.

2.3.4. <u>Wheeled vehicle Kalman filter-based estimators</u>

Bayramoglu et al. [63] describes a state estimation algorithm based on the extended Kalman filter for a differential-drive two-wheel robot. The state-transition model is a kinematic motion model based on the fusion of incremental measurements from wheel encoders and visual odometry data from a camera based on a vanishing point analysis in a corridor. Their system demonstrated localization accuracy of $< 3\ cm$ for path segments $\leq 10\ m$.

Jetto et al. [64] describes a state estimation algorithm based on the “adaptive” extended Kalman filter for a differential-drive two-wheel robot. The state-transition model is a kinematic motion model based on the fusion of incremental measurements from wheel encoders and extrinsic proximity sonar sensors. In this system, the noise covariances of the extended Kalman filter are varied as the robot moves through the environment. They found that an adaptive algorithm of this nature shows promise for preventing filter divergence over time.

Kiriy et al. [65] describes a state estimation algorithm based on the extended Kalman filter for a differential-drive three-wheel robot intended for lawn moving on golf courses. The state-transition model is a kinematic motion model based on the fusion of incremental measurements from wheel encoders, a fiber-optic rate gyroscope, and machine vision from a camera. This system demonstrated an average cartesian error of 0.352 m with a 0.296 m standard deviation, relative to a ground truth trajectory.

Kwon et al. [66], [67] describes a state estimation algorithm based on the extended Kalman filter for a differential-drive two-wheel robot. The state-transition model is a kinematic motion model based on the fusion of incremental measurements from wheel encoders and position data from an “indoor GPS” system. This indoor GPS employs four ultrasonic transducers in the ceiling, and two ultrasonic receivers positioned on the robot. They found that the estimator proved to be promising for indoor mobile robot localization, even with environmental disturbance such as an uneven floor, doorsill, and other obstacles.

Lee et al. [68] describes a state estimation algorithm based on the extended Kalman filter for a differential-drive two-wheel robot. The state-transition model is a kinematic motion model based on the fusion of incremental measurements from wheel encoders and one or more optical flow sensors. Because wheel encoders and optical flow are both incremental sensor systems (it lacks a correcting absolute or extrinsic sensor system like GNSS, vision, laser scans, etc.) the purpose of this work is to improve the accuracy of *intermediate* position and velocity estimates over a standalone wheel encoder system. They found that the fusion of the optical flow sensors gives better estimates than when dead reckoning.

Messom et al. [69] describes a state estimation algorithm based on the Kalman filter for a differential-drive two-wheel robot. The state-transition model is a kinematic motion model based on the fusion of incremental measurements from wheel encoders and visual data from a camera. In this particular application, the state estimator tracks, and localizes off of the position of a stationary ball using the camera. This system demonstrated improved ball tracking and interception rate and was successfully deployed in the 2002 Singapore Robotics Games and 2002 FIRA Robot World Soccer Competition.

2.3.5. Summarizing existing state estimation estimators for robot localization

There are two important takeaways from our survey of existing state estimation algorithms. Firstly, state observer state-transition models are based on a kinematic motion model or a dynamical model based on the dynamics of the robot, with a heavy preference on the former. This is somewhat surprising, since Kalman filters are typically thought of as dynamic state estimators, i.e. integrating the plant dynamics to predict the state of the system at a forward point in time. In these kinematic motion models, the state-transition equations follow the form given in equation (1).

$$x_{k+1} = x_k + \dot{x}_k \cdot dt \tag{1}$$

Where
x_{k+1} is the state at time $k + 1$
x_k is the state at time k
$\dot{x}_k$ is the derivative of the state
dt is the discrete-time step

Equation (1) can be interpreted as "the state at time $k + 1$ can be predicted by the state at time k summed with its motion (the product of its derivative and the discrete time step)". There are minor differences from robot to robot, particularly when the kinematic frame transformations are folded into the state-transition equations; however, the motion model consistently follows this intuitive form. The philosophy of the kinematic motion model can be described as predicting the positions and velocities of the robot based on the based on integrating its past motion based on an assumption of constant acceleration. For this reason, this kinematic motion model is often called the "constant acceleration" state-transition model [31], [41], [45], [69], [75], [76].

A dynamic state-transition model contains information about the dynamic constants of the system, e.g. inertia, drag, external forces, etc. These state-transition equations are given the input forces to the system at each point in time, and the equations can be integrated to solve for the resulting velocities and accelerations. The work by Fossen in [77] suggests the candidate two-dimensional model for estimation of velocities using only position and heading measurements in equation (2).

$$M\ddot{x} + C\dot{x} + D\dot{x} - R^T(\psi)b + g_0 = \tau + \tau_{wind}$$

Where
M is the mass/inertia matrix of the vehicle
C is the Coriolis matrix of the vehicle
D is the hydrodynamic damping matrix of the vehicle (2)
R is the measurement noise covariance
b bias vector due to ocean currents and unmodeled dynamics
τ is the vector of the controlling force input
τ_{wind} is the vector of wind-induced force

There are advantages to disadvantages to the kinematic motion model like the one given in equation (1) and a dynamic model like the one given in equation (2). A kinematic motion model allows the state observer to estimate the robot states without information about the dynamics of the robot or the surrounding environment. This has the advantage of a single estimator universally working for all robot systems, regardless of the size and dynamics of the robot system. The disadvantage of the kinematic motion model is inherent in its assumption of constant acceleration; because the state-transition equation does not take the controlling force input into account, it will have poorer "dead reckoning" (pose estimation without absolute and/or extrinsic position measurement) performance. By contrast, a dynamic model will demonstrate improved state estimation performance, but requires that the dynamic properties of the system are precisely characterized for the robot at each time step. Robot systems equipped with absolute and/or extrinsic position sensors like GNSS, visual feature tracking, laser scans, etc. can bound their absolute position error, and thus, do not require maximum dead reckoning performance. For these cases, the universality and ease-of-implementation of a kinematic motion model are generally preferred. In cases where robot systems are expected to dead reckon for extended periods of time, such as underwater and GNSS-denied applications, a fully characterized dynamical model will significantly improve estimator performance. Thusly, I will be using a state-transition model based on the kinematic motion model given in equation (1). This will be described in detail in section 2.6.

The second important takeaway from our survey of existing state estimation algorithm is the need for a generalized linear velocity sensor for both pose and twist estimation. When employing an absolute or extrinsic position sensor to bound the error of the position state, smooth position state estimation performance in between position measurement updates requires a linear velocity sensor; however, linear velocity sensors like wheel encoders, air speed, optical flow, or doppler velocity log sensors are only situationally applicable and/or cost prohibitive. In two fringe examples, Dhariwal et al. [53] and Gadre et al. [58] attempted to solve this problem by approximating the linear velocity state based on the open loop control inputs to the underwater thrusters. In both cases, the accuracy of the final velocity estimate was not evaluated; however, one can assume that these open loop methods are inferior to a true physical sensor. This reinforces the need for a state estimation algorithm that is capable of convergent pose *and* twist estimation for robots without a physical linear velocity sensor.

2.4. The discrete Kalman filter

The discrete Kalman filter is an iterative linear filtering and prediction process proposed in 1960 [8] that falls under a wider family of Bayesian filters [78]. Rather than treating measurements as direct reflections their respective state, the Kalman filter treats each measurement as a contribution to a joint probability distribution. The Kalman filter consists of two steps; the first *prediction* step takes state estimates from the previous iteration, and inputs them into its *state-transition* dynamical model to predict the state of the current iteration; the second *update* step takes the predicted states, and compares them against state measurements using a weighted average, with more weight given to whichever value has less covariance (more certainty). The weighted average of each state becomes the pose estimate of the Kalman filter, which is then used as the input to the prediction step of the next filter iteration.

The first step of the Kalman filter is referred to as the *predict* step. In this step, the Kalman filter calculates a state vector estimate *prediction* $\hat{\vec{x}}_{k+1}$ by evaluating the state-transition matrix $\vec{F}$ and input matrix $\vec{B}$ using the state vector from the previous iteration $\hat{\vec{x}}_k$ and the inputs from the current iteration $\vec{u}_k$. The Kalman filter also calculates the error covariance matrix *prediction* $\vec{P}_{k+1}$ using the error covariance matrix from the previous iteration $\vec{P}_k$, the state-transition matrix $\vec{F}$, and process noise covariance matrix $\vec{Q}_k$. The predict step equations are given in equations (3) and (4), respectively.

$$\hat{\vec{x}}_{k+1} = \vec{F}\hat{\vec{x}}_k + \vec{B}\vec{u}_k \tag{3}$$

Where

$\hat{\vec{x}}_{k+1}$ is the state vector estimate *prediction*. It describes the prediction of the state vector estimate after considering its internal state-transition model, but does not consider any measured data. It is a $[n \times 1]$ vector, where n is the number of states.

$\vec{F}$ is the state-transition matrix. It describes the equations-of-motion (or dynamical equations) of the system in matrix form. It is a $[n \times n]$ matrix, where n is the number of states.

$\hat{\vec{x}}_k$ is the state vector at time k. In most cases, this is the state vector estimate update from the previous iteration, which describes the Kalman filter's previous final estimate after considering its internal state-transition model and the measured data. It is a $[n \times 1]$ vector, where n is the number of states.

$\vec{B}$ is the input matrix. It describes the inputs to the equations-of-motion (or dynamical equations) of the system in matrix form. It is a $[n \times m]$ matrix, where m is the number of input states.

$\vec{u}_k$ is the input vector. It describes the inputs to the equations-of-motion for each iteration of the Kalman filter. It is a $[m \times 1]$ vector, where m is the number of input states.

$$\vec{P}_{k+1} = \vec{F}\vec{P}_k\vec{F}^T + \vec{Q}_k \tag{4}$$

Where

$\vec{P}_{k+1}$ is the error covariance matrix *prediction*. It describes the variance (certainty) of the state vector estimate prediction after considering its internal state-transition model, but does not consider any measured data. It is a $[n \times n]$ matrix, where n is the number of states.

$\vec{F}$ is the state-transition matrix. It describes the equations-of-motion (or dynamical equations) of the system in matrix form. It is a $[n \times n]$ matrix, where n is the number of states.

$\vec{P}_k$ is the error covariance matrix at time k. In most cases, this is the error covariance matrix update from the previous iteration, which describes the variance (certainty) of the state vector estimate update after considering its internal state-transition model and the measured data. It is a $[n \times n]$ matrix, where n is the number of states.

$\vec{Q}_k$ is the process noise covariance matrix. It describes the variance (uncertainty) of each component in the state-transition model in matrix form. It is a $[n \times n]$ matrix, where n is the number of states.

The second step of the Kalman filter is referred to as the *update* step. In this step, the filter calculates the Kalman gain matrix, $\vec{K}$, which is a matrix that describes the weighted average of the variance between the error covariance matrix prediction $\vec{P}_{k+1}$, and the measurement noise covariance matrix $\vec{R}_k$, while taking into account the observability matrix $\vec{K}_k$. The Kalman filter then calculates a state vector estimate *update* $\hat{\vec{x}}_{k+1}$, using the Kalman gain matrix $\vec{K}_k$ and observability matrix $\vec{H}_k$ to weight the state vector estimate *prediction* $\hat{\vec{x}}'_{k+1}$ against the measurements $\vec{z}_k$. Finally, the Kalman filter calculates an error covariance matrix *update* $\hat{\vec{P}}_{k|k}$ based on the Kalman gain matrix $\vec{K}_k$, observability matrix $\vec{H}_k$, and error covariance matrix prediction $\vec{P}'_{k+1}$. The Kalman gain matrix, state vector estimate update, and error covariance matrix update equations are given in equations (5),(6), and (7), respectively.

$$\vec{K}_k = \hat{\vec{P}}_{k+1}\vec{H}_k^T\left(\vec{H}_k\hat{\vec{P}}_{k+1}\vec{H}_k^T + \vec{R}_k\right)^{-1}$$

Where

$\vec{K}_k$ is the Kalman gain matrix. It describes the relative variance (certainty) between the internal state-transition model and the measured data. It is a $[n \times n]$ matrix, where n is the number of states.

$\vec{P}_{k+1}$ is the error covariance matrix *prediction*. It describes the variance (certainty) of the state vector estimate prediction after considering its internal state-transition model, but does not consider any measured data. It is a $[n \times n]$ matrix, where n is the number of states. (5)

$\vec{H}_k$ is the observation matrix. It describes the observability of each state in matrix form. It is a $[n \times n]$ matrix, where n is the number of states.

$\vec{R}_k$ is the measurement noise covariance matrix. It describes the variance (uncertainty) of each measurement in matrix form. It is a $[n \times n]$ matrix, where n is the number of states.

$$\hat{\vec{x}}_{k+1} = \hat{\vec{x}}'_{k+1} + \vec{K}_k\left(\vec{z}_k - \vec{H}_k\hat{\vec{x}}'_{k+1}\right)$$

Where

$\hat{\vec{x}}_{k+1}$ is the state vector estimate *update*. It describes the Kalman filter's previous final estimate after considering its internal state-transition model and the measured data. It is a $[n \times 1]$ vector, where n is the number of states.

$\hat{\vec{x}}'_{k+1}$ is the state vector estimate *prediction* calculated in the predict step of the Kalman filter. It describes the prediction of the state vector estimate after considering its internal state-transition model, but does not consider any measured data. It is a $[n \times 1]$ vector, where n is the number of states. (6)

$\vec{K}_k$ is the Kalman gain matrix. It describes the relative variance (certainty) between the internal state-transition model and the measured data. It is a $[n \times n]$ matrix, where n is the number of states.

$\vec{z}_k$ is the measurement input vector. It describes the measurements of each state for that iteration of the Kalman filter. It is a $[n \times 1]$ vector, where n is the number of states.

$\vec{H}_k$ is the observation matrix. It describes the observability of each state in matrix form. It is a $[n \times n]$ matrix, where n is the number of states.

$$\hat{\vec{\vec{P}}}_{k+1} = \hat{\vec{\vec{P}}}'_{k+1} - \vec{K}_k \vec{H}_k \hat{\vec{\vec{P}}}'_{k+1} \tag{7}$$

Where

$\hat{\vec{\vec{P}}}_{k+1}$ is the error covariance matrix *update*. It describes the variance (certainty) of the state vector estimate update after considering its internal state-transition model and the measured data. It is a $[n \times n]$ matrix, where n is the number of states.

$\hat{\vec{\vec{P}}}'_{k+1}$ is the error covariance matrix *prediction* calculated in the predict step of the Kalman filter. It describes the variance (certainty) of the state vector estimate prediction after considering its internal state-transition model, but does not consider any measured data. It is a $[n \times n]$ matrix, where n is the number of states.

$\vec{K}_k$ is the Kalman gain matrix. It describes the relative variance (certainty) between the internal state-transition model and the measured data. It is a $[n \times n]$ matrix, where n is the number of states.

$\vec{H}_k$ is the observation matrix. It describes the observability of each state in matrix form. It is a $[n \times n]$ matrix, where n is the number of states.

Note that this is the condensed, or sometimes called "optimal" form of the discrete Kalman filter. For the full, sometimes called "Joseph" form, see its original derivation [8].

The state vector $\hat{\vec{x}}_{k+1}$ in equation (6) is taken as the most current state estimate, and the error covariance matrix $\hat{\vec{\vec{P}}}_{k+1}$ in equation (7) is taken as the most current error covariance matrix for that iteration of the Kalman filter. At the next iteration, $\hat{\vec{x}}_{k+1}$ becomes $\hat{\vec{x}}_k$, and $\hat{\vec{\vec{P}}}_{k+1}$ becomes $\vec{P}_k$, and the cycle continues.

2.5. Measurements

The GNSS+IMU sensor system focused on in this work gives only limited state information. In this section, I will discuss the quality of the measurements that one can expect from a relatively low-cost GNSS and IMU sensor. I am particularly interested in sensor precision (variance) and refresh rate, as these quantities are crucially important when designing a state estimator. To do this, I survey the specifications of several low cost GNSS and IMU sensors, and select reasonable values for variance and refresh rate. Typical low-cost GNSS sensors are listed in Table 1, and typical low-cost IMU sensors are listed in Table

2. Note that neither of these Tables are intended to be a comprehensive study of sensor capabilities, rather, it is a starting point for understanding the measurement covariances of the relevant states.

Table 1: Typical Low-Cost GNSS Sensors

GPS Sensor	Cost	Approx. Variance	Refresh Rate
NEO-M8P [79]	$199.95 [80]	2.5 m	1 Hz
Copernicus II DIP [81]	$75.95 [82]	2.5 m	1 Hz
EM-506 [83]	$39.95 [84]	2.5 m	1 Hz
LS20031 [85]	$69.95 [86]	3 m	1 Hz
FGPMMOPA6H [87]	$39.95 [88]	3 m	1 Hz

Note that GNSS sensors can output a linear velocity measurement; however, for low-cost GNSS sensors operating in single point precision (SPP) mode, the linear velocity is calculated by differentiating two consecutive positions which results in meter-per-second-level of accuracy at the refresh rate of the sensor which is generally not useful for navigation [74]. Higher cost GNSS systems capable of carrier-phase measurement, and a means of carrier-phase ambiguity resolution (through a variety of methods) are capable of more accurate linear velocity measurement [89]; however, these systems are not low-cost and still a very active research area [77].

Table 2: Typical Low-Cost IMU Sensors

IMU Sensor	Cost	Rate Gyroscope		Accelerometer		AHRS (magnetometer & rate gyroscope fusion)	
		Approx. Variance	Refresh	Approx. Variance	Refresh Rate	Approx. Variance	Refresh Rate
MPU-9250 [90]	$39.95 [91]	5 °/s	250 Hz	0.58 m/s^2	500 Hz		
BNO080 [92]	$34.95 [93]	3.1 °/s	1000 Hz	0.3 m/s^2	500 Hz	4.5 °	100 Hz
UM7 [94]	$139.95 [95]		255 Hz		255 Hz	5 °	255 Hz
VMU931 [96]	$99.00 [97]		1000 Hz		1000 Hz		1000 Hz

For IMU sensors, it is common practice *not* to list the precision of each individual sensor, as these values can vary quite significantly. In particular, the magnetometer measures the surrounding magnetic field, this information is then often internally fused to other measurements to estimate angular position. The variance in sensor quality, estimation algorithm, and other facts can make direct comparisons between IMU sensors difficult. The information on Table 2 is only listed if it is explicitly stated in the respective sensor datasheet.

Table 3 takes the information from Table 1 and Table 2 in the context of the nine possible states of a robot in two dimensions: three degrees-of-freedom (x, y, ψ) for position, velocity, and acceleration. Because these values will be used to simulate the performance of the Kalman filter, care was taken to choose conservative values.

Table 3: Typical of State Measurements from a Low-Cost GNSS+IMU Sensor System

State	Measuring Sensor	Approx. Variance	Refresh Rate
linear position, x	GNSS	2.5 m	1 Hz
linear position, y	GNSS	2.5 m	1 Hz
angular position, ψ	IMU AHRS	5 °	20 Hz
linear velocity, $\dot{x}$	Unmeasured		
linear velocity, $\dot{y}$	Unmeasured		
angular velocity, $\dot{\psi}$	IMU Rate Gyroscope	5 °/s	20 Hz
linear acceleration, $\ddot{x}$	IMU Accelerometer	0.6 m/s^2	20 Hz
linear acceleration, $\ddot{y}$	IMU Accelerometer	0.6 m/s^2	20 Hz
angular acceleration, $\ddot{\psi}$	Unmeasured		

2.6. The constant-acceleration state-transition model

The purpose of the state-transition model, $\vec{F}$, is used in the predict step of the Kalman filter, i.e. equations (6) and (7). It takes the state estimate from the previous iteration of the Kalman filter $\hat{\vec{x}}_{k-1|k}$, and predicts its next state $\hat{\vec{x}}_{k|k-1}$. In the following update step of the Kalman filter, this state *prediction* is then weighted against state *measurements* to form an improved state *update*. An accurate state prediction is important, as it can serve two different functions in a Kalman filter: (1) if no measurement is available, the state prediction is the only quantity informing the state estimate, and (2) if a measurement is available, the weighting the probabilistic state prediction against the probabilistic state measurement will help filter noisy values in the state measurement.

The previously-mentioned constant-acceleration state-transition model is the most commonly cited model in open-source sensor fusion implementations [69], [75], [76]. It is based on the general form the Taylor series expansion given in equation (8).

$$\begin{aligned} f(x) &= f(a) + \frac{f'(a)\cdot(x-a)}{1!} + \frac{f''(a)\cdot(x-a)^2}{2!} + \frac{f'''(a)\cdot(x-a)^3}{3!} + \cdots \\ &= \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n \end{aligned} \tag{8}$$

In the constant-acceleration state-transition model, it is common to make three assumptions. First, the expansion function directly predicts the state variable, i.e., $f(a) = \vec{x}$. Second, the state-transition model operates in the Kalman filter discrete-time domain, i.e. $f\big(a(t)\big) \approx f[k]$, with time step dt. Third, the higher order terms of the Taylor series expansion can be neglected for simplicity. With these assumptions, equation (8) can be re-written for a single state variable x in equation (1) given previously in section 2.3.5. Equation (1) is expresses the state variable x as the full state vector $\vec{x}$ in equation (9).

$$\vec{x}_{k+1} = \vec{x}_k + \dot{\vec{x}}_k \cdot dt_k$$

Where
$\vec{x}_{k+1}$ is the state vector at time $k+1$
$\vec{x}_k$ is the state vector at time k
$\dot{\vec{x}}_k$ is the derivative of the state vector at time k
dt_k is the time step for iteration k
(9)

If the robot area-of-operation can be approximated as two-dimensional, there are two linear degrees-of-freedom (x, y) and a single rotational degree of freedom ψ. For robot localization with a GNSS and IMU sensor, I am interested in estimating the robot position, velocity, and acceleration. Expanding equation (9) with the full set of state variables results in equation (10).

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \psi_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \\ \dot{\psi}_{k+1} \\ \ddot{x}_{k+1} \\ \ddot{y}_{k+1} \\ \ddot{\psi}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \dot{x}_k \cdot dt_k \\ y_k + \dot{y}_k \cdot dt_k \\ \psi_k + \dot{\psi}_k \cdot dt_k \\ \dot{x}_k + \ddot{x}_k \cdot dt_k \\ \dot{y}_k + \ddot{y}_k \cdot dt_k \\ \dot{\psi}_k + \ddot{\psi}_k \cdot dt_k \\ \ddot{x}_k + \dddot{x}_k \cdot dt_k \\ \ddot{y}_k + \dddot{y}_k \cdot dt_k \\ \ddot{\psi}_k + \dddot{\psi}_k \cdot dt_k \end{bmatrix}$$

Where (10)

x_k is the linear position of the robot
y_k is the linear position of the robot
ψ_k is the angular position of the robot
$\dot{x}_k$ is the linear velocity of the robot
$\dot{y}_k$ is the linear velocity of the robot
$\dot{\psi}_k$ is the angular velocity of the robot
$\ddot{x}_k$ is the linear acceleration of the robot
$\ddot{y}_k$ is the linear acceleration of the robot
$\ddot{\psi}_k$ is the angular acceleration of the robot
dt is the discrete time step

Each state in the model given in equation (10) requires knowledge of its past derivative. This is not a problem for the position and velocity states (which require velocity and acceleration, respectively), but the acceleration state requires an estimate of the third derivative, sometimes called the "jerk" or "lurch" derivative. This state variable is neither measured, nor estimated, the prediction of this acceleration state in this form will not work; however, the inertial forces of most robot systems are much larger than the forces that put them into motion, thus, it's a reasonable approximation to ignore the jerk/lurch effects and leave the acceleration terms constant. This assumption of constant acceleration gives this model its name, the "constant-acceleration" state-transition model. This is shown in equation (11).

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \psi_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \\ \dot{\psi}_{k+1} \\ \ddot{x}_{k+1} \\ \ddot{y}_{k+1} \\ \ddot{\psi}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \dot{x}_k \cdot dt_k \\ y_k + \dot{y}_k \cdot dt_k \\ \psi_k + \dot{\psi}_k \cdot dt_k \\ \dot{x}_k + \ddot{x}_k \cdot dt_k \\ \dot{y}_k + \ddot{y}_k \cdot dt_k \\ \dot{\psi}_k + \ddot{\psi}_k \cdot dt_k \\ \ddot{x}_k \\ \ddot{y}_k \\ \ddot{\psi}_k \end{bmatrix} \tag{11}$$

See equation (10) for variable listing

Equation (11) is written more conveniently in matrix form in equation (12).

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \psi_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \\ \dot{\psi}_{k+1} \\ \ddot{x}_{k+1} \\ \ddot{y}_{k+1} \\ \ddot{\psi}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dt_k & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & dt_k & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & dt_k & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & dt_k & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt_k & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt_k \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \\ \psi_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{\psi}_k \\ \ddot{x}_k \\ \ddot{y}_k \\ \ddot{\psi}_k \end{bmatrix} \tag{12}$$

See equation (10) for variable listing

Equation (12) takes the state vector at time k, and multiplies it by a state-transition matrix to *predict* the state vector at time $k + 1$ without the need for information about inputs to the system. It is common for robot state estimators to incorporate kinematic transforms that convert state variables from one reference frame to another, e.g. a measurement taken with respect to the robot body frame to an equivalent measurement with respect to the local map frame, or vice-versa. This transform introduces sines and cosines into the state transition matrix, making equations of motion nonlinear; however, this transformation can also be performed on the measurements before or after the state-transition prediction, which will preserve the linearity of the state-transition matrix. In fact, it behooves the filter designer to perform these transformations outside of the state-transition equations, as I will discuss in detail in section 3.1.

2.7. Shortcomings of existing state estimation algorithms for velocity-denied mobile robots

The goal of this chapter was to describe a "general" form of the robot localization state estimation algorithm used for existing robot platforms. In section 2.1, I described the importance of state estimation and sensor fusion for the task of robot localization. In section 2.2, I discussed the two preeminent probabilistic state observer methods for robot localization: the Kalman filter and the non-parametric particle filter. I learned that the particle filter models the probability distribution of the surrounding environment, whereas the Kalman filter models the probability distribution of the robot using a unimodal Gaussian distribution. The particle filter has demonstrated promising results for robot localization in unstructured environments; however, it is more applicable to extrinsic sensors that measure properties of the surrounding environment, e.g. vision, laser scans, sonar, etc. The low-cost GNSS and IMU sensors used in our work directly measure robot state variables, which is more appropriate for the Kalman filter approach. Therefore, in section 2.3, I scrutinized open-source/well-documented state estimation algorithms based on the Kalman filter for known robot platforms. From this analysis, I learned two important points:

- If maximizing dead reckoning estimation performance is a priority, a state-transition model based on a dynamical model of the robot and environmental variables is necessary. If employing absolute and/or extrinsic sensors that can reliably bound absolute pose measurement error, a state-transition model based on a kinematic motion model is more universally applicable and easier to implement. For this reason, I use a kinematic motion model in my improved state estimation algorithm.
- Nearly all existing state estimation algorithms require a generalized linear velocity sensor for smooth position estimation performance between position measurement updates; however, linear velocity sensors like wheel encoders, air speed, optical flow, or doppler velocity log sensors are only situationally applicable and/or cost prohibitive. This reinforces the need for a state estimation algorithm that is capable

of convergent pose *and* twist estimation for robots without a physical linear velocity sensor.

In section 2.4, I discussed the operation of the discrete Kalman filter, the state observer at the heart of many state estimation algorithms. Then in sections 2.5, I described the measurements generally available to a GNSS+IMU sensor system. Finally, in section 2.6, I discussed the constant-acceleration state-transition kinematic motion-tracking model used in many state estimators for robot localization.

This concludes our analysis of the underlying operation and performance of existing state estimation algorithms. The goal of subsequent chapters in this work will be propose an improved state estimation algorithm that will convergently estimate pose and twist for a velocity-denied mobile robot using only a GNSS+IMU sensor system. This improved state estimation algorithm will need to address the shortcomings outlined in this chapter.

3. An improved state estimation algorithm based on a cascading, dual-frame, motion-tracking estimator

In this chapter, I describe our improved state estimation algorithm for pose and twist estimation for a velocity-denied robot using only a GNSS+IMU sensor system. This improved state estimator is based on a discrete Kalman filter with a state-transition model based on the kinematic motion model presented in equation (3). This state estimator solves four core challenges which are discussed in detail in the following sections. They are:

- **A discrete-time interpretation of the robot body reference frame.** Sensor data from the GNSS (linear position) is measured relative to the map frame, but sensor data from the IMU (angular position, linear velocity, and linear acceleration) is measured relative to the robot body frame. To fuse these states properly, the linear position data from the GNSS should be converted to the robot body frame; however, existing reference frame nomenclature does not define linear position states in the robot body reference frame, as these states do not typically have an immediate physical interpretation. In section 3.1, I describe a discrete-time interpretation of the robot body frame state vector that will allow us to define the state variables necessary to fuse data from the GNSS and IMU in the robot body frame.
- **Adaptive covariance profiling for sensor quantization mitigation and improved estimator prediction capability.** As shown in Table 3, the GNSS and IMU have significantly different update rates, this is problematic, as the filter designer needs to decide how to incorporate measurements into the state estimation algorithm in between sensor updates. In section 3.2, I describe my implementation of covariance profiling, which involves varying the relative measurement noise covariance of the state measurements to reflect the decaying accuracy of a measurement as time elapses since it's last update. I use covariance profiling to smooth jumps in the estimator output in response to new measurements (quantization) and/or increase our estimate accuracy.

- **Unobserved linear velocity estimation based on the moving-mean forward-difference of GNSS position data, and trapezoidal method of accelerometer (IMU) data.** In section 3.3 I discuss how our state estimation algorithm creates a pseudo-measurement of the hidden linear velocity state. Because the Kalman filter treats measurements as probabilities rather than direct reflections of state, I can construct a linear velocity estimate based on the knowledge that linear velocity is related to linear position by its derivative and that linear velocity is related to linear acceleration by its integral. Using the discrete-time interpretation of the robot frame discussed in section 3.1, I take the forward-difference (discrete derivative) of the linear position data from the GNSS, and the trapezoidal method (discrete integral) of the linear acceleration data from the IMU to generate a linear velocity measurement, which is then fused into the Kalman filter with an appropriate measurement covariance.
- **The cascading, dual-frame estimator in the robot body frame and then in the map frame.** As discussed in section 3.1, the sensor measurements are fused in the robot frame, which requires that the GNSS measurements are transformed from the map frame to the robot body frame. That said, for robot localization most applications require that pose (positions) is ultimately measured with respect to the map frame. This would require that the pose output from the robot frame estimator be transformed back to the map frame; however, because each frame transformation propagates error into the state estimate proportional to the error in the angular position state estimate, this double transformation of GNSS linear position data from the map frame to the robot body and then back to the map frame is not ideal. In section 3.4 I discuss a solution to this problem which employs two cascading state estimators. The first state estimator references the robot body frame, and fuses the transformed GNSS measurement with the IMU measurements to generate a robot body frame twist estimate. The linear velocity twist estimate is then fused with the original (non-transformed) GNSS linear position estimate in a second estimator referencing the map frame for the pose states, and the robot body frame for the twist

states. This cascading state estimation method circumvents all but a single frame transformation, maximizing the accuracy of the final state estimate.

Finally, in section 3.5, I present the full, improved state estimation algorithm combining all of the concepts discussed in this. I call this algorithm the *cascading dual-frame motion-tracking state estimator for velocity-denied robot localization*.

3.1. A discrete-time interpretation of the robot body reference frame

In robot localization, I am interested in tracking the positions, velocities, and accelerations of one or more robot systems. To do this, it is convenient to assign reference frames to reference points in the area-of-operation of the robot. A reference frame consists of a coordinate system and a state vector that standardizes measurements with respect to that frame. The reference frame you define will change depending on the robot type, and its area-of-operation, and its intended purpose; some examples of reference frames include spherical frames fixed to Earth, spherical frames that rotate with Earth, cartesian frames fixed to a point on the surface of Earth, cartesian frames that are fixed to a point on the robot body frame, cartesian frames that are fixed to a sensor or actuator, etc. For the problem statement outlined in this work, I assume that the accelerations due to Earth's rotation can be safely neglected [77], [98], and the area-of-operation of the robot can be approximated as a cartesian coordinate system. Under this common assumption set, I assign two reference frames, the *map* and *robot body* frame.

3.1.1. The map and robot body frame coordinate systems

The coordinate systems for the map and robot body reference frames are illustrated in Figure 1.

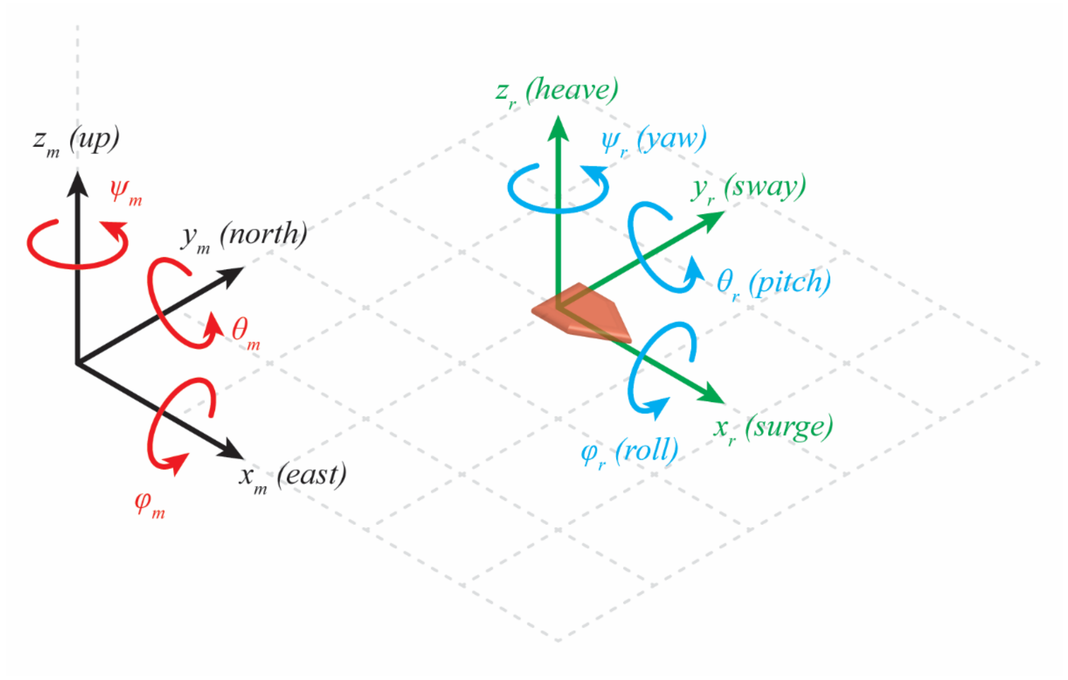

Figure 1: Map frame (left) and robot body frame (right) coordinate systems.

The map frame is an inertial reference frame that is fixed to a reference location on Earth. It has a right-handed cartesian coordinate system oriented as x-east, y-north, z-up, as recommended by the standard units of measurement in REP 103 [99]. The map frame is an inertial reference frame, because the reference frame undergoes negligible accelerations (as I assume the acceleration due to the Earth's rotation is negligible). The theory of special relativity states that the laws of physics are equivalent for all truly inertial reference frames, therefore, the map frame origin is typically located based on convenience; often a ground station, a known location on planet Earth, or the position where the robot first begins collecting data.

The robot body frame is a non-inertial reference frame that is rigidly fixed to the origin of the robot; this origin could be the center of gravity/mass, center of buoyancy, center of flotation, or a kinematic center of the robot. The robot body frame is a right-handed cartesian coordinate system oriented as x-forward, y-left, z-up, as recommended by the standard units of measurement in REP 103 [99]. This reference frame is non-inertial reference frame, because the robot undergoes non-negligible accelerations as it traverses the environment.

3.1.2. The map and robot body frame state vector

In three-dimensions, a body has three linear degrees-of-freedom, and three rotational degrees-of-freedom; thusly, I can represent the linear position and angular position of the robot using a 6×1 state vector following the form $(x, y, z, \phi, \theta, \psi)$. Representing linear position in three-dimensions is straightforward, because linear translations are commutative, i.e. the order in which the translations occur do not affect the outcome. This is demonstrated in Figure 2.

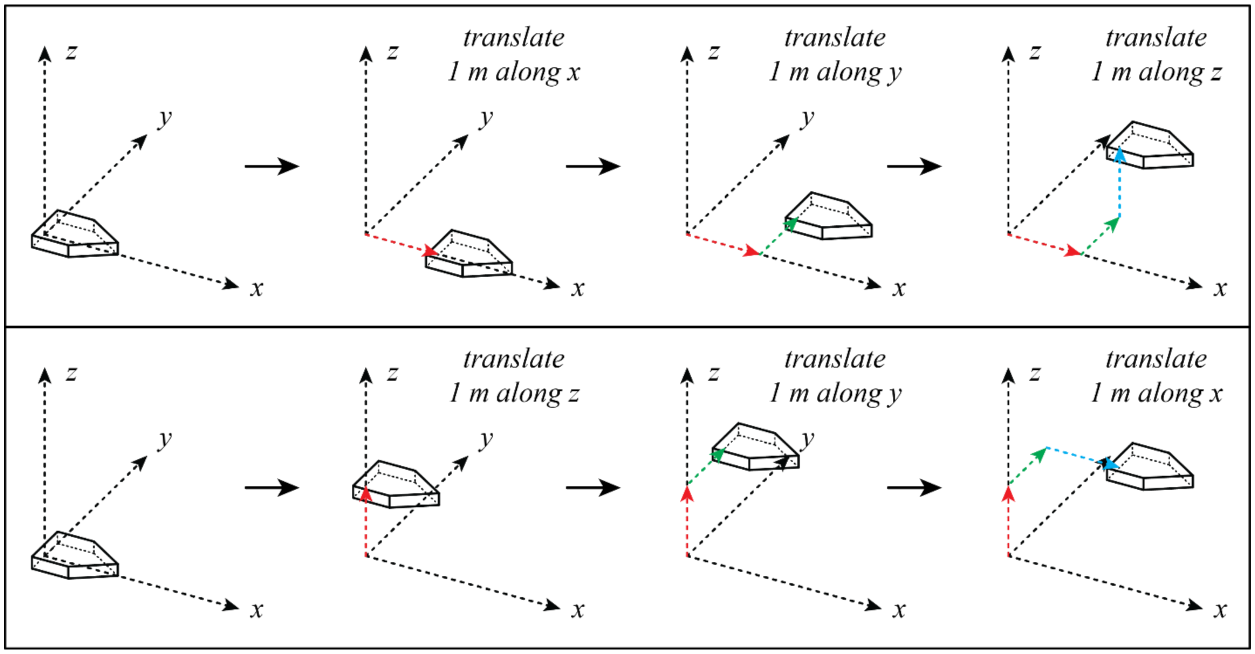

Figure 2: Translation of a body 1 m along x, 1 m along y, and then 1 m along z (top); and translation of a body 1 m along z, 1 m along y, and 1 m along x (bottom). These two different translation orderings result in the same final position because linear translations are commutative.

Unlike linear transformations, angular rotations are non-commutative, i.e. the order in which rotations occur will affect the final rotated position. This is demonstrated in Figure 3.

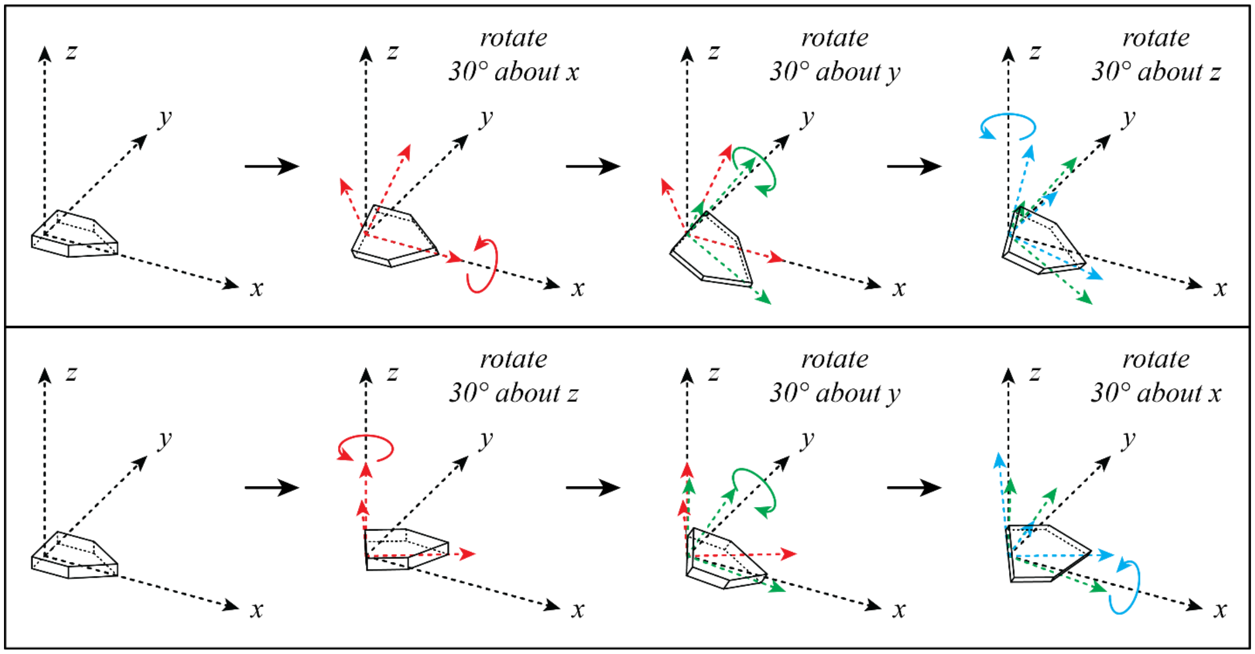

Figure 3: Rotation of a body 30° about x, 30° about y, and then 30° about z (top); and a rotation of a body 30° about z, 30° about y, and then 30° about z (bottom). These two different rotation orderings result in a different final orientation because angular rotations are non-commutative.

Because rotations in three-dimensions are non-commutative, defining the orientation of a body using three equivalent rotations about the cartesian coordinate axes—often called "Euler angle" representation—is only valid if the order of Euler angle rotations is pre-specified; there are 12 possible orderings of Euler angles in a cartesian coordinate system, referenced either to the fixed or moving axis system of the body, for a total of 24 valid Euler angle orderings [100]. This order ambiguity makes Euler angles more challenging to implement, particularly when compared to other rotation representation methods like rotation matrices, angle-axis representation, and quaternions [99]. Despite this, the mathematics for converting between the various representation methods is generally well-understood; this has resulted in Euler angles persisting as a common means of defining the angular rotation of a body-fixed coordinate system as a state vector, especially in maritime applications [77], [98], [99]. Following this convention, I describe orientation using the fixed-xyz ordered Euler angle representation in our three-dimensional state vector, and convert to rotation matrices and/or other rotation representation formats as it is mathematically appropriate to do so, i.e. any Euler angle vector representation may be assumed to employ the fixed-xyz ordering throughout this work. The mathematics for converting between fixed-xyz Euler angles, rotation matrices, and angle-axis representation is presented in Appendix 6.1.

Given the map frame and robot frame coordinate systems outlined in the previous section, and the assumption of fixed-xyz Euler angle representation, I can assign a state vector describing the positions (pose), velocities (twist), and accelerations of the robot relative to each frame. The map frame state vector is denoted by $\vec{x}_m$, and expresses the positions, velocities, and accelerations of the robot with respect to the map frame coordinate system. The robot body frame state vector is denoted by $\vec{x}_r$, and expresses the positions, velocities, and accelerations of the robot with respect to the robot body frame coordinate system. The $\vec{x}_m$ and $\vec{x}_r$ state vectors are described in Table 4.

Table 4: Map Frame and Robot Body Frame State Vector Components

Map Frame State Vector		Robot Body Frame State Vector	
$\vec{x}_m = \begin{cases} x_m \\ y_m \\ z_m \\ \phi_m \\ \theta_m \\ \psi_m \\ \dot{x}_m \\ \dot{y}_m \\ \dot{z}_m \\ \dot{\phi}_m \\ \dot{\theta}_m \\ \dot{\psi}_m \\ \ddot{x}_m \\ \ddot{y}_m \\ \ddot{z}_m \\ \ddot{\phi}_m \\ \ddot{\theta}_m \\ \ddot{\psi}_m \end{cases}$	$x_m =$ linear position along x_m axis	$\vec{x}_r = \begin{cases} x_r \\ y_r \\ z_r \\ \phi_r \\ \theta_r \\ \psi_r \\ \dot{x}_r \\ \dot{y}_r \\ \dot{z}_r \\ \dot{\phi}_r \\ \dot{\theta}_r \\ \dot{\psi}_r \\ \ddot{x}_r \\ \ddot{y}_r \\ \ddot{z}_r \\ \ddot{\phi}_r \\ \ddot{\theta}_r \\ \ddot{\psi}_r \end{cases}$	$x_r =$ linear position along x_r axis
	$y_m =$ linear position along y_m axis		$y_r =$ linear position along y_r axis
	$z_m =$ linear position along z_m axis		$z_r =$ linear position along z_r axis
	$\phi_m = xyz$-fixed Euler angle about x_m axis		$\phi_r = xyz$-fixed Euler angle about x_r axis
	$\theta_m = xyz$-fixed Euler angle about y_m axis		$\theta_r = xyz$-fixed Euler angle about y_r axis
	$\psi_m = xyz$-fixed Euler angle about z_m axis		$\psi_r = xyz$-fixed Euler angle about z_r axis
	$\dot{x}_m =$ linear velocity along x_m axis		$\dot{x}_r =$ linear velocity along x_r axis
	$\dot{y}_m =$ linear velocity along y_m axis		$\dot{y}_r =$ linear velocity along y_r axis
	$\dot{z}_m =$ linear velocity along z_m axis		$\dot{z}_r =$ linear velocity along z_r axis
	$\dot{\phi}_m =$ time derivative of ϕ_m		$\dot{\phi}_r =$ time derivative of ϕ_r
	$\dot{\theta}_m =$ time derivative of θ_m		$\dot{\theta}_r =$ time derivative of θ_r
	$\dot{\psi}_m =$ time derivative of ψ_m		$\dot{\psi}_r =$ time derivative of ψ_r
	$\ddot{x}_m =$ linear acceleration along x_m axis		$\ddot{x}_r =$ linear acceleration along x_r axis
	$\ddot{y}_m =$ linear acceleration along y_m axis		$\ddot{y}_r =$ linear acceleration along y_r axis
	$\ddot{z}_m =$ linear acceleration along z_m axis		$\ddot{z}_r =$ linear acceleration along z_r axis
	$\ddot{\phi}_m =$ time derivative of $\dot{\phi}_m$		$\ddot{\phi}_r =$ time derivative of $\dot{\phi}_r$

	$\ddot{\theta}_m$ = time derivative of $\dot{\theta}_m$		$\ddot{\theta}_r$ = time derivative of $\dot{\theta}_r$
	$\ddot{\psi}_m$ = time derivative of $\dot{\psi}_m$		$\ddot{\psi}_r$ = time derivative of $\dot{\psi}_r$

This interpretation of the map and robot body frame is very straightforward, but mathematically problematic: **because the robot body frame is fixed to the robot, the robot cannot have a position relative to the robot body frame at that same point in time,** i.e. ($x_r = 0, y_r = 0, z_r = 0, \phi_r = 0, \theta_r = 0, \psi_r = 0$). By extension, the successive velocity and acceleration derivatives of the robot in the robot body frame must also be zero, since a time-varying position must be zero if the position is constantly zero. Because all the robot body frame state vector components constantly evaluate to zero, this interpretation of the robot body frame is clearly not useful! I refer to this problem as the *zero-state robot body frame problem.*

3.1.3. Existing nomenclature's solution to the zero-state robot body frame problem

To circumvent the zero-state robot body frame problem, existing nomenclature is very strategic in its definition of the robot body frame state vector components. The general solution is to omit the position components of the robot in the robot frame while keeping the velocity and acceleration components. Eliminating the position components creates a mathematical degeneracy between the state vector derivatives (a zero and/or constant position cannot have a non-zero velocity derivative); however, this degeneracy is addressed in the nomenclature described by Fossen by describing the robot body frame velocity components with respect to the map frame, but expressed in the body frame [77], i.e. the velocities do not strictly exist in the robot body frame, but the velocities do exist in the map frame, and can be expressed in the robot body frame coordinate system.

For clarity, the state vector component nomenclature defined by SNAME [98] is shown in Table 5, and the nomenclature later defined by Fossen [77] is shown in Table 6. The frame names and their respective state vector component definitions are reproduced as faithfully as possible in context. To illustrate which state vector components are omitted from the SNAME and Fossen nomenclature, they are written in the same form as our general state vector definition given in Table 1, leaving a "☒" symbol where specific state vector components were omitted.

Table 5: Inertial Fixed Axes Frame and Body Frame State Vector Components Defined by SNAME [98]

Fixed Axes Frame State Vector		Body Frame State Vector	
$\vec{x}_0 = \left\{\begin{matrix} x_0 \\ y_0 \\ z_0 \\ \phi \\ \theta \\ \psi \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \end{matrix}\right.$	x_0 = linear position along x_0 axis; x_0 is the fixed longitudinal axis, in a fixed direction in a horizontal plane, considered as the forward direction	$\vec{x}_b = \left\{\begin{matrix} \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ u \\ v \\ w \\ p \\ q \\ r \\ X \\ Y \\ Z \\ K \\ M \\ N \end{matrix}\right.$	*body frame linear position along x axis not explicitly defined*; x is the longitudinal axis of body (aft-to-fore)
	y_0 = linear position along y_0 axis; y_0 is the fixed transverse axis, perpendicular to x_0 in a horizontal plane, directed to starboard		*body frame linear position along y axis not explicitly defined*; y is the transverse axis of body (port-to-starboard)
	z_0 = linear position along z_0 axis; z_0 is the fixed vertical axis, directed downwards		*body frame linear position along z axis not explicitly defined*; z is the normal axis of body (top-to-bottom/deck-to-keel)
	ϕ = angle of roll or heel; the angle from the vertical xz_0-plane to the principal plane of symmetry xz, positive in the positive sense of rotation about the x-axis		*body frame angular position about x not explicitly defined*
	θ = angle of pitch or trim; the angle of elevation of the x-axis; i.e. the angle between Ox and the horizontal plane x_0y_0, positive in the sense of rotation from the z_0 to the x-axis		*body frame angular position about y not explicitly defined*
	ψ = angle of yaw; the angle from the vertical plane z_0x_0 to the vertical plane z_0x, positive in the positive sense of rotation about the z_0 axis		*body frame angular position about z not explicitly defined*
	map frame linear velocity along x_0 not explicitly defined		u = component in the x coordinate system of the linear velocity of the origin of the body axes relative to the fluid
	map frame linear velocity along y_0 not explicitly defined		v = component in the y coordinate system of the linear velocity of the origin of the body axes relative to the fluid
	map frame linear velocity along z_0 not explicitly defined		w = component in the z coordinate system of the linear velocity of the origin of the body axes relative to the fluid
	$\dot{\phi}$ = time derivative of ϕ		p = angular velocity component relative to body axis x; angular velocity of roll
	$\dot{\theta}$ = time derivative of θ		q = angular velocity component relative to body axis y; angular velocity of roll

	$\dot{\psi}$ = time derivative of ψ		r = angular velocity component relative to body axis z; angular velocity of roll
	map frame linear acceleration along x_0 *not explicitly defined*		X = hydrodymamic force components relative to longitudinal body frame x axis
	map frame linear acceleration along y_0 *not explicitly defined*		Y = hydrodymamic force components relative to lateral body frame y axis
	map frame linear acceleration along z_0 *not explicitly defined*		Z = hydrodymamic force components relative to normal body frame z axis
	time derivative of $\dot{\phi}$ *not explicitly defined*		K = hydrodynamic moment component relative to rolling body frame x axis
	time derivative of $\dot{\theta}$ *not explicitly defined*		M = hydrodynamic moment component relative to pitching body frame y axis
	time derivative of $\dot{\psi}$ *not explicitly defined*		N = hydrodynamic moment component relative to yawing body frame z axis

Table 6: Inertial NED (North-East-Down) Frame and BODY Frame State Vector Components Defined by Fossen [77]

NED Frame State Vector		BODY Frame State Vector	
$\vec{x}_n = \begin{cases} N \\ E \\ D \\ \phi \\ \theta \\ \psi \\ \dot{N} \\ \dot{E} \\ \dot{D} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \end{cases}$	N = linear position along x axis; x is the fixed axis pointing towards true North	$\vec{x}_b = \begin{cases} \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ \boxtimes \\ u \\ v \\ w \\ p \\ q \\ r \\ X \\ Y \\ Z \\ K \\ M \\ N \end{cases}$	*body frame linear position along* x_b *not explicitly defined*; x_b is the longitudinal axis of the craft body (directed from aft to fore)
	E = linear position along y axis; y is the fixed axis pointing towards East.		*body frame linear position along* y_b *not explicitly defined*; y_b is the transversal axis of the craft body (directed to starboard)
	D = linear position along the z axis; z is the fixed axis pointing down.		*body frame linear position along* z_b *not explicitly defined*; z_b is the normal axis of the craft body (directed from top to bottom)
	ϕ = Euler angle between x (in NED frame) and x_b (in BODY frame)		*body frame angular position about* x_b *not explicitly defined*
	θ = Euler angle between y (in NED frame) and y_b (in BODY frame)		*body frame angular position about* y_b *not explicitly defined*
	ψ = Euler angle between z (in NED frame) and z_b (in BODY frame)		*body frame angular position about* z_b *not explicitly defined*
	$\dot{N}$ = time derivative of N		u = linear velocity of the origin point of the body frame along x_b with respect to the NED frame expressed in the BODY frame.
	$\dot{E}$ = time derivative of E		v = linear velocity of the origin point of the body frame along y_b with respect to the NED frame expressed in the BODY frame.
	$\dot{D}$ = time derivative of D		w = linear velocity of the origin point of the body frame along z_b with respect

			to the NED frame expressed in the BODY frame.
	$\dot{\phi}$ = time derivative of ϕ		p = angular velocity of the BODY frame about x_b with respect to the NED frame, expressed in the BODY frame
	$\dot{\theta}$ = time derivative of θ		q = angular velocity of the BODY frame about y_b with respect to the NED frame, expressed in the BODY frame
	$\dot{\psi}$ = time derivative of ψ		r = angular velocity of the BODY frame about z_b with respect to the NED frame, expressed in the BODY frame
	map frame linear acceleration along x not explicitly defined		X = force with line of action through the origin point of the body frame along x_b, expressed in the BODY frame
	map frame linear acceleration along y not explicitly defined		Y = force with line of action through the origin point of the body frame along y_b, expressed in the BODY frame
	map frame linear acceleration along z not explicitly defined		Z = force with line of action through the origin point of the body frame along z_b, expressed in the BODY frame
	time derivative of $\dot{\phi}$ not explicitly defined		K = moment about the origin point of the body frame about x_b, expressed in the BODY frame
	time derivative of $\dot{\theta}$ not explicitly defined		M = moment about the origin point of the body frame about y_b, expressed in the BODY frame
	time derivative of $\dot{\psi}$ not explicitly defined		N = moment about the origin point of the body frame about z_b, expressed in the BODY frame

Firstly, the SNAME and Fossen nomenclatures are fundamentally similar, despite using slightly different variable names. One minor difference between the two nomenclatures is that the Fossen nomenclature defines linear and angular velocity terms in the map frame $(\dot{N}, \dot{E}, \dot{D}, \dot{\phi}, \dot{\theta}, \dot{\psi})$, whereas the SNAME nomenclature only defines angular velocity terms in the map frame $(\dot{\phi}, \dot{\theta}, \dot{\psi})$. It is not explicitly stated why the map frame linear velocity terms were omitted from the SNAME nomenclature; however, it is reasonable to assume that they were omitted because they are generally not useful, and not as an indication that they do not exist. Note that the purpose of both the SNAME and Fossen nomenclatures are to define a practical means of describing the motion of a body through a fluid, not to describe all possible state vector components and transformations of a generalized mechanical system.

Secondly, both the SNAME and Fossen nomenclature define their robot body frame accelerations in terms of forces and moments (X, Y, Z, K, M, N), rather than of accelerations and angular accelerations. Newton's second law tells us that a mass subjected to an acceleration field will impart a force equal to the mass times the acceleration field, $F = ma$. In the SNAME and Fossen derivations of equations of motion, their respective acceleration components follow the intuitive dot notation $(\dot{u}, \dot{v}, \dot{w}, \dot{p}, \dot{q}, \dot{r})$; therefore, the forces (X, Y, Z, K, M, N) can be assumed to describe the same general phenomena as $(\ddot{x}_r, \ddot{y}_r, \ddot{z}_r, \ddot{\phi}_r, \ddot{\theta}_r, \ddot{\psi}_r)$ in our general nomenclature in Table 4.

Thirdly, both SNAME and Fossen employ a North-East-Down coordinate system in their versions of the map frame (fixed axes and NED frame in the SNAME and Fossen nomenclature, respectively), whereas I use an East-North-Up coordinate system. These coordinate systems are illustrated in Figure 4.

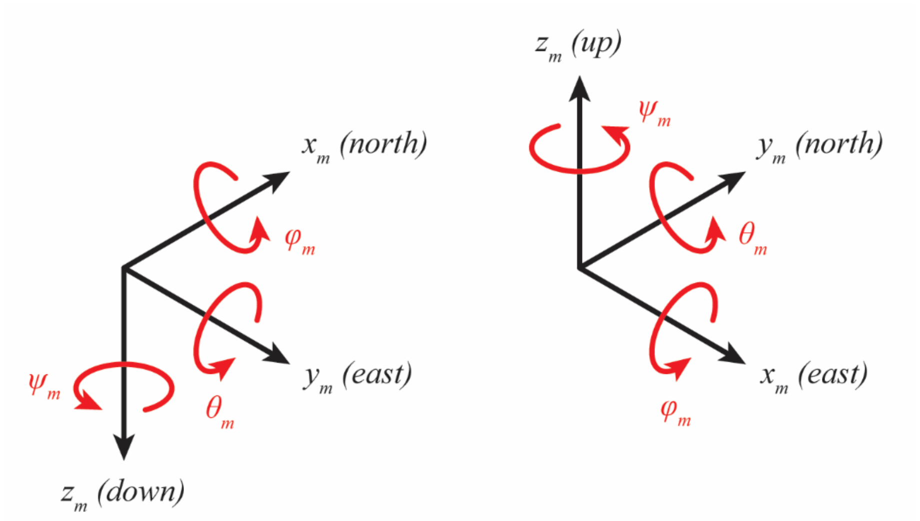

Figure 4: North-East-Down (left) versus East-North-Up (right)

Because North-East-Down and East-North-Up are both right-handed coordinate systems, they are equivalent coordinate system representations. North-East-Down is popular for maritime systems, where surface and underwater vehicles are common; however, East-North-Up is recommended by REP 103 [99], and is the representation used in this work.

Lastly, and most importantly, both SNAME and Fossen nomenclatures omit the position terms in their equivalent robot body frame. In addition, both nomenclatures employ Euler angle representation in the map frame, and angular velocity representation in the robot body frame; these two forms of orientation representation require a fairly complex transformation to convert between (described in appendix 6.2), in addition to any translation and orientation transformation that may already exist. In [2], Fossen states "It should be noted that the angular body velocity vector $\omega_{b/n}^{b} = [p, q, r]^T$ cannot be integrated directly to obtain actual angular coordinates. This is due to the fact that $\int_o^t \omega_{\frac{b}{n}}^{b}(\tau) d\tau$ does not have any immediate physical interpretation; however, the vector $\Theta_{\text{nb}} = [\phi, \theta, \psi]^T$ does represent proper generalized coordinates.", i.e. the angular velocity of the robot body frame cannot be integrated because this quantity has no physical interpretation, but the Euler angle time derivatives in the map frame can. If I were to take the integrals of the twist terms in the robot body frame following Fossen's nomenclature, it would take the awkward form $(\int u\, dt, \int v\, dt, \int w\, dt, \int p\, dt, \int q\, dt, \int r\, dt)$. For most applications involving describing the motion of a body though a fluid, the SNAME and Fossen nomenclatures are sufficient; however, defining only position and velocity in the map frame, and velocity and force in the robot body frame, while employing different orientation representations in each frame, does not lend itself well for the purposes of state estimation and sensor fusion. In sensor fusion, I need the flexibility to position, velocity, and acceleration data from sensors in the map and robot body frames, and often in different discrete time domains. This necessitates a state vector nomenclature that (1) that defines state vector components for position, velocity, and acceleration, (2) employs a consistent orientation specification for both reference frames, and (3) addresses the zero-state robot body frame problem described earlier.

3.1.4. <u>The two-dimensional discrete-time reference frame</u>

As discussed previously, it is impossible for a robot to have a non-zero position (and velocity and acceleration by extension) in a robot body frame fixed to its own origin at that same point in time (what I call the "zero-state robot body frame problem"); however, this statement is not necessarily true for a robot pose defined with respect to a coordinate frame from a previous point in time. When working in discrete-time, we can conveniently define the position of the robot at time k with respect to the robot body frame coordinate system at time $k-1$. This is illustrated in two dimensions in Figure 5.

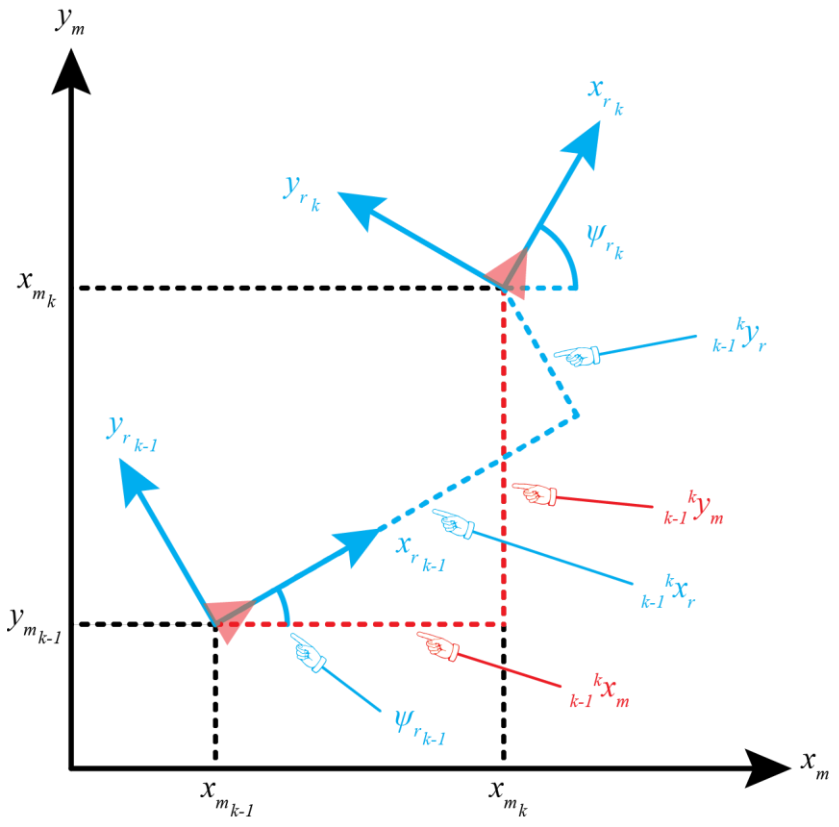

Figure 5: Example two-dimensional map frame, and robot frame coordinate systems in discrete-time.

At time k, the change in robot body frame states are measured with respect to the coordinate system at time $k-1$. Summing (integrating) these change in positions over time allows us to collect a time history of position and velocity; this can then be used to calculate sucessive derivatives as desired. A discrete-time integrator for approximating the state vector in the robot frame and map frame equation (13).

$$\vec{x}_{r_k} = \begin{bmatrix} x_{r_k} \\ y_{r_k} \\ \psi_{r_k} \\ \dot{x}_{r_k} \\ \dot{y}_{r_k} \\ \dot{\psi}_{r_k} \\ \ddot{x}_{r_k} \\ \ddot{y}_{r_k} \\ \ddot{\psi}_{r_k} \end{bmatrix} \approx \begin{Bmatrix} x_{r_{k-1}} + \dot{x}_{r_{k-1}} \cdot dt \\ y_{r_{k-1}} + \dot{y}_{r_{k-1}} \cdot dt \\ \psi_{r_{k-1}} + \dot{\psi}_{r_{k-1}} \cdot dt \\ \dot{x}_{r_{k-1}} + \ddot{x}_{r_{k-1}} \cdot dt \\ \dot{y}_{r_{k-1}} + \ddot{y}_{r_{k-1}} \cdot dt \\ \dot{\psi}_{r_{k-1}} + \ddot{\psi}_{r_{k-1}} \cdot dt \\ \ddot{x}_{r_{k-1}} + \dddot{x}_{r_{k-1}} \cdot dt \\ \ddot{y}_{r_{k-1}} + \dddot{y}_{r_{k-1}} \cdot dt \\ \ddot{\psi}_{r_{k-1}} + \dddot{\psi}_{r_{k-1}} \cdot dt \end{Bmatrix} = \begin{Bmatrix} x_{r_{k-1}} + {}^{k}_{k-1}x_r \\ y_{r_{k-1}} + {}^{k}_{k-1}y_r \\ \psi_{r_{k-1}} + {}^{k}_{k-1}\psi_r \\ \dot{x}_{r_{k-1}} + {}^{k}_{k-1}\dot{x}_r \\ \dot{y}_{r_{k-1}} + {}^{k}_{k-1}\dot{y}_r \\ \dot{\psi}_{r_{k-1}} + {}^{k}_{k-1}\dot{\psi}_r \\ \ddot{x}_{r_{k-1}} + {}^{k}_{k-1}\ddot{x}_r \\ \ddot{y}_{r_{k-1}} + {}^{k}_{k-1}\ddot{y}_r \\ \ddot{\psi}_{r_{k-1}} + {}^{k}_{k-1}\ddot{\psi}_r \end{Bmatrix}$$

where

$\vec{x}_{r_k}$ is the robot body frame state vector at time k

x_{r_k} is the linear position of the robot, measured along the robot body frame coordinate system $x_{r_{k-1}}$ axis

y_{r_k} is the linear position of the robot, measured along the robot body frame coordinate system $y_{r_{k-1}}$ axis (13)

ψ_{r_k} is the angular position of the robot, measured about the robot body frame coordinate system $z_{r_{k-1}}$ axis

$\dot{x}_{r_k}$ is the linear velocity of the robot, measured along the robot body frame coordinate system $x_{r_{k-1}}$ axis

$\dot{y}_{r_k}$ is the linear velocity of the robot, measured along the robot body frame coordinate system $y_{r_{k-1}}$ axis

$\dot{\psi}_{r_k}$ is the angular velocity of the robot, measured about the robot body frame coordinate system $z_{r_{k-1}}$ axis

$\ddot{x}_{r_k}$ is the linear acceleration of the robot, measured along the robot body frame coordinate system $x_{r_{k-1}}$ axis

$\ddot{y}_{r_k}$ is the linear acceleration of the robot, measured along the robot body frame coordinate system $y_{r_{k-1}}$ axis

$\ddot{\psi}_{r_k}$ is the angular acceleration of the robot, measured about the robot body frame coordinate system $z_{r_{k-1}}$ axis

dt is the discrete time step

In equation (13), the product of the finite time step dt and the derivative of the respective state takes on small, but finite values as the robot moves through space; summing (integrating) these values to the previous state vector component results in the approximation of the robot motion over time. Conversely, when $dt \to 0$, the product of dt and the derivative of the respective state evaluates to zero, i.e. no change in each state vector component for all time. Equation (13) confirms the zero-state robot body frame problem, where all states are constantly zero when measured with respect to the current-time robot body frame; and demonstrates that the discrete-time interpretation of the robot body frame can approximate robot motion by referencing the robot body frame coordinate system at discrete-time $k-1$. This discrete-time interpretation of the robot frame allows us to relate the position, velocity, and acceleration states of the robot with respect to the robot body frame to each other using a set of ordinary differential equations. Because the sensor fusion process utilizes real (discrete) sensor data and a discrete observer (a discrete Kalman filter in our case), this discrete-time robot frame approximation is very appropriate for our purposes. Equation (13) is simplified for two dimensions; however, the same conclusions extend to the three-dimensional discrete-time state vector proposed in Table 4.

Additionally, while the inertial map frame does not demonstrate the same zero-state frame problem present in the robot body frame, there is still value in defining the map frame in discrete-time. This is because fusing information from different frames (which will occur in our GNSS+IMU sensor system), it is convenient to evaluate the state vectors in both reference frames in the same discrete-time domain. The discrete-time interpretation of the map frame is given in equation (14).

$$\vec{x}_{m_k} = \begin{bmatrix} x_{m_k} \\ y_{m_k} \\ \psi_{m_k} \\ \dot{x}_{m_k} \\ \dot{y}_{m_k} \\ \dot{\psi}_{m_k} \\ \ddot{x}_{m_k} \\ \ddot{y}_{m_k} \\ \ddot{\psi}_{m_k} \end{bmatrix} \approx \begin{Bmatrix} x_{m_{k-1}} + \dot{x}_{m_{k-1}} \cdot dt \\ y_{m_{k-1}} + \dot{y}_{m_{k-1}} \cdot dt \\ \psi_{m_{k-1}} + \dot{\psi}_{m_{k-1}} \cdot dt \\ \dot{x}_{m_{k-1}} + \ddot{x}_{m_{k-1}} \cdot dt \\ \dot{y}_{m_{k-1}} + \ddot{y}_{m_{k-1}} \cdot dt \\ \dot{\psi}_{m_{k-1}} + \ddot{\psi}_{m_{k-1}} \cdot dt \\ \ddot{x}_{m_{k-1}} + \dddot{x}_{m_{k-1}} \cdot dt \\ \ddot{y}_{m_{k-1}} + \dddot{y}_{m_{k-1}} \cdot dt \\ \ddot{\psi}_{m_{k-1}} + \dddot{\psi}_{m_{k-1}} \cdot dt \end{Bmatrix} = \begin{Bmatrix} x_{m_{k-1}} + {}^{k}_{k-1}x_m \\ y_{m_{k-1}} + {}^{k}_{k-1}y_m \\ \psi_{m_{k-1}} + {}^{k}_{k-1}\psi_m \\ \dot{x}_{m_{k-1}} + {}^{k}_{k-1}\dot{x}_m \\ \dot{y}_{m_{k-1}} + {}^{k}_{k-1}\dot{y}_m \\ \dot{\psi}_{m_{k-1}} + {}^{k}_{k-1}\dot{\psi}_m \\ \ddot{x}_{m_{k-1}} + {}^{k}_{k-1}\ddot{x}_m \\ \ddot{y}_{m_{k-1}} + {}^{k}_{k-1}\ddot{y}_m \\ \ddot{\psi}_{m_{k-1}} + {}^{k}_{k-1}\ddot{\psi}_m \end{Bmatrix}$$

where

$\vec{x}_{m_k}$ is the map frame state vector at time k

x_{m_k} is the linear position of the robot, measured along the map frame coordinate system $x_{m_{k-1}}$ axis (14)

y_{m_k} is the linear position of the robot, measured along the map frame coordinate system $y_{m_{k-1}}$ axis

ψ_{m_k} is the angular position of the robot, measured about the map frame coordinate system $z_{m_{k-1}}$ axis

$\dot{x}_{m_k}$ is the linear velocity of the robot measured along the map frame coordinate system $x_{m_{k-1}}$ axis

$\dot{y}_{m_k}$ is the linear velocity of the robot, measured along the map frame coordinate system $y_{m_{k-1}}$ axis

$\dot{\psi}_{m_k}$ is the angular velocity of the robot, measured about the map frame coordinate system $z_{m_{k-1}}$ axis

$\ddot{x}_{m_k}$ is the linear acceleration of the robot, measured along the map frame coordinate system $x_{m_{k-1}}$ axis

$\ddot{y}_{m_k}$ is the linear acceleration of the robot, measured along the map frame coordinate system $y_{m_{k-1}}$ axis

$\ddot{\psi}_{m_k}$ is the angular acceleration of the robot, measured about the map frame coordinate system $z_{m_{k-1}}$ axis

dt is the discrete time step

One should also note that in two-dimensions, the specification of angular orientation and its successive derivatives is significantly simplified, since only a single rotation axis exists (the non-commutative property of rotations discussed in Figure 3 does not matter when only a single rotational degree-of-freedom exists), i.e. angular velocity $\dot{\psi}$ is exactly equal to the time derivative of the angular position $\frac{d}{dt}(\psi)$. If this same process were followed in three-dimensions, relating three-dimensional angular orientation representations to their respective three-dimensional angular velocity representations will need to follow the process outlined in appendix 6.2.

Another non-intuitive side-effect of the discrete-time robot body frame in two dimensions is related to the angular position, velocity, and acceleration of the robot. The linear positions in the map frame (x_m, y_m) differ from the linear positions in the robot frame (x_r, y_r) because the linear positions of the robot with respect to the robot body frame are coupled to the angle of the robot with respect to the map frame; however, the angular positon is not coupled to any other state vector component. Therefore, any change in angular position in the discrete-time map frame is identical to a change in angular position in the discrete-time robot frame. *If the map frame and robot frame can be assumed to operate in the same discrete-time domain, and the angular state vector components in the map frame are initialized to the same value, then they will be equivalent for all discrete-time*, i.e. $\psi_m = \psi_r$ and $\dot{\psi}_m = \dot{\psi}_r$ for all time provided that $\psi_{m_0} = \psi_{r_0}$. Note that this *does not* mean that the angular state vector components in the map frame *must* be equal to the angular state vector components in the robot frame, rather that they *can* be conveniently assumed to be equal if $\psi_{m_0} = \psi_{r_0}$.

3.1.5. Transforming pose between discrete-time map and robot body frames

During the state estimation process I need the ability to convert a pose transformation in the map frame, to a pose transformation in the robot body frame, and vice-versa. In our context, a pose *transformation* consists of a pose vector at time $k-1$, $(x_{k-1}, y_{k-1}, \psi_{k-1})$, and pose vector at time k, (x_k, y_k, ψ_k). An example pose transformation is shown in Figure 6.

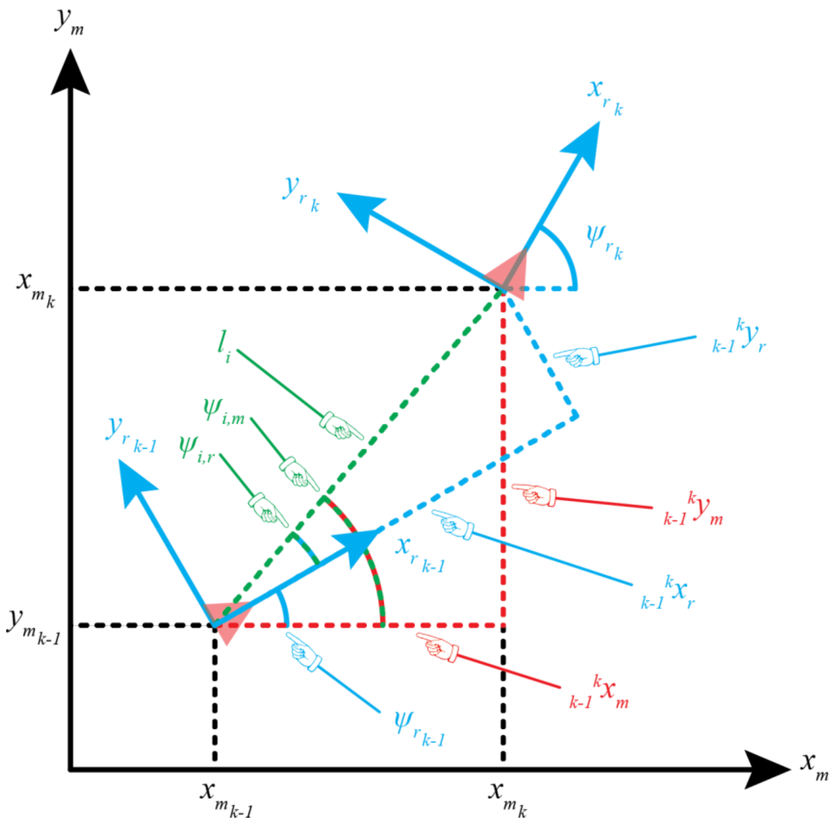

Figure 6: Illustrated pose transformation. The map frame coordinate system is colored black, and the robot body frames are colored blue.

When converting a pose transformation from the map frame to the robot body frame, I aim to solve the problem outlined in equation (15).

Given:

$$\begin{aligned}
&\text{map frame pose at time } k-1: \begin{cases} x_{m_{k-1}} \\ y_{m_{k-1}} \\ \psi_{m_{k-1}} \end{cases} \\
&\text{map frame pose at time } k: \begin{cases} x_{m_k} \\ y_{m_k} \\ \psi_{m_k} \end{cases} \\
&\text{robot body frame pose at time } k-1: \begin{cases} x_{r_{k-1}} \\ y_{r_{k-1}} \\ \psi_{r_{k-1}} \end{cases}
\end{aligned} \tag{15}$$

Solving for:

$$\text{robot body frame pose at time } k: \begin{cases} x_{r_k} \\ y_{r_k} \\ \psi_{r_k} \end{cases}$$

First, I solve for the transition vector between the two map frame states, $\left({}_{k-1}^{\mathrm{k}}x_m, {}_{k-1}^{k}y_m\right)$. This is calculated in equation (16).

$$\begin{cases} {}_{k-1}^{k}x_m = x_{m_k} - x_{m_{k-1}} \\ {}_{k-1}^{k}y_m = y_{m_k} - y_{m_{k-1}} \end{cases} \tag{16}$$

I can then calculate the magnitude, l_i, and direction, ψ_i, of the transition vector, which are shared between the map and robot body frame pose transition representations. This is given in equation (17).

$$\begin{cases} l_i = \sqrt{{}_{k-1}^{k}x_m{}^2 + {}_{k-1}^{k}y_m{}^2} \\ \psi_{i_m} = \tan^{-1}\left(\dfrac{y_{m_k} - y_{m_{k-1}}}{x_{m_k} - x_{m_{k-1}}}\right) = \operatorname{atan2}\left(y_{m_k} - y_{m_{k-1}}, x_{m_k} - x_{m_{k-1}}\right) \end{cases} \tag{17}$$

I can then calculate the transition vector between the two robot body frame states $\left({}_{k-1}^{\mathrm{k}}x_r, {}_{k-1}^{k}y_r\right)$. This is calculated in equation (18).

$$\begin{cases} {}_{k-1}^{k}x_r = \cos\left(\psi_{i_m} - \psi_{m_k}\right) \cdot l_i \\ {}_{k-1}^{k}y_r = \sin\left(\psi_{i_m} - \psi_{m_k}\right) \cdot l_i \end{cases} \tag{18}$$

Finally, the robot body frame components can be calculated from equation (19).

$$\begin{cases} x_{r_k} = x_{r_{k-1}} + {}_{k-1}^{k}x_r \\ y_{r_k} = y_{r_{k-1}} + {}_{k-1}^{k}y_r \\ \psi_{r_k} = \psi_{m_k} \end{cases} \tag{19}$$

When converting a pose transformation from the robot body frame to the map frame, I aim to solve the problem outlined in (20).

$$\begin{aligned} &\text{Given:} \\ &\quad \text{robot body frame pose at time } k-1\text{:} \begin{cases} x_{r_{k-1}} \\ y_{r_{k-1}} \\ \psi_{r_{k-1}} \end{cases} \\ &\quad \text{robot body frame pose at time } k\text{:} \begin{cases} x_{r_k} \\ y_{r_k} \\ \psi_{r_k} \end{cases} \\ &\quad \text{map frame pose at time } k-1\text{:} \begin{cases} x_{m_{k-1}} \\ y_{m_{k-1}} \\ \psi_{m_{k-1}} \end{cases} \\ &\text{Solving for:} \\ &\quad \text{map frame pose at time } k\text{:} \begin{cases} x_{m_k} \\ y_{m_k} \\ \psi_{m_k} \end{cases} \end{aligned} \tag{20}$$

First, I solve for the transition vector between the two robot body frame states, $\left({}_{k-1}^{\mathrm{k}}x_r, {}_{k-1}^{k}y_r\right)$. This is calculated in equation (21).

$$\begin{cases} {}_{k-1}^{k}x_r = x_{r_k} - x_{r_{k-1}} \\ {}_{k-1}^{k}y_r = y_{r_k} - y_{r_{k-1}} \end{cases} \tag{21}$$

I can then calculate the magnitude, l_i, and direction, ψ_i, of the transition vector, which are shared between the map and robot body frame pose transition representations. This is given in equation (22).

$$\begin{cases} l_i = \sqrt{{}_{k-1}^{k}x_r^{\,2} + {}_{k-1}^{k}y_r^{\,2}} \\ \psi_{i_r} = \tan^{-1}\left(\dfrac{y_{r_k} - y_{r_{k-1}}}{x_{r_k} - x_{r_{k-1}}}\right) = \ \operatorname{atan2}\left(y_{r_k} - y_{r_{k-1}}, x_{r_k} - x_{r_{k-1}}\right) \end{cases} \tag{22}$$

I can then calculate the transition vector between the two map frame states $\left({}_{k-1}^{\mathrm{k}}x_m, {}_{k-1}^{k}y_m\right)$. This is calculated in equation (23).

$$\begin{cases} {}_{k-1}^{k}x_m = \cos\left(\psi_{i_r} + \psi_{r_k}\right) \cdot l_i \\ {}_{k-1}^{k}y_m = \sin\left(\psi_{i_r} + \psi_{r_k}\right) \cdot l_i \end{cases} \tag{23}$$

Finally, the map frame components can be calculated from (24).

$$\begin{cases} x_{m_k} = x_{m_{k-1}} + {}_{k-1}^{k}x_m \\ y_{m_k} = y_{m_{k-1}} + {}_{k-1}^{k}y_m \\ \psi_{m_k} = \psi_{r_k} \end{cases} \tag{24}$$

3.1.6. Concluding remarks

In this section, I defined the coordinate systems for the inertial map frame (x-east, y-north, z-up) and the non-inertial robot body frame (x-forward, y-left, z-up) for robot localization. I then defined a generalized state vector that defines a linear position, velocity, and acceleration along each axis, and a rotational position, velocity, and acceleration about each axis for both the map frame and robot body frame in Table 4. Defining the position states and all of its derivatives is necessary for fusion of sensor data across derivatives; however, this frame definition is problematic for the robot body frame, whose position state vector components evaluate to of zero for all continuous-time, because the robot body frame moves with the robot. To address this problem, I propose an alternative definition of the robot body frame state vector in discrete-time. In discrete-time, the position of the robot at time k is located relative to the coordinate system of the robot at time $k-1$; this allows the discrete-time robot body frame to approximate the non-zero positions, velocities, and accelerations of the robot relative to its own coordinate system. These approximations increase in accuracy as the discrete time step decreases. This discrete-time definition of the robot body frame is now more useful for state estimation and sensor fusion and can be defined alongside the conventional (continuous-time) definitions of the robot body frame proposed by SNAME [1] and Fossen [2] in Table 5 and Table 6, respectively. I also described the framework for converting pose transformations in the map frame to equivalent pose transformations in the robot frame, and vice-versa.

3.2. Sliding adaptive covariance profiling for sensor quantization mitigation and improved estimator prediction capability

The digital nature of computer-interpreted sensors results in signal quantization, since all sensor systems have a finite resolution and refresh rate. These measurements, which can be described as "discrete-time" inputs, do not necessarily guarantee observability for the state observer due to their discrete nature [101]. Sensor quantization is an important consideration for the performance of the Kalman filter, which typically computer-implemented, and increments quickly enough that it can be approximated as a continuous-time observer relative to the sensor update rate [102]. If implemented improperly the Kalman filter can be "fooled" into overconfidence that the state is stationary, even though the state *is* changing, but the sensor is between update intervals and does not sense the change. When the sensor updates and reflects a quantized change in state, the Kalman filter believes that the change occurred more rapidly than reality, leading to a "stairstep"-like estimator quality. This is demonstrated in Figure 7.

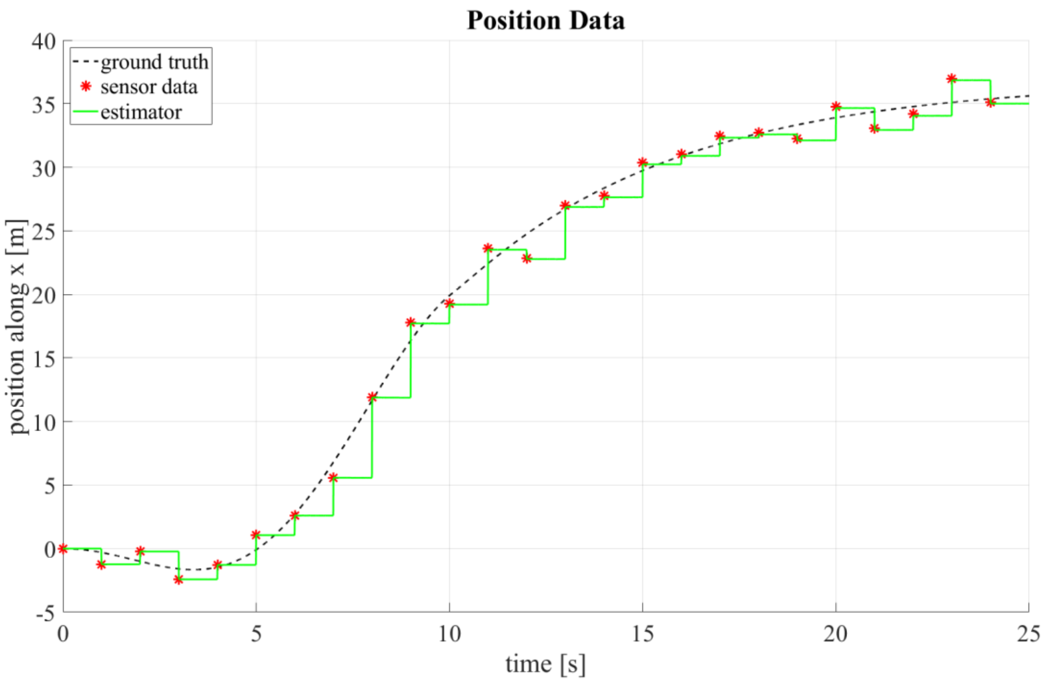

Figure 7: The effect of unmitigated sensor quantization on the output of an estimator updating more quickly than the sensor data. The simulated ground truth is shown as a black dashed line, the simulated 1 Hz GNSS data is shown as red asterisks, and the estimator output is showan as a green solid line. Note the stairstep-like estimator quality.

For state-transition models that make predictions based on a derivative component of past signals (e.g. the constant-acceleration state-transition model described in the previous section), sensor quantization is detrimental to sensor performance.

The most primitive quantization mitigation method involves synchronizing the update rate of the Kalman filter to match the update rate of the measuring sensor. This method of *sensor update synchronization* is demonstrated in Figure 8.

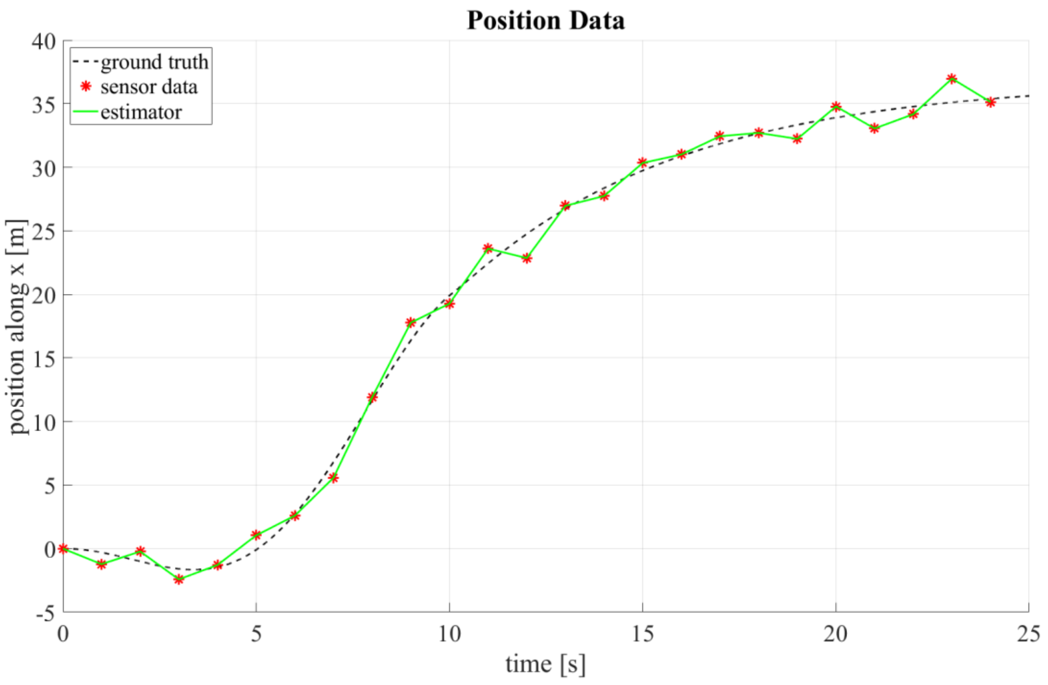

Figure 8: The effect of sensor update synchronization on the output of an estimator. The simulated ground truth is shown as a black dashed line, the simulated 1 Hz GNSS data is shown as red asterisks, and the estimator output is shown as a green solid line. Note the improvement over the stairstep-like estimator quality in Figure 7.

Sensor update synchronization ensures that no redundant measurements are input to the Kalman filter, which eliminates all quantization effects; however, this method is only applicable when: (1) the user does not require state estimates faster than the sensor rate, and (2) only a single sensor rate is employed in the system. For observer systems employing sensors with fast and uniform update rates, sensor update synchronization is effective [103]; however, for mobile robot platforms with employing slow-updating GNSS systems in real-time, this method is extremely limiting.

Another method of quantization mitigation, that I will call *update limiting*, limits the update cycles of the Kalman filter to only occur when the sensor produces an update. In update limiting, if the sensor updates during the time step of a Kalman filter update, then the Kalman filter proceeds as normal, generating a state *prediction* based on the last state estimate, collecting a state *measurement* from the sensor, and weighing those values and their covariances to produce a resulting state estimate. If the Kalman filter updates without a sensor update, then the Kalman filter only generates a state prediction, which is treated as the state estimate for that iteration. This allows the Kalman filter to update more quickly than the sensor rate without utilizing redundant sensor measurements. This general observation process was first documented in the 1992 paper [104], and has received considerable subsequent attention due to its relevancy to multiple fields of research, and difficulty of characterizing the input conditions for which a generalized observer converges [102], [105]–[107]. Assuming the observability of the discrete sensor input is sufficient for convergence, an update limiting Kalman filter can produce estimates more quickly than the sensor update rate; however, the method is still limited to a single sensor rate. If multiple sensors are employed in a system, update limiting can be applied to the fastest sensor in the system, resulting in quantization on the slower sensor(s), or updates can be limited to the slowest sensor, resulting in ignoring data from the faster sensor(s). For systems with significantly different update rates between sensors, as is the case in the GNSS+IMU sensor system employed in our work, neither of these solutions are ideal.

In update limiting, the Kalman filter only calculates an update step if the sensor produced a measurement for that iteration. I can apply this same concept to multiple sensor rates by designing the Kalman filter to always calculate predict and update steps, but when measurements are not present, the diagonal entries of the measurement noise covariance matrix (which represent the variance of the state measurement [78]) are driven to be arbitrarily large, effectively removing the effect of the measurements on the state estimate. This technique, called *covariance profiling*, operates on the same concept as update limiting, but it allows the filter designer to vary the weighted contribution from the measurements on a per-sensor basis, i.e. individual sensors can contribute updates to the system state estimate as they produce updates. This concept has been visited by a few researchers in various contexts. In [108], covariance profiling was used to suppress quantization from optical shaft encoders alongside other sensor sources; shaft encoders in particular are challenging, as they produce non-uniform quantized steps as a function of angular velocity. In [64], a similar algorithm was employed, adaptively varying the covariance of a wheeled robot with wheel encoders to prevent filter divergence. In [60], [109], [110], a fuzzy logic controller was used to adaptively change the covariance matrices in response to unstable measurements. In [7], this sensor quantization problem was treated more generally as “missing data” in learning and interference problems, and noted specifically that the problem is “typically not well addressed in linear dynamical systems [Kalman filtering] literature”. [7] then proposes the idea of “spatially adaptive observation noise”, where the observation noise can have different statistics in different parts of the state/observation space, rather than being described by a single matrix.

There is also a significant body of work in estimation of *delayed*, or out-of-sequence, sensor measurements, i.e. when various sensors are employed with significantly varying latencies; failure to compensate for these delays results in the estimator fusing data from dissimilar time increments [111]–[114]. The general approach to address sensor delay is to take an estimator that has the desired performance for delay free measurements, and modify update step such that it compares the delayed measurement with its corresponding backward time-shifted estimate [113]. The estimation of delayed sensor measurements is not discussed in detail in this work; however, the problem of delayed sensor measurements is highly relevant for practical sensor deployments. The solution to both problems are similar: use the predict-update nature of the Kalman filter to fuse sensor data on a per-sensor basis.

In our work, I implement a sliding adaptive covariance profiling method based on the time elapsed since the most recent update. Our method is “adaptive” with respect to time, and “sliding” because the function defining this adaptive behavior is a logarithmic sliding function. It is a $\log_2$ piecewise function. It is described in equation (25).

$$G = \begin{cases} G_m \cdot 2^{\left(-\Delta t \cdot \left(\frac{1}{t_{1/2}}\right)\right)} + 1, & 0 \leq \Delta t < t_{smooth} \\ G_m \cdot 2^{\left(\left(\Delta t - (t_{scale} + t_{smooth})\right) \cdot \left(\frac{1}{t_{\frac{1}{2}}}\right)\right)} + 1, & t_{smooth} \leq \Delta t < t_{smooth} + t_{scale} \\ G_m, & t_{scale} \leq \Delta t \end{cases} \tag{25}$$

Where

G is the sliding covariance gain
G_m is the desired maximum covariance gain, typically 10^5
Δt is the time elapsed since the last sensor update
$t_{1/2}$ is the half life of the exponential function
$0 \leq \Delta t < t_{smooth}$ is the smoothing region
$t_{smooth} \leq \Delta t < t_{smooth} + t_{scale}$ is the scaling region
$t_{scale} \leq \Delta t$ is the rejected region

Because covariances are relative weights, it is more mathematically appropriate to scale the covariance based on a logarithmic function. In our case, I chose $\log_2$, as it is much more intuitive to select a half life (the time that the exponential function takes to reduce to half of its initial value) to define the rate of exponential decay.

Region one of equation (25) is the *smoothing* region; in this region, the adaptive function scales the covariance gain from its maximum value G_m to one, i.e. it scales from the least confidence to the most confidence. Region two is the *scaling* region; in this region, the adaptive function scales the covariance gain from one back up to G_m, i.e. from the most confidence to the least confidence. Region three is the *rejected* region; in this region, the function remains constant at G_m (its least confidence state), i.e. the covariance gain is so large that the measurement is not taken into account. The filter designer must provide four variables. These are listed in equation (26).

G_m is the maximum covariance gain, or the point at which we consider the measurements to no longer be considered. $G_m = 10^5$ is used by default, which is applicable to most cases.
t_{smooth} is the length of time spent in the smoothing region; it is convenient to express this as a function of the sensor time step, dt, as it will scale the smoothing time based on how quickly the sensor updates. $t_{smooth} = 0.15 \cdot dt$ is used by default.
t_{scale} is the length of time spent in the scaling region; it is convenient to express this as a function of the sensor time step, dt, as it will scale the smoothing time based on how quickly the sensor updates. $t_{smooth} = 0.1 \cdot dt$ is used by default. (26)
$t_{1/2}$ is the half life of the exponential decay; it is convenient to express this as a function of the respective t_{smooth} and t_{scale} time steps, as it will scale the half life based on the total length of the region. $t_{1/2} = 0.1 \cdot t_{smooth}$ and $t_{1/2} = 0.1 \cdot t_{scale}$ are used by default.

Equation (25) using the default variables in equation (26) is plotted for a one second sensor update in Figure 9.

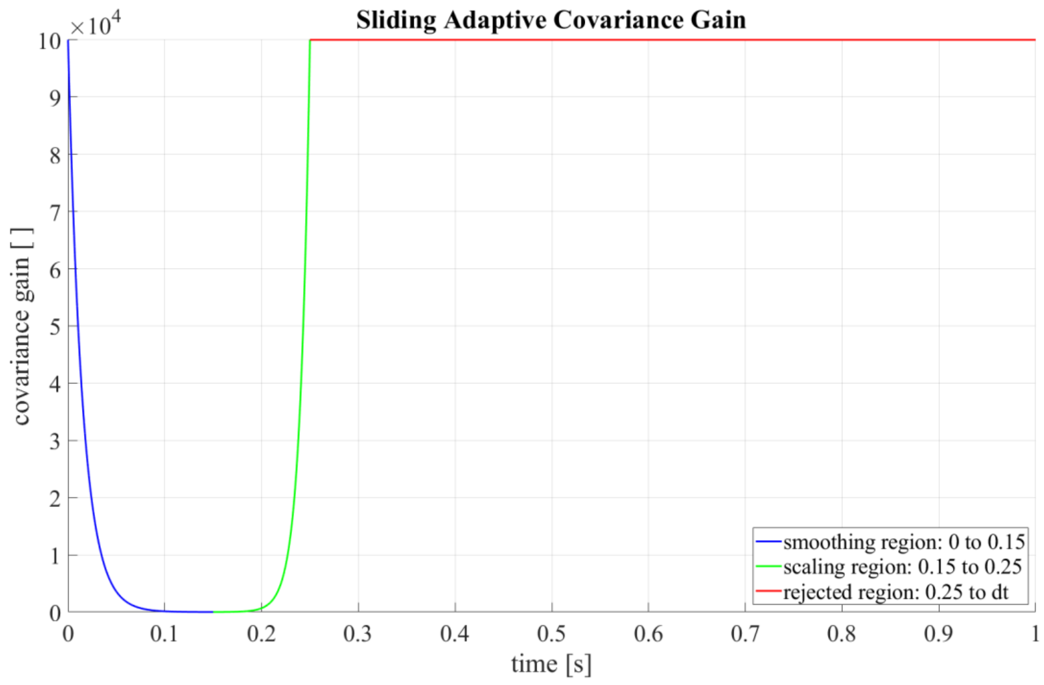

Figure 9: Example sliding adaptive covariance gain for a one second sensor update interval. G_m, t_{smooth} (region one), t_{scale} (region two), and $t_{1/2}$ can be varied to change how far of a range the covariance scales over and how quickly it decays.

The sliding adaptive covariance gain G in equation (25) should be calculated for each sensor at each discrete-time iteration of the Kalman filter. Then, multiplying G by its respective row in the measurement noise matrix $\vec{R}_k$ from equation (5) will return a revised covariance that is appropriately scaled based on the time elapsed since the last sensor update. This is given in equation (27).

$$\vec{R}_{k_{adaptive}} = \vec{R}_k \cdot \vec{G}$$

$$\Rightarrow \vec{R}_{k_{adaptive}} = \begin{bmatrix} R_x G_x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & R_y G_y & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & R_\psi G_\psi & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & R_{\dot{x}} G_{\dot{x}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & R_{\dot{y}} G_{\dot{y}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & R_{\dot{\psi}} G_{\dot{\psi}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & R_{\ddot{x}} G_{\ddot{x}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{\ddot{y}} G_{\ddot{y}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{\ddot{\psi}} G_{\ddot{\psi}} \end{bmatrix} \tag{27}$$

Where
$\vec{R}_{k_{adaptive}}$ is the revised measurement noise covariance matrix incorporating the sliding adaptive covariance gain
$\vec{R}_k$ is the measurement noise covariance matrix
$\vec{G}$ is the sliding adaptive covariance gain vector

Increasing the length of region one by increasing t_{smooth} will suppress "jumps" in the estimator output in real time, but comes at the cost of introducing a lag into the estimator output. Additionally, increasing the length of region two by increasing t_{scale} will increase the estimator's confidence in that measurement for a longer period. Depending on the predictive ability of the state-transition model, this may be desirable or undesirable.

Finally, note that one can "disable" sliding adaptive covariance profiling by setting t_{smooth} and t_{scale} to zero. In this case, the adaptive covariance gain $G = 1$ when a sensor produces an update, and $G = G_m = 10^5$ (arbitrarily large, and therefore not considered) for all other discrete time. This is still an adaptive covariance behavior, as the gain changes with time, but is better described as "binary" adaptive covariance profiling, since the measurement is either considered entirely, or rejected. In our work, I employ sliding adaptive covariance profiling for the GNSS sensor states using the default values in equation (26), and I employ binary adaptive covariance profiling for the IMU sensor states. This is because the GNSS updates slowly and is weighted heavily, which produces strong quantization artifacts in the estimator; conversely, the IMU produces updates very quickly, so quantization smoothing is not necessary.

3.3. Unobserved linear velocity estimation based on the moving-mean forward-difference of GNSS position data, and trapezoidal method of accelerometer (IMU) data.

In a velocity-denied robot employing the constant-acceleration state-transition model given in equation, the linear velocity states do not have measurements to correct their prediction. An accurate linear velocity state is important for predicting the linear position states and it is useful for higher level robot control. Consider the constant-acceleration state-transition model given in equation (11); it is listed at equation (28) below for convenience.

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \psi_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \\ \dot{\psi}_{k+1} \\ \ddot{x}_{k+1} \\ \ddot{y}_{k+1} \\ \ddot{\psi}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \dot{x}_k \cdot dt_k \\ y_k + \dot{y}_k \cdot dt_k \\ \psi_k + \dot{\psi}_k \cdot dt_k \\ \dot{x}_k + \ddot{x}_k \cdot dt_k \\ \dot{y}_k + \ddot{y}_k \cdot dt_k \\ \dot{\psi}_k + \ddot{\psi}_k \cdot dt_k \\ \ddot{x}_k \\ \ddot{y}_k \\ \ddot{\psi}_k \end{bmatrix}$$

Where (28)

x_k is the linear position of the robot
y_k is the linear position of the robot
ψ_k is the angular position of the robot
$\dot{x}_k$ is the linear velocity of the robot
$\dot{y}_k$ is the linear velocity of the robot
$\dot{\psi}_k$ is the angular velocity of the robot
$\ddot{x}_k$ is the linear acceleration of the robot
$\ddot{y}_k$ is the linear acceleration of the robot
$\ddot{\psi}_k$ is the angular acceleration of the robot
dt is the discrete time step

Equation (28) demonstrates that the pose and twist state predictions, $(x_{k+1}, y_{k+1}, \psi_{k+1}, \dot{x}_{k+1}, \dot{y}_{k+1}, \dot{\psi}_{k+1})$ are calculated by summing the last state with its derivative times the time step, dt_k. Due to the variance sum law [115], the variance of a probability distribution always increases when summed with successive independent variances. Therefore, without measurements, to correct prediction errors, equation (28) will increase without bound. From this, one may conclude that the lack of a linear velocity sensor may be a debilitating limitation; however, the GNSS+IMU sensor still gives us conditionally probabilistic information that improves our confidence in the linear velocity state. First, linear velocity is related to linear position based on its derivative, which is measured by the GNSS sensor; and second, linear velocity is related to linear acceleration based on its integral, which is measured by the IMU sensor. To maximize our use of available information in our algorithm, I use both pieces of information to produce a pseudo-measurement of linear velocity. Because this measurement comes from discrete-time data, I will calculate the derivative of the GNSS linear position data using the forward-difference method, and I will calculate the integral of the IMU linear acceleration data using the trapezoidal-method.

3.3.1. Forward-difference of incremental moving mean of linear position measurements for linear velocity approximation

In this sub-section I describe the process for approximating linear velocity by taking the forward-difference (discrete derivative) of the linear position measurements from the GNSS sensor. One drawback of the forward-difference method is the its high sensitivity to noisy data. The proof for this sensitivity is shown in appendix 6.3. Because of this, it is important to first smooth the linear position data, to reduce abrupt jumps due to noise. I do this using an incremental moving mean. I then take the forward-difference of the next incremental moving mean from the previous incremental moving mean to get the approximate derivative of the linear position data. The algebraic manipulations to accomplish these sequential tasks can be symbolically evaluated for arrive at a single, convenient linear position measurement equation.

First consider a sequence of sensor measurements, s, given in equation (29).

$$s = \{s_1, s_2, s_3, \dots, s_n\} \tag{29}$$

The classic definition of the mean of sequence s is given in equation (30).

$$\begin{aligned} \bar{s}_n &= \left(\frac{1}{n}\right)(s_1 + s_2 + s_3 + \cdots + s_n) \\ &= \frac{1}{n}\sum_{j=1}^{n} s_j \end{aligned} \tag{30}$$

Equation (30) is valid for taking mean of the *entire sequence* of numbers; however, I am only interested in taking the mean of the last n values of a sequence. I assign an indexing variable i to the sequence, and define the mean of the last n values in a sequence using equation (31).

$$\begin{aligned}\bar{s}_{(i-n+1)\to i} &= \left(\frac{1}{n}\right)(s_{i-n+1} + s_{i-n+2} + s_{i-n+3} + \cdots + s_i) \\ &= \frac{1}{n}\sum_{j=i-n+1}^{n} s_j\end{aligned} \tag{31}$$

Where
$\bar{s}_{(i-n+1)\to i}$ is the mean of the last n values in a the sequence s, from index i
i is the index of the most current (last) value in the sequence
n is the number of values in the sequence over which to take the mean

Equation (31) could be implemented in a filtering algorithm as-is; however, it is computationally cheaper to calculate the moving mean in an incremental fashion, as it requires fewer variables to keep in memory for each iteration. In an incremental mean, I use the mean from the previous $(i-1)$ iteration, or $\bar{s}_{((i-1)-n+1)\to(i-1)}$, to calculate the mean from the current iteration, $\bar{s}_{(i-n+1)\to i}$. This requires that I re-write equation (31) shown in equation (32).

$$\begin{aligned}\bar{s}_{(i-n+1)\to i} &= \frac{1}{n}\sum_{j=i-n+1}^{n} s_j \\ &= \frac{1}{n}\left(\left(\sum_{j=((i-1)-n+1)}^{n} s_j\right) - s_{i-n} + s_i\right) \\ &= \frac{1}{n}\left(\left(n\left(\bar{s}_{((i-1)-n+1)\to(i-1)}\right)\right) - s_{i-n} + s_i\right) \\ &= \bar{s}_{((i-1)-n+1)\to(i-1)} + \frac{1}{n}(s_i - s_{i-n})\end{aligned} \tag{32}$$

Where
$\bar{s}_{(i-n+1)\to i}$ is the mean of the last n values in a the sequence s, from index i
$\bar{s}_{((i-1)-n+1)\to(i-1)}$ is the mean from the previous iteration
i is the index of the most current (last) value in the sequence
n is the number of values in the sequence over which to take the mean

I have now established a means for taking an incremental moving mean of data. The goal in this section is to take the discrete derivative of our, now "smoothed" linear position measurements. There are multiple methods of discrete differentiation; for a stream of discrete data, the forward-difference method works well. It is based on the definition of the derivative, which is given in equation (33).

$$f'(x) = \lim_{h\to 0}\frac{f(x+h)-f(x)}{h} \tag{33}$$

In the discrete-time domain, time is described by integer iterations k, with known time steps, dt. Equation (33) can be approximated by the discrete-time forward-difference formula, give in equation (34).

$$f'(x) \approx \frac{x_{k+1}-x_k}{dt} \tag{34}$$

Combining equation (34) to approximate the derivative of a n-means of the linear position state measurements; i.e. the forward-difference of $\bar{s}_{((i-1)-n+1)\to(i-1)}$ to $\bar{s}_{(i-n+1)\to i}$ with equation (32) over the time step dt_s gives equation (35).

$$\begin{aligned}
f'\left(\bar{s}_{(i-n+1)\to i} \to \bar{s}_{((i-1)-n+1)\to(i-1)}\right) &\approx \frac{\left(\bar{s}_{(i-n+1)\to i}\right)-\left(\bar{s}_{((i-1)-n+1)\to(i-1)}\right)}{(dt_s)} \\
&\approx \frac{1}{dt_s}\left(\left(\bar{s}_{((i-1)-n+1)\to(i-1)}+\frac{1}{n}(s_i-s_{i-n})\right)\right. \\
&\qquad \left. -\left(\bar{s}_{((i-1)-n+1)\to(i-1)}\right)\right) \\
&\approx \frac{1}{dt_s}\left(\frac{1}{n}(s_i-s_{i-n})\right) \\
&\approx \frac{1}{dt_s\cdot n}(s_i-s_{i-n})
\end{aligned} \tag{35}$$

Note that when taking the forward difference of the means of two sequences offset by a single index, all of the "inner" terms of the sequence cancel, and I are equivalently left with only the oldest term of the oldest sequence s_{i-n}, and the newest term of the newest sequence s_i. This is convenient, as the estimation algorithm only needs to keep two pieces of information in memory, even for extremely large sequences of numbers.

Writing equation (35) in terms of the linear position sensor measurements gives us equation (36).

$$\begin{cases} \dot{x}_G \approx \dfrac{1}{dt_G \cdot n}\left(x_{G,i} - x_{G,i-n}\right) \\ \dot{y}_G \approx \dfrac{1}{dt_G \cdot n}\left(y_{G,i} - y_{G,i-n}\right) \end{cases}$$

Where
- $\dot{x}_G$ is the linear velocity pseudo-measurement along the x-axis based on the forward difference of GNSS linear position data along the x-axis
- $\dot{y}_G$ is the linear velocity pseudo-measurement along the y-axis based on the forward difference of GNSS linear position data along the y-axis
- dt_G is the time step between GNSS linear position measurements (assumed to be constant for all measurements in the sequence)
- n is the number of linear position measurements over which to take the mean
- $x_{G,i}$ is the most current GNSS linear position measurement along the x-axis
- $y_{G,i}$ is the most current GNSS linear position measurement along the y-axis
- $x_{G,i-n}$ is the GNSS linear position measurement along the x-axis from $i-n$ iterations ago
- $y_{G,i-n}$ is the GNSS linear position measurement along the y-axis from $i-n$ iterations ago

(36)

The filter designer must provide the number of state sensor updates used to calculate the incremental mean, n. For a stationary, normally distributed signal, taking the mean of n successive measurements improves the signal-to-noise ratio at a rate of $\sqrt{n}$, e.g. the mean of four measurements, improves the signal to noise ratio by a factor of two [116]. Mobile robot linear position measurements are clearly not stationary, meaning that this rule-of-thumb only represents an ideal case, as the transient part of the signal is expressed in the mean at a rate of $\frac{1}{n \cdot dt_s}$. Selecting n is a dependent on three factors: (1) the acceptable signal-to-noise ratio in the smoothed signal, (2) the acceptable accuracy in the forward-difference approximation, and (3) the refresh rate of the sensor. The range of acceptable values for n will be explored in this work.

3.3.2. Trapezoidal method of linear acceleration measurements for linear velocity prediction

In this sub-section I describe the process for approximating linear velocity by taking the trapezoidal method (discrete integral) of the linear acceleration measurements from the IMU sensor. In the previous section, I took the incremental moving mean of the linear position measurements before performing the forward-difference, because the method is highly sensitive to noisy data. This extra step is not necessary for the trapezoidal method, because the trapezoidal method is orders of magnitude less sensitive to noise. The proof for this is shown in appendix 6.4.

Like discrete differentiation, there are multiple methods of discrete integration. The most appropriate method for a stream of discrete data is the trapezoidal method. The definition of the definite integral is given in equation (37).

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\,\Delta x \tag{37}$$

The trapezoidal method works by partitioning function $f(x)$ into k intervals over the integration range, approximating the function over each interval as a trapezoid, and summing the results. This approximation gets more accurate as the size of each interval decreases. The trapezoidal method is given in equation (38).

$$\int_a^b f(x)\,dx \approx \sum_{k=1}^{N} \frac{f(x_k) + f(x_{k+1})}{2} \Delta x_{k+1} \tag{38}$$

As previously discussed, the trapezoidal method is not sensitive to noise in the same was that the forward-difference is. Therefore, I can directly substitute the linear acceleration data from the IMU into equation (38) to approximate the linear velocity. This is performed in equation (39).

$$\begin{cases} \dot{x}_I \approx \dot{x}_{k-1} + \dfrac{\ddot{x}_{I,i-1} + \ddot{x}_{I,i}}{2} dt_I \\ \dot{y}_I \approx \dot{y}_{k-1} + \dfrac{\ddot{y}_{I,i-1} + \ddot{y}_{I,i}}{2} dt_I \end{cases} \tag{39}$$

Where
- $\dot{x}_I$ is the linear velocity pseudo-measurement along the x-axis based on the trapezoidal method of IMU linear acceleration along the x-axis.
- $\dot{y}_I$ is the linear velocity pseudo-measurement along the y-axis based on the trapezoidal method of IMU linear acceleration along the y-axis.
- $\dot{x}_{k-1}$ is the linear velocity estimate along the x-axis from the previous estimator iteration
- $\dot{y}_{k-1}$ is the linear velocity estimate along the y-axis from the previous estimator iteration
- $\ddot{x}_{I,i-1}$ is the IMU linear acceleration measurement along the x-axis from the last sensor update
- $\ddot{y}_{I,i-1}$ is the IMU linear acceleration measurement along the y-axis from the last sensor update
- $\ddot{x}_{I,i}$ is the most current IMU linear acceleration measurement along the x-axis
- $\ddot{y}_{I,i}$ is the most current IMU linear acceleration measurement along the y-axis
- dt_I is the time step between IMU linear acceleration measurements

Note that the trapezoidal method requires a previous velocity measurement to sum with the area under the curve of the most recent increment; I take this information from the Kalman filter linear velocity output from the previous iteration ($\dot{x}_{k-1}, \dot{y}_{k-1}$). One may be inclined to believe that this velocity estimate should come from the pseudo-measurement of linear velocity from the previous IMU accelerometer sensor update, since this would keep the pseudo-measurement entirely self-contained to the IMU accelerometer. Unfortunately, this does not work in practice, as any amount of systematic bias in the linear acceleration measurement will cause the integral to increase without bound, and low-cost MEMS accelerometers are notoriously prone to systematic drift. Referencing the integration to the

Kalman filter linear velocity estimate from the previous iteration will introduce coupling into the pseudo-measurement; however, this is a necessary tradeoff, as the linear velocity estimate error can be bounded by forward-difference pseudo-measurement from the GNSS sensor, provided that the covariance weighting is properly handled.

3.3.3. <u>Combining the linear-velocity pseudo-measurements and covariances</u>

In the previous sub-sections, I developed pseudo-measurements for linear velocity based on the forward-difference of GNSS linear position measurements $\left(\dot{x}_{G,i}, \dot{y}_{G,i}\right)$, and the trapezoidal method of IMU linear acceleration measurements $\left(\dot{x}_{I,i}, \dot{y}_{I,i}\right)$. I aim to combine these two pseudo-measurements into a single pseudo-measurement for linear velocity. The obvious method of handling this would be to take the average of the two measurements, which weights the two measurements equally. This method works, but it does not consider known information about the covariances of the two sensor systems. A more ideal combinational method should weight the measurements based on their covariance; i.e. a measurement with less covariance (more certainty) should be weighted more heavily.

First, I a must extract the variance of each of the linear velocity pseudo-measurements, which are characterized apriori in the Kalman filter measurement noise covariance matrix $\vec{R}_k$ matrix in equation (5). Incorporating the sliding adaptive covariance gain G discussed in section 3.2 results in the revised measurement noise covariance matrix $\vec{R}_{k_{adaptive}}$ given in equation (27). Because the linear velocity is calculated based on the forward difference of the GNSS linear position measurements and the trapezoidal method of the IMU linear acceleration measurements, the linear velocity covariance terms cannot be assumed to be independent, i.e. the linear velocity covariance terms of the of the $\vec{R}_{k_{adaptive}}$ matrix given in equation (27) are not correct. To calculate the correct linear velocity variance, I must calculate the symbolic variance of the pseudo linear velocity measurements. I performed this calculation in appendix 6.3 for the forward-difference method, given in equation (67), and in appendix 6.4 for the trapezoidal method, given in equation (70).

Substituting in the individual covariance matrix terms from equation (27) into equations (67) and (71) yields four variance equations for the two linear velocities $(\dot{x}, \dot{y})$. This is given in equation (40).

$$\begin{cases} var(\dot{x}_G) = \dfrac{\sqrt{2}}{{dt_G}^2} \cdot var(x_G) \\ var(\dot{y}_G) = \dfrac{\sqrt{2}}{{dt_G}^2} \cdot var(y_G) \\ var(\dot{x}_I) \approx var(\dot{x}) \\ var(\dot{y}_I) \approx var(\dot{y}) \end{cases}$$

$$\Rightarrow \begin{cases} var(\dot{x}_G) = \dfrac{\sqrt{2}}{{dt_G}^2} \cdot (R_x) \cdot G_x \\ var(\dot{y}_G) = \dfrac{\sqrt{2}}{{dt_G}^2} \cdot \left(R_y\right) \cdot G_y \\ var(\dot{x}_I) \approx var(\dot{x}) \\ var(\dot{y}_I) \approx var(\dot{y}) \end{cases} \tag{40}$$

Where

$var(\dot{x}_G)$ is the variance of the linear velocity pseudo-measurement along the x-axis due to the forward-difference of the GNSS linear position data

$var(\dot{y}_G)$ is the variance of the linear velocity pseudo-measurement along the y-axis due to the forward-difference of the GNSS linear position data

$var(\dot{x}_I)$ is the variance of the linear velocity pseudo-measurement along the x-axis due to the trapezoidal method of the IMU linear acceleration data

$var(\dot{y}_I)$ is the variance of the linear velocity pseudo-measurement along the y-axis due to the trapezoidal method of the IMU linear acceleration data

dt_G discrete-time step of the GNSS linear position measurements

dt_I discrete-time step of the IMU linear acceleration measurements

$var(x_G)$ is the variance of the GNSS linear position measurement along the x-axis

$var(y_G)$ is the variance of the GNSS linear position measurement along the y-axis

$var(\dot{x})$ is the variance of the linear velocity measurement

$var(\dot{y})$ is the variance of the linear velocity measurement

R_x covariance of GNSS linear position measurements along x-axis

G_x sliding adaptive covariance gain of the GNSS linear position measurements along x-axis

R_y covariance of GNSS linear position measurements along y-axis

G_y sliding adaptive covariance gain of the GNSS linear position measurements along y-axis

I now have the symbolic form of the variance for all four linear velocity pseudo-measurements $(\dot{x}_G, \dot{y}_G, \dot{x}_I, \dot{y}_I)$. Instead of taking the average of the measurements $\dot{x}_G$ and $\dot{x}_I$, I can take the *weighted average*, based on their respective variances; $\dot{y}_G$ and $\dot{y}_I$. Note that evaluating this weighted average is the mathematical equivalent of multiplying the two Gaussian probability density functions, resulting in a scaled Gaussian function (but non-Gaussian probability density function) [78], [117]. The well-known equations for the mean and variance of the product of two Gaussian probability density functions are given in context in equations (41) and (42), respectively.

$$\begin{cases} \dot{x}_P = \dfrac{\dot{x}_G \cdot var(\dot{x}_G) + \dot{x}_I \cdot var(\dot{x}_I)}{var(\dot{x}_G) + var(\dot{x}_I)} \\ \dot{y}_P = \dfrac{\dot{y}_G \cdot var(\dot{y}_G) + \dot{y}_I \cdot var(\dot{y}_I)}{var(\dot{y}_G) + var(\dot{y}_I)} \end{cases} \tag{41}$$

Where

$\dot{x}_P$ is the linear velocity pseudo-measurement along the x-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$\dot{y}_P$ is the linear velocity pseudo-measurement along the y-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$\dot{x}_G$ is the linear velocity pseudo-measurement along the x-axis based on the forward difference of GNSS linear position data along the x-axis

$\dot{y}_G$ is the linear velocity pseudo-measurement along the y-axis based on the forward difference of GNSS linear position data along the y-axis

$var(\dot{x}_G)$ is the variance of the linear velocity pseudo-measurement along the x-axis due to the forward-difference of the GNSS linear position data

$var(\dot{y}_G)$ is the variance of the linear velocity pseudo-measurement along the y-axis due to the forward-difference of the GNSS linear position data

$\dot{x}_I$ is the linear velocity pseudo-measurement along the x-axis based on the trapezoidal method of IMU linear acceleration along the x-axis

$\dot{y}_I$ is the linear velocity pseudo-measurement along the y-axis based on the trapezoidal method of IMU linear acceleration along the y-axis

$var(\dot{x}_I)$ is the variance of the linear velocity pseudo-measurement along the x-axis due to the forward-difference of the IMU linear acceleration data

$var(\dot{y}_I)$ is the variance of the linear velocity pseudo-measurement along the y-axis due to the forward-difference of the IMU linear acceleration data

$$\begin{cases} var(\dot{x}_P) = \dfrac{var(\dot{x}_G) \cdot var(\dot{x}_I)}{var(\dot{x}_G) + var(\dot{x}_I)} \\ var(\dot{y}_P) = \dfrac{var(\dot{y}_G) \cdot var(\dot{y}_I)}{var(\dot{y}_G) + var(\dot{y}_I)} \end{cases} \tag{42}$$

Where

$var(\dot{x}_P)$ is the variance of the linear velocity pseudo-measurement along the x-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$var(\dot{y}_P)$ is the variance of the linear velocity pseudo-measurement along the y-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$\dot{x}_G$ is the linear velocity pseudo-measurement along the x-axis based on the forward difference of GNSS linear position data along the x-axis

$\dot{y}_G$ is the linear velocity pseudo-measurement along the y-axis based on the forward difference of GNSS linear position data along the y-axis

$var(\dot{x}_G)$ is the variance of the linear velocity pseudo-measurement along the x-axis due to the forward-difference of the GNSS linear position data

$var(\dot{y}_G)$ is the variance of the linear velocity pseudo-measurement along the y-axis due to the forward-difference of the GNSS linear position data

$var(\dot{x}_I)$ is the variance of the linear velocity pseudo-measurement along the x-axis due to the forward-difference of the IMU linear acceleration data

$var(\dot{y}_I)$ is the variance of the linear velocity pseudo-measurement along the y-axis due to the forward-difference of the IMU linear acceleration data

Note however, that our symbolic formulation of the variance of the trapezoidal method in equation (71) tells us that the variance of the trapezoidal method can be approximated as the original variance of the linear velocity measurement, since the summed integration term adds an infinitesimally small variance onto the previous velocity. Because the only other term influencing the linear velocity variance is the variance of the GNSS sensor $var(\dot{x}_G)$ and $var(\dot{y}_G)$, I assume that $var(\dot{x}_I) \approx var(\dot{x}_G)$, and $var(\dot{y}_I) \approx var(\dot{y}_G)$. With this information equation (41) reduces down to a simple mean given in equation (43); likewise (42) simplifies into the form given in equation (44) below.

$$\begin{cases} \dot{x}_P = \dfrac{(\dot{x}_G + \dot{x}_I)}{2} \\ \dot{y}_P = \dfrac{(\dot{y}_G + \dot{y}_I)}{2} \end{cases} \tag{43}$$

Where

$\dot{x}_P$ is the linear velocity pseudo-measurement along the x-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$\dot{y}_P$ is the linear velocity pseudo-measurement along the y-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$\dot{x}_G$ is the linear velocity pseudo-measurement along the x-axis based on the forward difference of GNSS linear position data along the x-axis

$\dot{y}_G$ is the linear velocity pseudo-measurement along the y-axis based on the forward difference of GNSS linear position data along the y-axis

$var(\dot{x}_G)$ is the variance of the linear velocity pseudo-measurement along the x-axis due to the forward-difference of the GNSS linear position data

$var(\dot{y}_G)$ is the variance of the linear velocity pseudo-measurement along the y-axis due to the forward-difference of the GNSS linear position data

$$\begin{cases} var(\dot{x}_P) = \dfrac{var(\dot{x}_G)^2}{2 \cdot var(\dot{x}_G)} \\ var(\dot{y}_P) = \dfrac{var(\dot{y}_G)^2}{2 \cdot var(\dot{y}_G)} \end{cases} \tag{44}$$

Where

$var(\dot{x}_P)$ is the variance of the linear velocity pseudo-measurement along the x-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$var(\dot{y}_P)$ is the variance of the linear velocity pseudo-measurement along the y-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$\dot{x}_G$ is the linear velocity pseudo-measurement along the x-axis based on the forward difference of GNSS linear position data along the x-axis

$\dot{y}_G$ is the linear velocity pseudo-measurement along the y-axis based on the forward difference of GNSS linear position data along the y-axis

$var(\dot{x}_G)$ is the variance of the linear velocity pseudo-measurement along the x-axis due to the forward-difference of the GNSS linear position data

$var(\dot{y}_G)$ is the variance of the linear velocity pseudo-measurement along the y-axis due to the forward-difference of the GNSS linear position data

To summarize, the pseudo-measurements $\dot{x}_P$ and $\dot{y}_P$ from equation (43) replace the fourth and fifth elements, respectively, in the measurement matrix $\vec{z}_k$ in equation (6); and the variance of the pseudo-measurements $var(\dot{x})$ and $var(\dot{y})$ from equation (44) replace the (4,4) and (5,5) positions in the sliding adaptive measurement noise covariance matrix $\vec{R}_{k_{adaptive}}$ matrix in equation (27). The revised state measurement vector $\vec{z}_k$ is given in equation (45), and the revised sliding adaptive measurement noise covariance matrix $\vec{R}_{k_{adaptive}}$ is given in equation (46).

$$\vec{z}_k = \begin{cases} x_G \\ y_G \\ \psi_I \\ \dot{x}_P \\ \dot{y}_P \\ \dot{\psi}_I \\ \ddot{x}_I \\ \ddot{y}_I \\ 0 \end{cases}$$

Where
- $\vec{z}_k$ is the state measurement vector for the robot body frame state estimator at time k
- x_G is the is the most current GNSS linear position measurement along the x-axis (45)
- y_G is the is the most current GNSS linear position measurement along the y-axis
- ψ_I is the most current IMU angular position measurement about the z-axis
- $\dot{x}_P$ is the linear velocity pseudo-measurement along the x-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration
- $\dot{y}_P$ is the linear velocity pseudo-measurement along the y-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements
- $\dot{\psi}_I$ is the most current IMU angular velocity measurement about the z-axis
- $\ddot{x}_I$ is the most current IMU linear acceleration measurement along the x-axis
- $\ddot{y}_I$ is the most current IMU linear acceleration measurement along the y-axis

$$\vec{R}_{k_{adaptive}} = \vec{R}_k \cdot \vec{G}$$

$$\Rightarrow \vec{R}_{k_{adaptive}} = \begin{bmatrix} R_x G_x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & R_y G_y & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & R_\psi G_\psi & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & var(\dot{x}_P) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & var(\dot{y}_P) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & R_{\dot{\psi}} G_{\dot{\psi}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & R_{\ddot{x}} G_{\ddot{x}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{\ddot{y}} G_{\ddot{y}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_{\ddot{\psi}} G_{\ddot{\psi}} \end{bmatrix} \quad (46)$$

Where

$\vec{R}_{k_{adaptive}}$ is the revised measurement noise covariance matrix incorporating the sliding adaptive covariance gain

$\vec{R}_k$ is the measurement noise covariance matrix

$\vec{G}$ is the sliding adaptive covariance gain vector

$var(\dot{x}_P)$ is the variance of the linear velocity pseudo-measurement along the x-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$var(\dot{y}_P)$ is the variance of the linear velocity pseudo-measurement along the y-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

In summary, equation (40) symbolically calculates the variance of the linear velocity pseudo-measurements based on the forward-difference of linear position GNSS data, and the trapezoidal method of linear acceleration IMU data, which are derived in appendix 6.3 and 6.4, respectively. Equations (41) and (42) outline the general framework for combining the two Gaussian distributions based on their mean and variance. Although it is not needed in the particular GNSS+IMU sensor system tested in this work, equations (41) and (42) can be more generally used to incorporate additional (redundant) state measurements for any of the state variables in a state estimator based on a Kalman filter. Equations (43) and (44) make the symbolic substitutions for the final Gaussian mean and variance for the linear velocity pseudo-measurements. Equation (45) reflects the updated state measurement vector, and equation (46) reflects the updated measurement noise covariance matrix.

3.4. The cascading, dual-frame estimator in the robot body frame and then in the map frame

In this section, I describe the configuration of our Kalman filter-based state observer, which can be described as a "cascading, dual-frame" estimator. This state observer configuration is necessary because the GNSS sensor records measurements with respect to the map frame[*] and the IMU sensor records measurements with respect to the body frame. The GNSS and IMU state variable measurements and the reference frames they measure with respect to are listed in Table 7.

Table 7: GNSS and IMU State Measurements and Reference Frames

GNSS		IMU	
Measured State	Reference frame	Measured State	Reference frame
Linear position, x	Map frame	Angular position, ψ	Robot body frame
Linear position, y	Map frame	Angular velocity, $\dot{\psi}$	Robot body frame
		Linear acceleration, $\ddot{x}$	Robot body frame
		Linear acceleration, $\ddot{y}$	Robot body frame

In order to fuse the information from both sensor systems their measurements must measure with respect to the same reference frame. There are two intuitive solutions to this problem:

- **Convert the IMU measurements to the map frame.** This put all the measurements in the map frame, and the Kalman filter calculates state estimates with respect to the map frame.

[*] It is more accurate to say that the GNSS sensor records measurements in the Earth frame, as the data is typically described in latitude-longitude; however, in most applications where the robot area-of-operation can be approximated as a cartesian coordinate system, the GNSS measurements are immediately converted to the map frame. Thus, in the eyes of the state observer, the GNSS measurements are measured with respect to the map frame.

- **Convert the GNSS measurements to the robot body frame.** This will put all the measurements in the robot body frame, and the Kalman filter calculates state estimates with respect to the robot body frame.

Fusing the sensor data with respect to the map frame is advantageous, since I ultimately want the robot pose to be expressed with respect to the map. This makes the first option sound more appealing, as it requires only a single transformation of the IMU data to the map frame; however, transforming the IMU data from the robot body frame to the map frame incurrs significant error in the acceleration measurements. Equation (42) demonstrates that a rotational pose transformation between frames requires knowledge of the angular position ψ, and equation (59) demonstrates that a rotational *velocity* transformation between frames requires knowledge of the angular position and velocity $(\psi, \dot{\psi})$. By extension, a rotational *acceleration* transformation (a very intensive symbolic derivation) requires angular position, velocity, and acceleration $(\psi, \dot{\psi}, \ddot{\psi})$. Because the angular acceleration state $\ddot{\psi}$ is neither measured nor estimated, this transformation is very error prone.

In the second option, I convert the GNSS measurements to the robot body frame. This option is more realistically feasible, because the GNSS sensor only measures linear positions. Therefore, the pose transformation described in equation (42) only requires knowledge of the angular position ψ, which is measured. That said, if the Kalman filter produces estimates with respect to the robot body frame, it must be transformed back into the map frame, as I ultimately want the robot pose to be expressed with respect to the map frame. This means that the pose information from the GNSS sensor undergoes two frame transformations, and the twist and acceleration data from the IMU sensor undergoes one frame transformation. Because each frame transformation introduces additional error into the state estimates (stemming from the error in the angular position ψ estimate), two transformations of the GNSS pose information introduces too much error into the system to be useful.

The cascading, dual-frame state estimator addresses these error propagation problems by employing two separate Kalman filters that feed into each other in a series (or "cascading") fashion. This configuration allows us to take advantage of the best parts of options one and two, while circumventing the problematic parts. This results in a significantly more accurate state estimate that avoids all but one minor frame transformation.

The first state estimator produces estimates with respect to the robot body frame; I call this first state estimator the *robot body frame state estimator*. It takes the robot body frame state vector estimate from the previous iteration and calculates a state vector prediction using the constant-acceleration state-transition matrix given in equation (12). First, the linear position measurements from the GNSS sensor (x_G, y_G) must be converted from the map frame to the robot frame; this is the only measurement that will be frame transformed in our state estimator. This map-frame-to-robot-body-frame transformation can be performed using the process described in section 3.1.5 (equations (15) through (19)). Once transformed, the state measurement vector can be completed using the angular position, angular velocity, and linear acceleration measurements from the IMU, and the linear velocity pseudo-measurements described by equation (47).

$$\vec{z}_{r_k} = \begin{cases} x_{G_r} \\ y_{G_r} \\ \psi_{I_r} \\ \dot{x}_{P_r} \\ \dot{y}_{P_r} \\ \dot{\psi}_{I_r} \\ \ddot{x}_{I_r} \\ \ddot{y}_{I_r} \\ 0 \end{cases}$$

Where

$\vec{z}_{r_k}$ is the state measurement vector for the robot body frame state estimator at time k

x_{G_r} is the is the most current GNSS linear position measurement along the x-axis converted from the map frame to the robot frame (47)

y_{G_r} is the is the most current GNSS linear position measurement along the y-axis converted from the map frame to the robot frame

ψ_{I_r} is the most current IMU angular position measurement about the z-axis

$\dot{x}_{P_r}$ is the linear velocity pseudo-measurement along the x-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration

$\dot{y}_{P_r}$ is the linear velocity pseudo-measurement along the y-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$\dot{\psi}_{I_r}$ is the most current IMU angular velocity measurement about the z-axis

$\ddot{x}_{I_r}$ is the most current IMU linear acceleration measurement along the x-axis

$\ddot{y}_{I_r}$ is the most current IMU linear acceleration measurement along the y-axis

The revised sliding adaptive measurement noise covariance matrix $\vec{R}_{k_{adaptive}}$ remains as it was given in equation (46).

The second state estimator three pose state variables referenced with respect to the map frame (x_m, y_m, ψ_m) and three twist state variables referenced with respect to the robot body frame $(x_r, y_r, \dot{\psi}_r)$; this makes this second state estimator referenced with respect to both the map and robot body frame, so I will call this second state estimator the *hybrid frame state estimator*. The linear position measurements come directly from the GNSS sensor (x_m, y_m), the angular position and angular velocity measurements come from the IMU (remember that $\psi_r = \psi_m$ and $\dot{\psi}_r = \dot{\psi}_m$ in two-dimensions). The key difference in the map frame state estimator is the linear velocity measurements come from the *state estimate output* from the robot body frame state estimator. These linear velocity measurements are still referenced to the robot body frame, which may lead one to conclude that a velocity frame transformation is required, e.g. equation (60); however, if transforming a velocity only for the purpose of pose prediction using the constant-acceleration state-transition model, only an angular *position* (not velocity) transform is necessary. The constant-acceleration state-transition model in equation (11) is revised for the six-state map frame state estimator, and given in equation (48). Because it only employs six states (no acceleration state information), this revised model could be called a constant-velocity state-transition model.

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \\ \dot{z}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \dot{x}_k \cdot dt_k \\ y_k + \dot{y}_k \cdot dt_k \\ \psi_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{\psi}_k \end{bmatrix} \tag{48}$$

Where

x_k is the linear position of the robot
y_k is the linear position of the robot
ψ_k is the angular position of the robot
$\dot{x}_k$ is the linear velocity of the robot
$\dot{y}_k$ is the linear velocity of the robot
$\dot{\psi}_k$ is the angular velocity of the robot

Note how equation (48) makes an assumption of constant angular position, linear velocity, and angular velocity. Normally this is *not* an acceptable assumption for predictive estimation; however, remember that the angular position, linear velocity, and angular velocity states were already estimated in the robot body frame estimator, and therefore do not need state-transition equations. The purpose of hybrid frame state estimator is to estimate the robot linear position states in the map frame (x_m, y_m); the linear velocity states are only needed for its role in the second term of the linear position prediction equations in equation (48). In this capacity, the linear velocity is integrated in discrete-time by multiplying by the time step dt_k, i.e. the linear velocity already represents a linear position, and therefore only requires a linear position frame transformation. A linear position frame transformation introduces significantly less error into the state estimate compared to a linear velocity frame transformation, since a linear position frame transformation is only dependent on the angular position. A full linear velocity frame estimate requires angular position and angular velocity, e.g. equation (60). There are multiple methods this transformation could be implemented into the state estimator; probably the most straightforward method is to integration the frame transformation directly into the state-transition equations. This is given in equation (49).

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \\ \dot{z}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \cos(\psi_k) \cdot \dot{x}_k \cdot dt_k - \sin(\psi_k) \cdot \dot{y}_k \cdot dt_k \\ y_k + \sin(\psi_k) \cdot \dot{x}_k \cdot dt_k + \cos(\psi_k) \cdot \dot{y}_k \cdot dt_k \\ \psi_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{\psi}_k \end{bmatrix} \tag{49}$$

Where
- x_k is the linear position of the robot
- y_k is the linear position of the robot
- ψ_k is the angular position of the robot
- $\dot{x}_k$ is the linear velocity of the robot
- $\dot{y}_k$ is the linear velocity of the robot
- $\dot{\psi}_k$ is the angular velocity of the robot

To match the state-transition model in equation (49), the hybrid frame state estimator measurement matrix is a 6×1 vector given in equation (50).

$$\vec{z}_{h_k} = \begin{cases} x_{G_m} \\ y_{G_m} \\ \hat{\psi}_r \\ \hat{\dot{x}}_r \\ \hat{\dot{y}}_r \\ \hat{\dot{\psi}}_{I_r} \end{cases}$$

Where

$\vec{z}_{r_k}$ is the state measurement vector for the hybrid frame state estimator at time k

x_{G_m} is the is the most current GNSS linear position measurement along the x-axis

y_{G_m} is the is the most current GNSS linear position measurement along the y-axis (50)

$\hat{\psi}_r$ is the most current angular position update about the robot body frame z-axis from the robot body frame Kalman filter; note that $\psi_{I_r} = \psi_{I_m}$ for our purposes (see section 3.1.4)

$\hat{\dot{x}}_r$ is the most current linear velocity update along the robot body frame x-axis from the robot body frame Kalman filter

$\hat{\dot{y}}_r$ is the most current linear velocity update along the robot body frame y-axis from the robot body frame Kalman filter

$\hat{\dot{\psi}}_{I_r}$ is the most current angular velocity update about the robot body frame z-axis from the robot body frame Kalman filter; note that $\dot{\psi}_{I_r} = \dot{\psi}_{I_m}$ for our purposes (see section 3.1.4)

Also note that because the angular position, linear velocity, and angular velocity states of equation (48) have already been estimated in the robot frame state estimator, so the hybrid frame state-transition equations should not change anything about this information. Thus, their respective terms in the process noise covariance matrix of the hybrid map frame estimator should be tuned to be arbitrarily large (compared to the measurement noise covariance). The revised sliding adaptive measurement noise covariance matrix $\vec{R}_{k_{adaptive}}$ for the hybrid frame state estimator is given in equation (51).

$$\vec{R}_{k_{adaptive}} = \vec{R}_k \cdot \vec{G}$$

$$\Rightarrow \vec{R}_{k_{adaptive}} = \begin{bmatrix} R_x G_x & 0 & 0 & 0 & 0 & 0 \\ 0 & R_y G_y & 0 & 0 & 0 & 0 \\ 0 & 0 & R_\psi G_\psi & 0 & 0 & 0 \\ 0 & 0 & 0 & var(\dot{x}_P) & 0 & 0 \\ 0 & 0 & 0 & 0 & var(\dot{y}_P) & 0 \\ 0 & 0 & 0 & 0 & 0 & R_{\dot{\psi}} G_{\dot{\psi}} \end{bmatrix}$$

Where

$\vec{R}_{k_{adaptive}}$ is the revised measurement noise covariance matrix incorporating the sliding adaptive covariance gain (51)

$\vec{R}_k$ is the measurement noise covariance matrix

$\vec{G}$ is the sliding adaptive covariance gain vector

$var(\dot{x}_P)$ is the variance of the linear velocity pseudo-measurement along the x-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

$var(\dot{y}_P)$ is the variance of the linear velocity pseudo-measurement along the y-axis based on the weighted average of the forward-difference of GNSS linear position measurements and trapezoidal-method of IMU linear acceleration measurements

3.5. Putting it all together: the full cascading dual-frame motion-tracking state estimator for velocity-denied robot localization

In this section, I discuss the full cascading, dual-frame motion-tracking state estimator. A flow chart of its operating is shown in Figure 10.

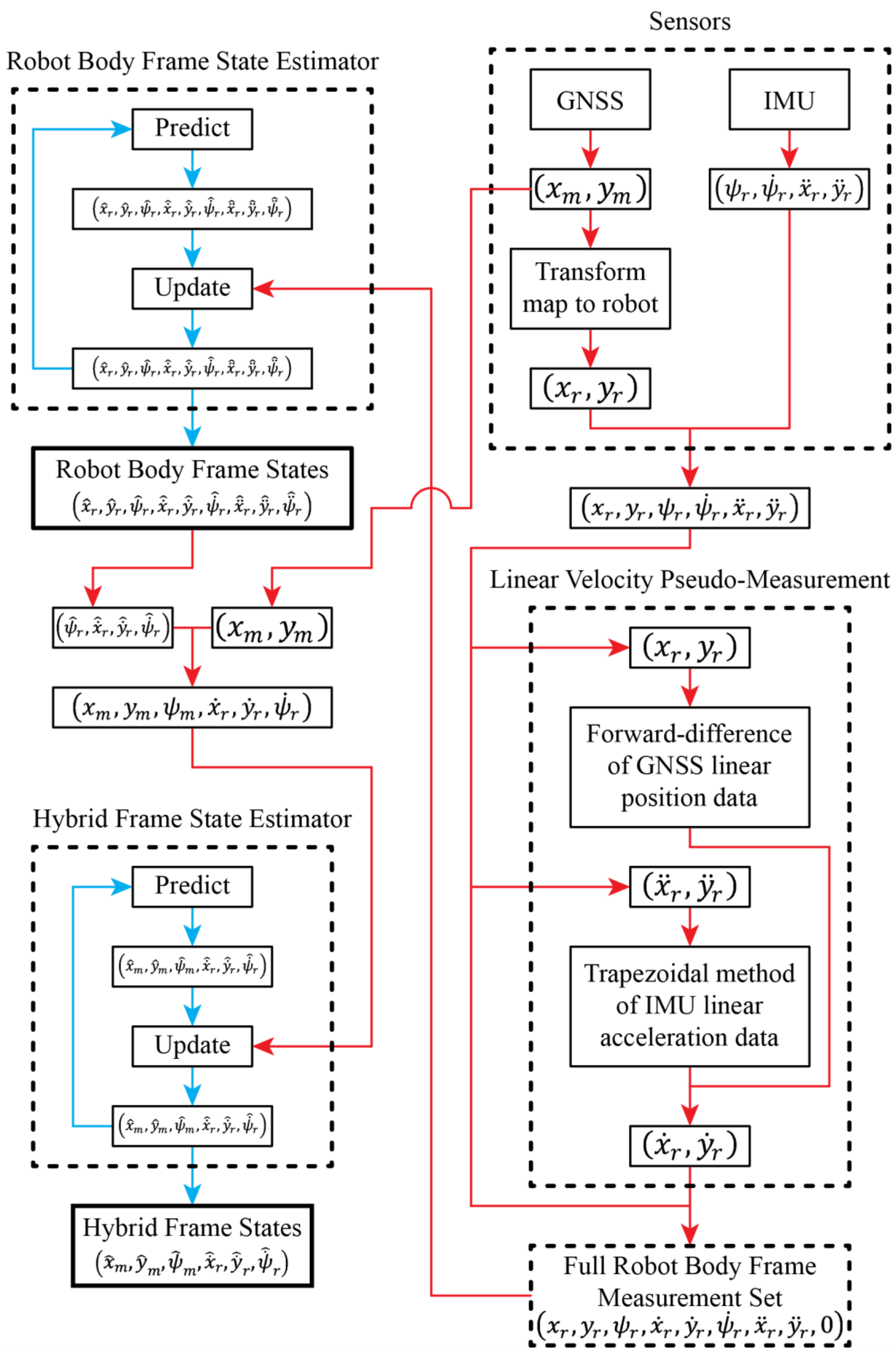
Robot Body Frame State Estimator
Predict
$(\hat{x}_r, \hat{y}_r, \hat{\psi}_r, \hat{\dot{x}}_r, \hat{\dot{y}}_r, \hat{\dot{\psi}}_r, \hat{\ddot{x}}_r, \hat{\ddot{y}}_r, \hat{\ddot{\psi}}_r)$
Update
$(\hat{x}_r, \hat{y}_r, \hat{\psi}_r, \hat{\dot{x}}_r, \hat{\dot{y}}_r, \hat{\dot{\psi}}_r, \hat{\ddot{x}}_r, \hat{\ddot{y}}_r, \hat{\ddot{\psi}}_r)$
Sensors
GNSS
IMU
(x_m, y_m)
$(\psi_r, \dot{\psi}_r, \ddot{x}_r, \ddot{y}_r)$
Transform map to robot
(x_r, y_r)
Robot Body Frame States
$(\hat{x}_r, \hat{y}_r, \hat{\psi}_r, \hat{\dot{x}}_r, \hat{\dot{y}}_r, \hat{\dot{\psi}}_r, \hat{\ddot{x}}_r, \hat{\ddot{y}}_r, \hat{\ddot{\psi}}_r)$
$(x_r, y_r, \psi_r, \dot{\psi}_r, \ddot{x}_r, \ddot{y}_r)$
$(\hat{\psi}_r, \hat{\dot{x}}_r, \hat{\dot{y}}_r, \hat{\dot{\psi}}_r)$
(x_m, y_m)
$(x_m, y_m, \psi_m, \dot{x}_r, \dot{y}_r, \dot{\psi}_r)$
Linear Velocity Pseudo-Measurement
(x_r, y_r)
Forward-difference of GNSS linear position data
$(\ddot{x}_r, \ddot{y}_r)$
Trapezoidal method of IMU linear acceleration data
$(\dot{x}_r, \dot{y}_r)$
Hybrid Frame State Estimator
Predict
$(\hat{x}_m, \hat{y}_m, \hat{\psi}_m, \hat{\dot{x}}_r, \hat{\dot{y}}_r, \hat{\dot{\psi}}_r)$
Update
$(\hat{x}_m, \hat{y}_m, \hat{\psi}_m, \hat{\dot{x}}_r, \hat{\dot{y}}_r, \hat{\dot{\psi}}_r)$
Hybrid Frame States
$(\hat{x}_m, \hat{y}_m, \hat{\psi}_m, \hat{\dot{x}}_r, \hat{\dot{y}}_r, \hat{\dot{\psi}}_r)$
Full Robot Body Frame Measurement Set
$(x_r, y_r, \psi_r, \dot{x}_r, \dot{y}_r, \dot{\psi}_r, \ddot{x}_r, \ddot{y}_r, 0)$

Figure 10: Full cascading dual-frame motion-tracking state estimator.

Following is a bulleted list that roughly reflects the chronological order of the full state estimation algorithm:

- At discrete-time k, the robot body frame Kalman filter calculates a state vector prediction using equation (3) with state-transition model in equation (12), and error covariance matrix prediction using equation (4). It uses the state estimate prediction and error covariance matrix from the robot body frame Kalman filter at discrete-time $k - 1$.
- The state estimation algorithm retrieves sensor data from the GNSS and IMU sensors. It calculates the time since the last update of each sensor, which it uses to calculate the sliding adaptive covariance gains for each sensor using equation (25) to account for decays in measurement accuracy due to elapsed time and to smooth the output data in real time.
- The state estimation algorithm transforms the GNSS linear position data from the map frame to the robot body frame using equations (15) through (19).
- The state estimation algorithm calculates the forward-difference of the transformed (from map to robot body frame) GNSS linear position measurements using equation (36), and the trapezoidal method of the IMU linear acceleration measurements using equation (39).
- The GNSS linear position forward-difference is combined with the IMU linear acceleration trapezoidal method using equations (43). Their covariances are calculated using equation (44).
- The measurement vector is updated with linear velocity pseudo-measurements using equation (47), and the measurement noise covariance matrix is updated with the linear velocity covariance data and sliding adaptive covariance gains using equation (46).
- The robot body frame Kalman filter calculates a state vector update using equation (6), and error covariance matrix update using equation (7). It uses the state vector prediction in equation (3), error covariance matrix prediction in equation (4), measurement matrix in equation (45), and updated measurement noise covariance in equation (46).

- The hybrid frame Kalman filter calculates a state vector prediction using equation (3) with state-transition model in equation (49), and error covariance matrix prediction using equation (4). It uses the state estimate prediction and error covariance matrix from the hybrid frame Kalman filter at discrete-time $k - 1$.
- The hybrid frame Kalman filter measurement vector uses linear velocity terms from the state estimate output from the robot body frame Kalman filter state vector update, alongside GNSS linear position measurements, IMU angular position, and IMU angular velocity measurements.
- The hybrid frame Kalman filter calculates a state vector update using equation (6), and error covariance matrix update using equation (7). It uses the state vector prediction in equation (3), error covariance matrix prediction in equation (4), measurement matrix in equation (50), and updated measurement noise covariance in equation (51).
- The pose and twist components of the state vector update from the robot body frame Kalman filter $\left(\hat{x}_r, \hat{y}_r, \hat{\psi}_r, \hat{\dot{x}}_r, \hat{\dot{y}}_r, \hat{\dot{\psi}}_r\right)$ and the pose components of the state vector update from the hybrid frame Kalman filter $\left(\hat{x}_m, \hat{y}_m, \hat{\psi}_m\right)$ represent the most accurate post and twist estimates available from this state estimation algorithm. The Kalman filter waits for a pre-specified dt_k, and repeats.

4. Validating the revised algorithm

In this chapter, I will validate the performance of our cascading, dual-frame, motion-tracking state estimation algorithm by evaluating its performance against a robot simulation environment, and then against real sensor data. I are interested in evaluating the performance two key elements of the state estimation algorithm: (1) the accuracy of the linear position state prediction in between GNSS sensor updates, or the "dead-reckoning" performance of the state estimator, and (2) the accuracy of the unobserved linear velocity state estimate.

Note that I am not particularly interested in the absolute estimation accuracy of the measured states, i.e. linear position, angular position, angular velocity, and linear acceleration. Without additional state information in the form of competitive, complementary, and/or cooperative sensor information for any of the measured states (from our discussion in chapter 2), it is unrealistic to expect that the state estimation algorithm will make significant improvements to state accuracy. The state estimator does offer some smoothing performance for individual state measurements as it probabilistically fuses individual measurements with their time history. This smoothing can be considered an improvement over measurement accuracy if those measurements were truly stationary unimodal Gaussian distributions; however, the states are clearly not stationary in a mobile robot, and the assumption of a unimodal Gaussian distribution will only go so far in practice. Therefore, I am significantly more interested in analyzing the performance of the *unobserved* state information; i.e. linear position state estimation accuracy *in between* the GNSS sensor one-second-long refresh rate (the dead reckoning accuracy), and the accuracy of the unobserved linear velocity states.

4.1. Estimator performance on simulated sensor data

To initially validate the state estimation techniques described in the previous section, I developed a comprehensive 2D robot simulation based on the Euler numerical method.

This simulation approximates the dynamics of a robot as a point mass with linearized mass (m, I), damping $(b_{surge}, b_{sway}, b_{yaw})$, and forcing inputs $\left(F_{surge}, F_{sway}, F_{yaw}\right)$. The simulation allows for varying the forcing inputs at any time step and generates "ground truth" data for position, velocity, and acceleration states. It also generates sensor data with configurable noise and refresh rate that representative of a GNSS+IMU sensor system. This noisy sensor data is then fed into our cascading, dual-frame, motion-tracking state estimator; the outputs of the state estimator can then be compared against the simulation ground truth to evaluate accuracy. Details regarding equations of motion and Euler simulation can be found in appendix 6.5 , and its source code can be found in supplementary materials 8.1.

4.1.1. <u>Straight-line simulation</u>

In this first scenario, the robot accelerates in a straight line along the map frame x-axis (east). The map frame xy-position "top down view" is given in Figure 11; the robot body frame states plotted versus time are given in Figure 12; and the hybrid frame states plotted versus time are given in Figure 13.

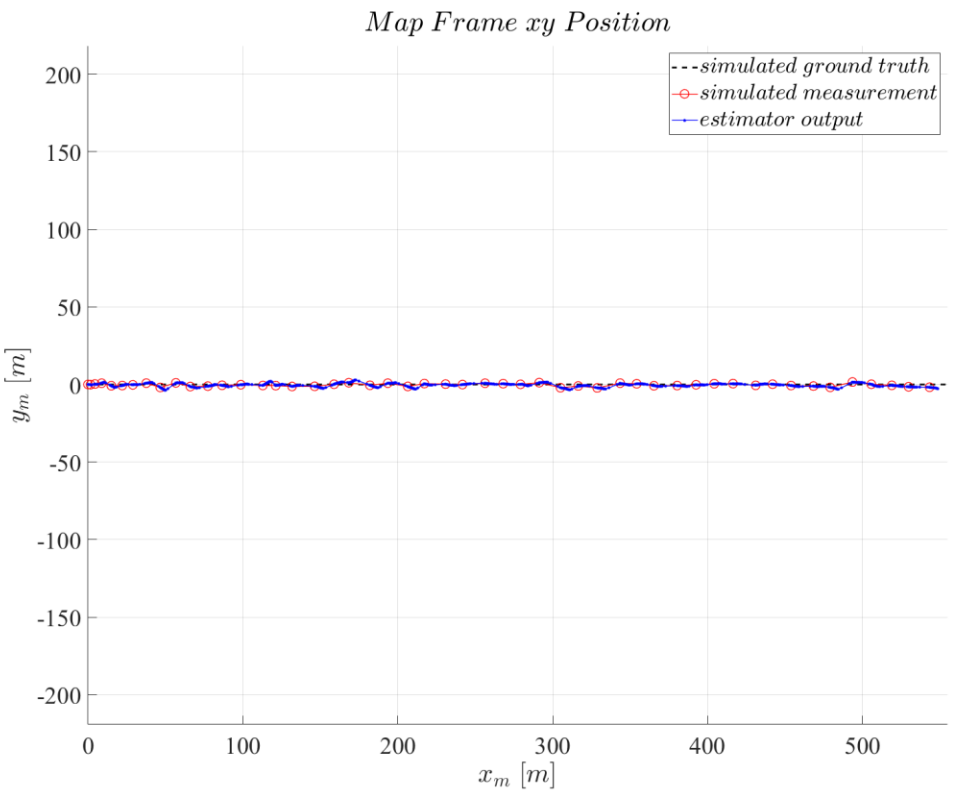

Figure 11: Straight-line simulation map frame xy-position “top-down” view.

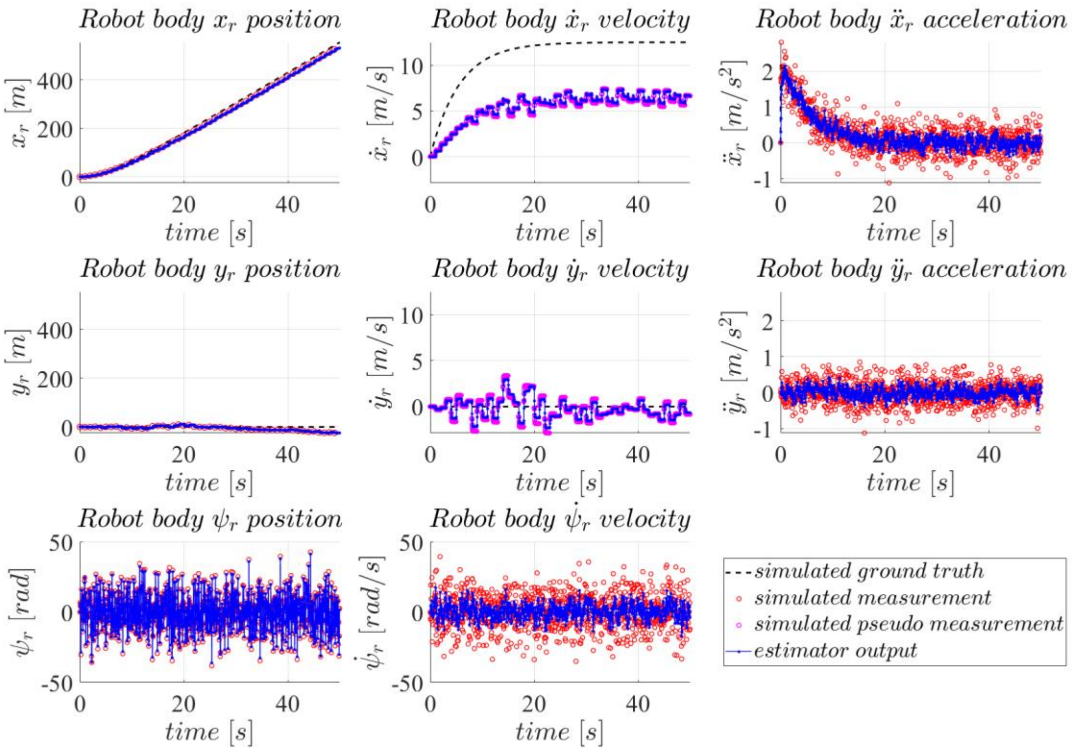

Figure 12: Straight-line simulation robot body frame states plotted versus time.

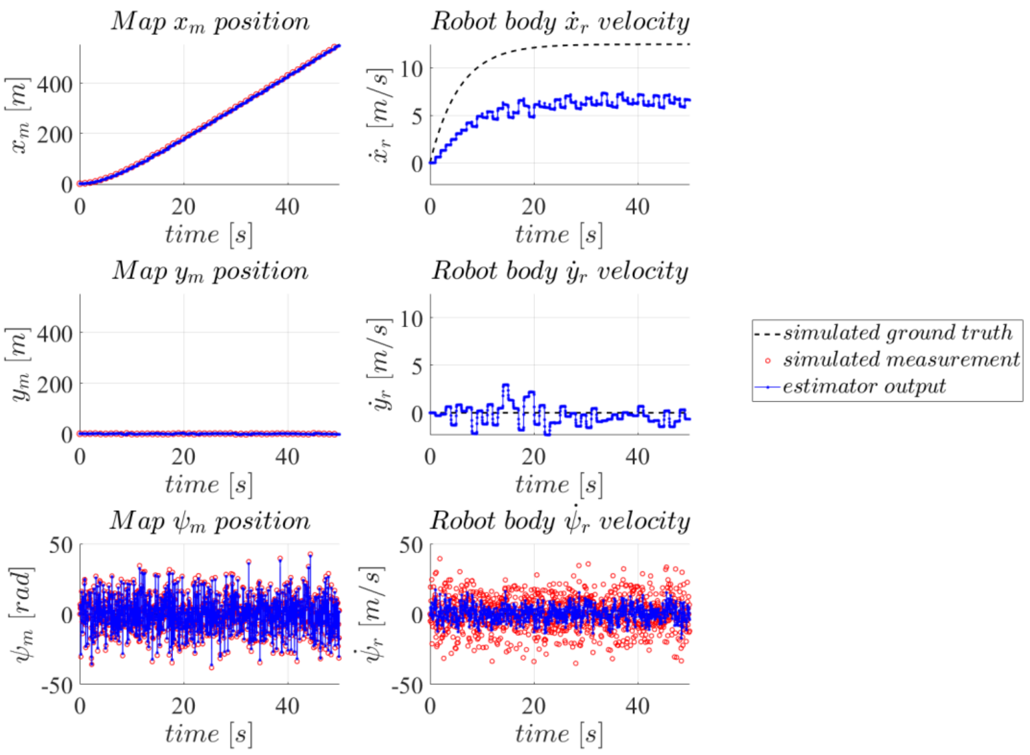

Figure 13: Straight-line simulation hybrid frame states plotted versus time.

In Figure 11, the robot starts at (0,0) in the map frame, and moves in the positive x-direction. In Figure 12, the robot body frame estimator outputs are plotted against their simulated ground truth and simulated measurements. In this simulation, the robot increases in its x_r linear position (top left), while staying relatively still in its y_r linear position (middle left), and ψ_r angular position (bottom left), which is good, as no sway input force or yaw input moment was commanded. The $\dot{x}_r$ linear velocity (top middle) increases to a steady-state value and remains constant, as expected when employing a non-zero drag coefficient; and the $\dot{y}_r$ linear velocity and $\dot{\psi}_r$ angular velocity remains at zero as expected. The $\ddot{x}_r$ linear acceleration (top right) starts at a maximum and decreases to zero as expected, and the $\ddot{y}_r$ linear acceleration (middle right) remains at zero. In Figure 13, the hybrid frame estimator outputs are similarly plotted against their simulated ground truth and simulated measurements. The x_m linear position (top left) and y_m linear position (middle left) are the only plots that reflect the new information from the hybrid frame estimator, are plotted against each other in Figure 11. The $\psi_r = \psi_m$ angular position, $\dot{x}_r$ linear velocity, $\dot{y}_r$ linear velocity, and $\dot{\psi}_r = \dot{\psi}_m$ angular velocity is identical to their respective counterparts in the robot body frame; they are listed again for continuity.

One important takeaway from the straight-line acceleration simulation is the significant steady-state error in the linear velocity state (Figure 12, top middle); the pseudo-measurements are significantly underestimating the actual velocity of the robot. This results from the significant latency in the forward-difference velocity pseudo-measurement; because the GNSS sensor only updates at 1 Hz, by the linear measurement is always one second behind (at a *minimum*) resulting in a systematic under-estimation of linear velocity via the forward-difference. I can demonstrate this in two ways, first by artificially increasing the GNSS refresh rate while keeping all other variables constant. This is performed in Figure 14 for a GNSS refresh rate of 2 Hz, Figure 15 for a GNSS refresh rate of 4 Hz, and Figure 16 for a GNSS refresh rate of 8 Hz (note that the default GNSS refresh rate is 1 Hz, as discussed in Table 3).

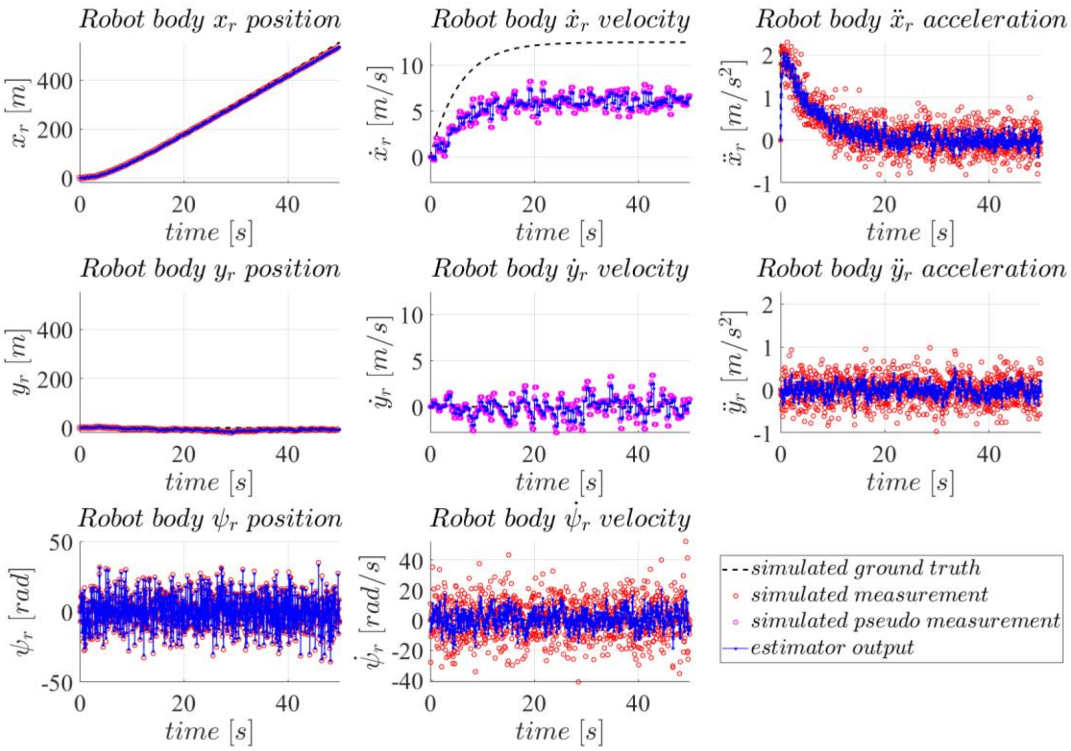

Figure 14: Straight-line simulation robot body frame states, where GNSS refresh rate is increased from 1 Hz (default) to 2 Hz.

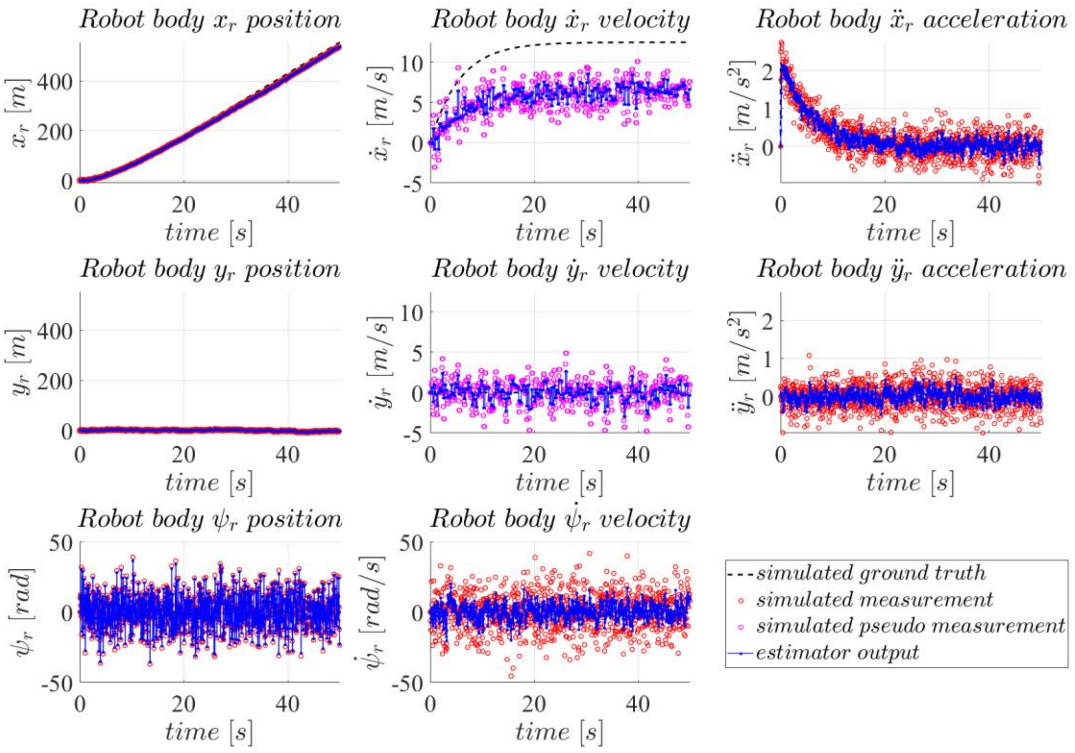

Figure 15: Straight-line simulation robot body frame states, where GNSS refresh rate is increased from 1 Hz (default) to 4 Hz.

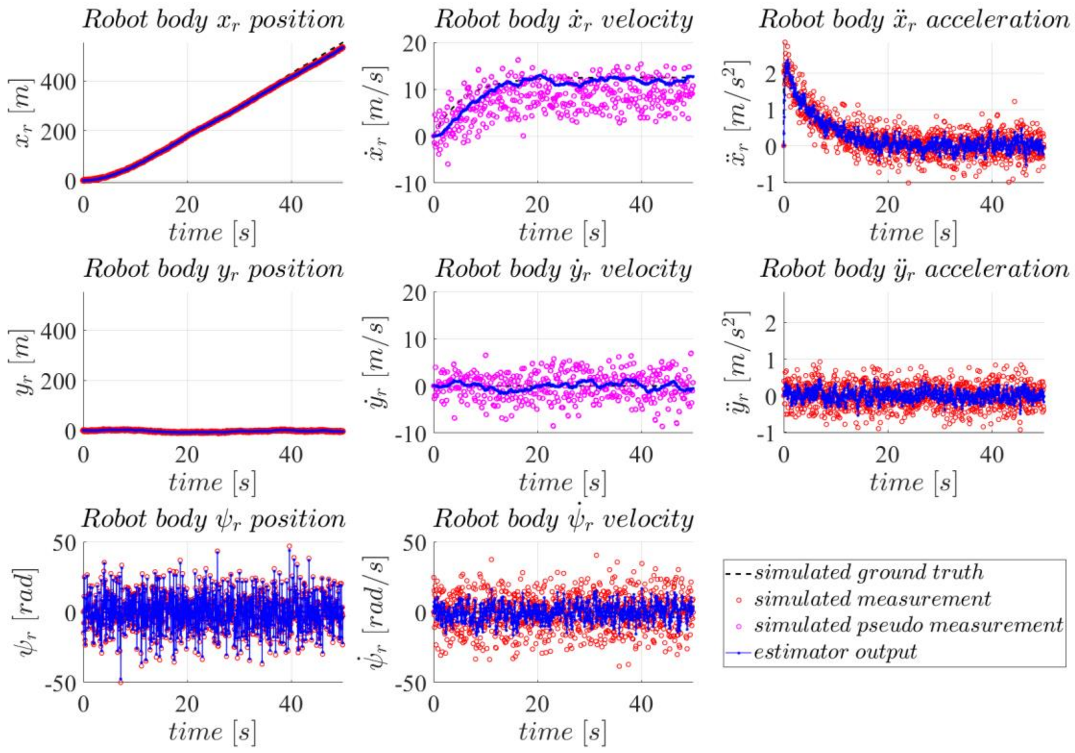

Figure 16: Straight-line simulation robot body frame states, where GNSS refresh rate is increased from 1 Hz (default) to 8 Hz.

In Figure 14 through Figure 16, note how the steady-state error of the $\dot{x}_r$ linear velocity pseudo-measurements (top middle) decreases. This tells us that a *faster* GNSS (or generalized linear position sensor) refresh rate significantly improves the steady-state error of the pseudo-linear velocity measurement, as expected.

Also note the significant increase in noise in the linear velocity pseudo-measurement as the refresh rate of the GNSS sensor increases, despite no change in variance of the raw GNSS sensor measurements. Although non-intuitive, this behavior is actually predicted by our symbolic formulation of the variance for the forward-difference method, which is discussed in detail in appendix 6.3. Because the square of the time step dt^2 is in the denominator of the variance of the forward-difference method in equation (67), as the time step between measurements decreases below 1 second, the total variance of the forward-difference method *increases*. In our case, the estimator does a reasonably good job filtering these noisy measurements; however, if employing a faster refresh rate linear position sensor (e.g. interpolating GNSS sensor, or if receiving linear position measurements from another source like vision, laser scan, etc.), one may consider increasing the number linear position measurements n to include in the moving mean of the forward-difference method in equation (35).

This leads us to our second point of discussion; how many linear position measurements should be used in the forward-difference calculation for a standard 1 Hz GNSS? In Figure 17 I evaluate the same straight-line acceleration simulation for $n = 4$, in Figure 18, for $n = 8$, and Figure 19 for $n = 16$.

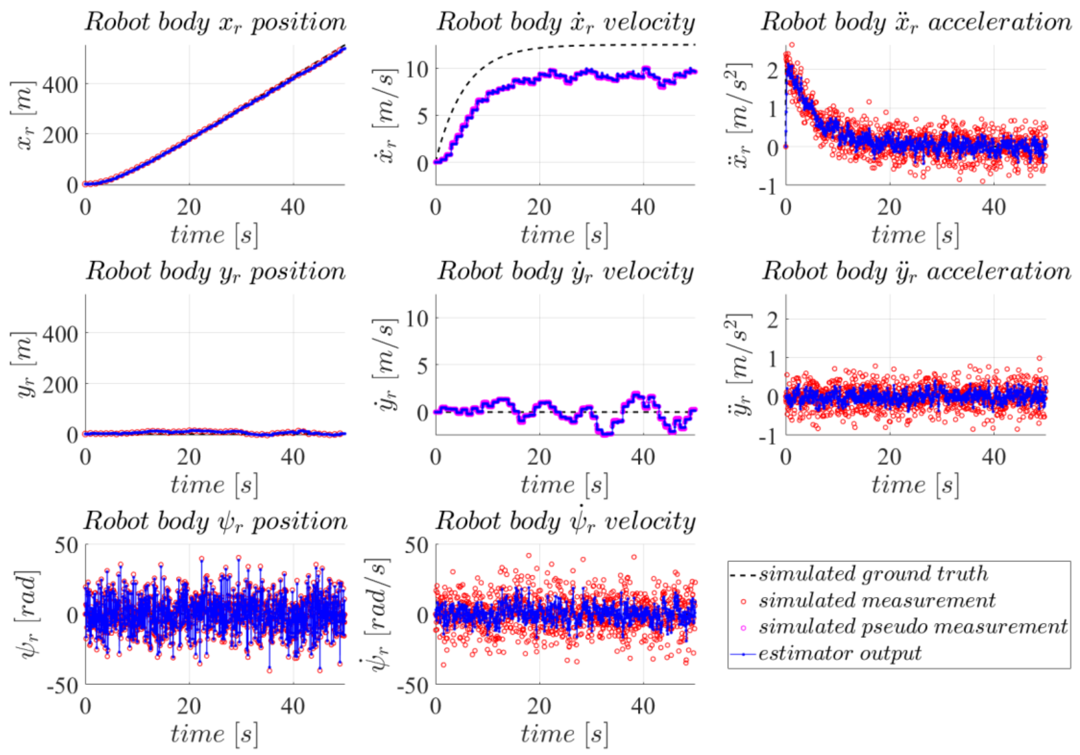

Figure 17: Straight-line simulation robot body frame states, where n GNSS linear position measurements in the forward-difference calculation is increased from 2 (default) to 4.

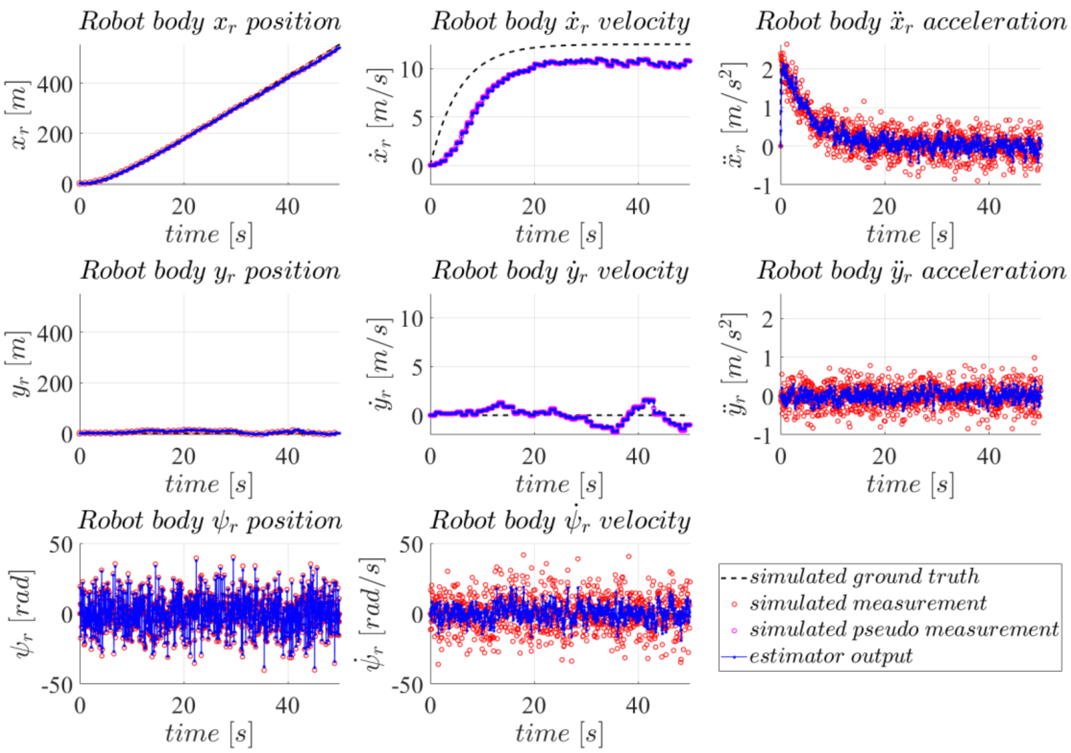

Figure 18: Straight-line simulation robot body frame states, where n GNSS linear position measurements in the forward-difference calculation is increased from 2 (default) to 8.

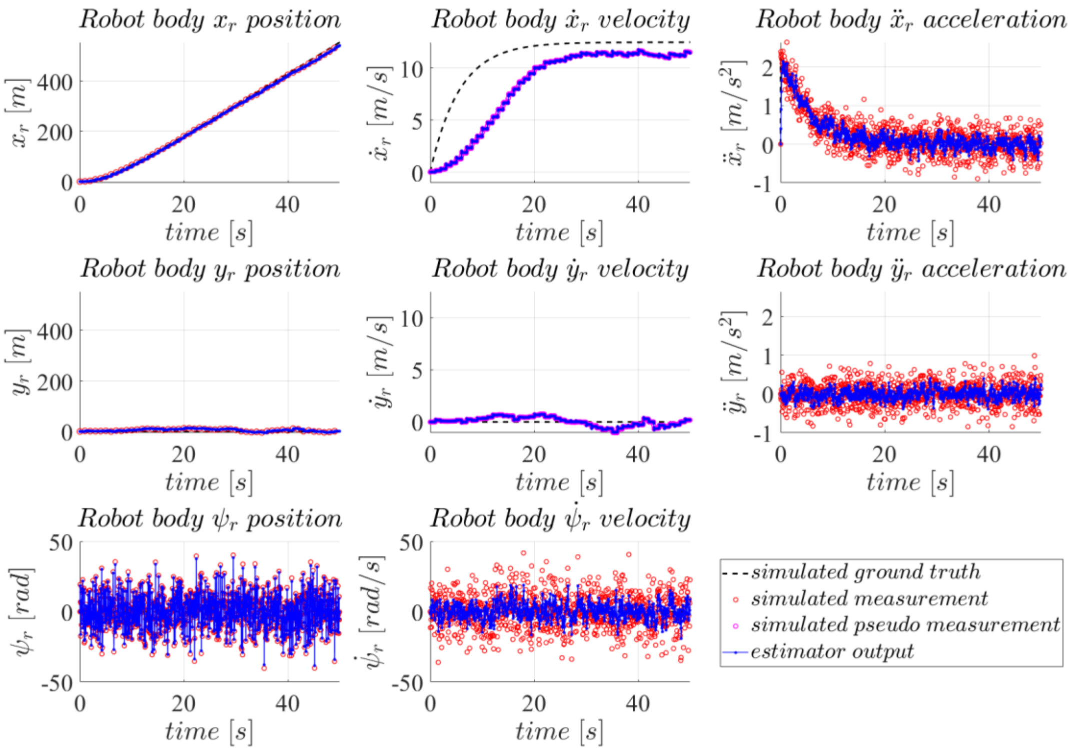

Figure 19: Straight-line simulation robot body frame states, where n GNSS linear position measurements in the forward-difference calculation is increased from 2 (default) to 16.

Comparing Figure 17 through Figure 19 to the robot body frame states from the original simulation in Figure 12, the simulation demonstrates that increasing n decreases the variance and the steady-state error of the linear-velocity pseudo-measurements (both good things), but at the cost of increasing the response time (time constant) of the linear-velocity state pseudo-measurement. Depending on the refresh rate of the linear position sensor, the answer to this question will change from system to system, but, for a transient mobile robot system, a faster response of a reasonable linear velocity measurement is likely more useful than minimizing the steady state error. For the 1 Hz GNSS linear position sensor employed in this work, $n = 2$ (the minimum possible) gives the best balance of linear velocity characterization, and response time.

4.1.2. Forward, backward, and forward simulation

In this simulation, the robot will start by driving forward along the map frame x-axis (east), then at $t = 15\ s$ the robot will apply an equivalent reverse thrust, backing up along the same trajectory, then at $t = 30\ s$, the robot will apply an equivalent forward thrust, moving forward along the same trajectory. The map frame xy-position is given in Figure 20; the robot body frame states plotted versus time are given in Figure 21; and the hybrid frame states plotted versus time are given in Figure 22.

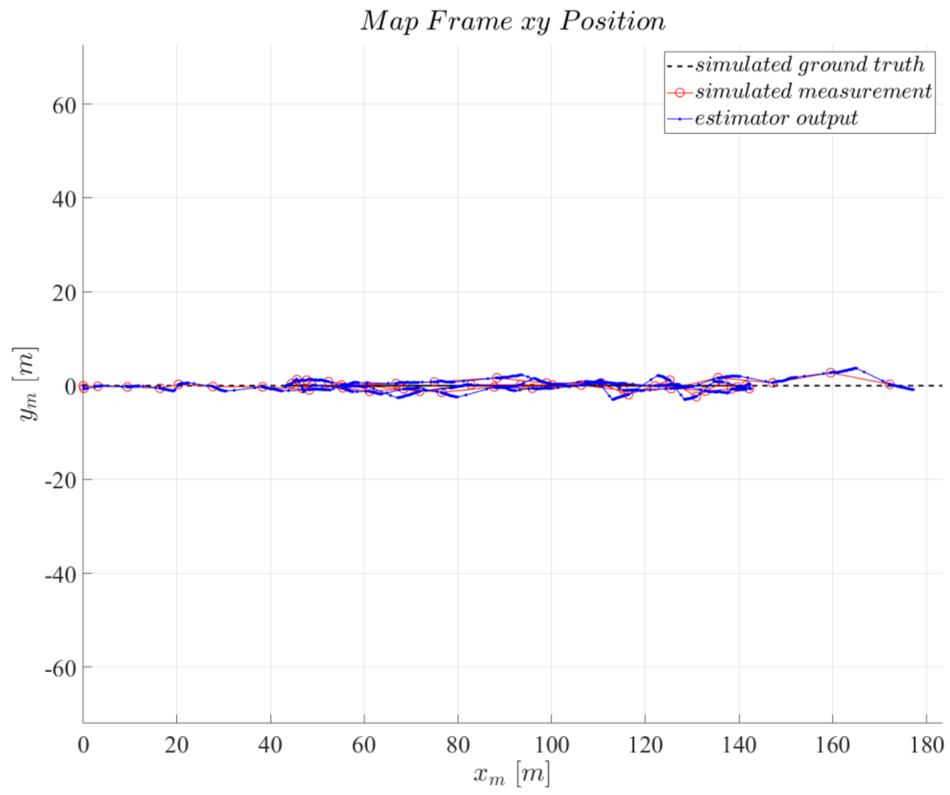

Figure 20: Forward-backward-forward simulation map frame xy-position “top-down” view.

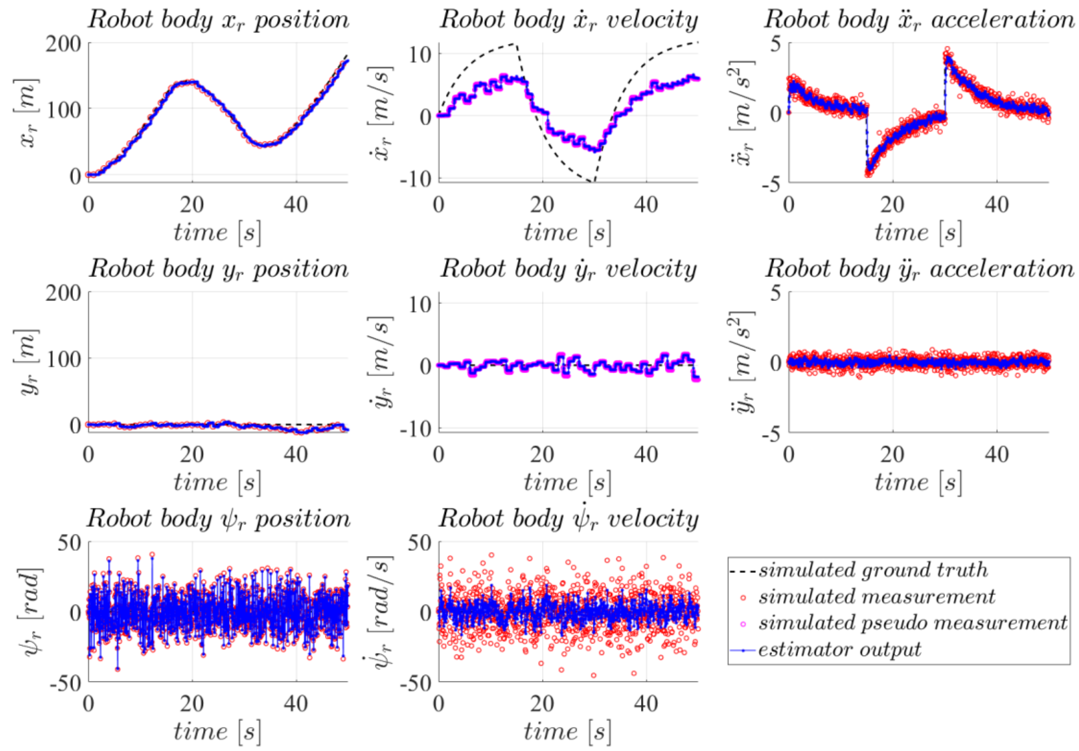

Figure 21: Forward-backward-forward simulation robot body frame states plotted versus time.

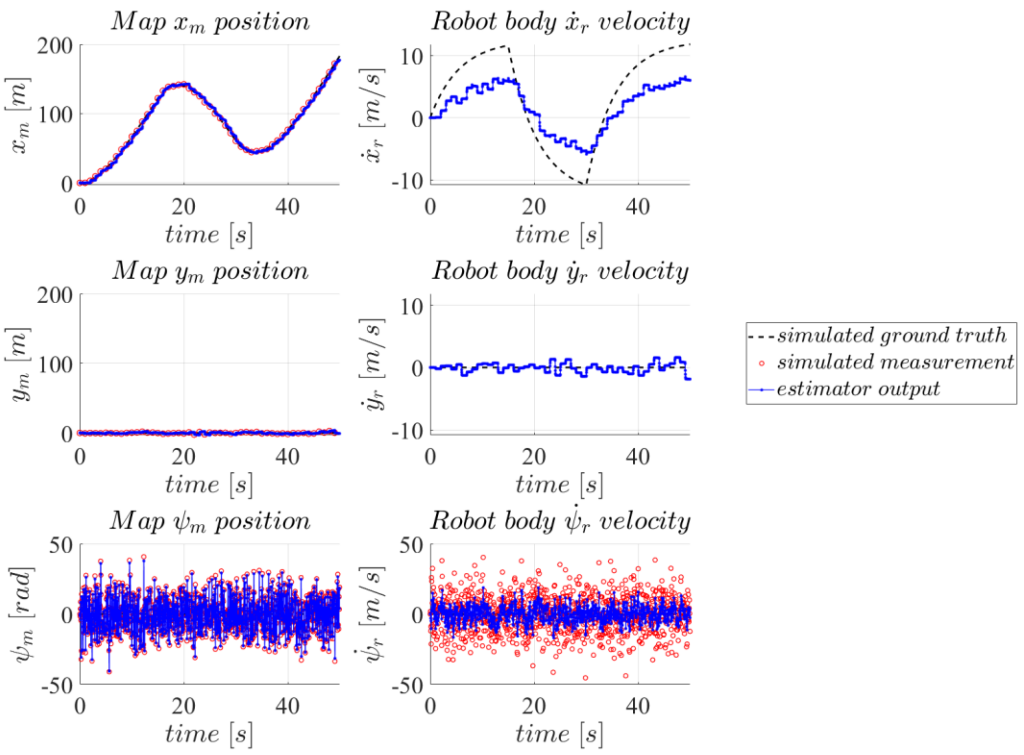

Figure 22: Forward-backward-forward simulation hybrid frame states plotted versus time.

As mentioned in the straight-line simulation, the steady state error in the linear velocity state appears to be a systematic offset based on the refresh-rate of the linear position sensor. Increasing the refresh rate of the linear position sensor, or increasing the number of linear position measurements decreases the steady state error; however our ability to increase the refresh rate of the linear position sensor is limited by the sensor system, and increasing the number of linear position measurements has detrimental effects on other parts of the system. This forward-backward-forward simulation supports the idea that this steady state error is systematic and steady, provided that the parameters sensors remain the same. Thusly, this steady-state error can be compensated for using an open-loop bias (multiplier in this case) on the linear velocity pseudo-measurements. Using the 1 Hz GNSS sensor measurements, and 20 Hz IMU sensor measurements used in this simulation, an open-loop linear velocity measurement bias of 1.35 best compensated for the steady-state error. The robot body frame states of the same simulation with the bias implemented is shown in Figure 23.

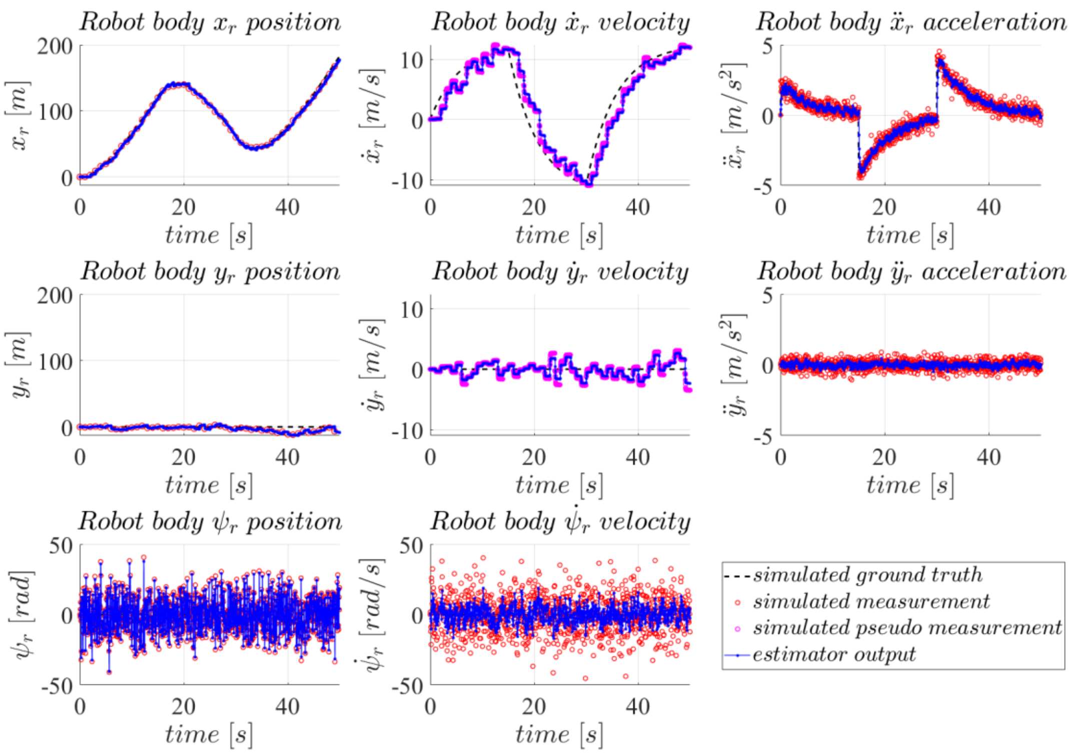

Figure 23: Forward-backward-forward simulation robot body frame states plotted versus time, with an open-loop linear velocity bias of 1.35.

As a thought experiment, I applied this same bias to the straight-line acceleration simulation from the previous section. The robot body frame states if this simulation is shown in Figure 24.

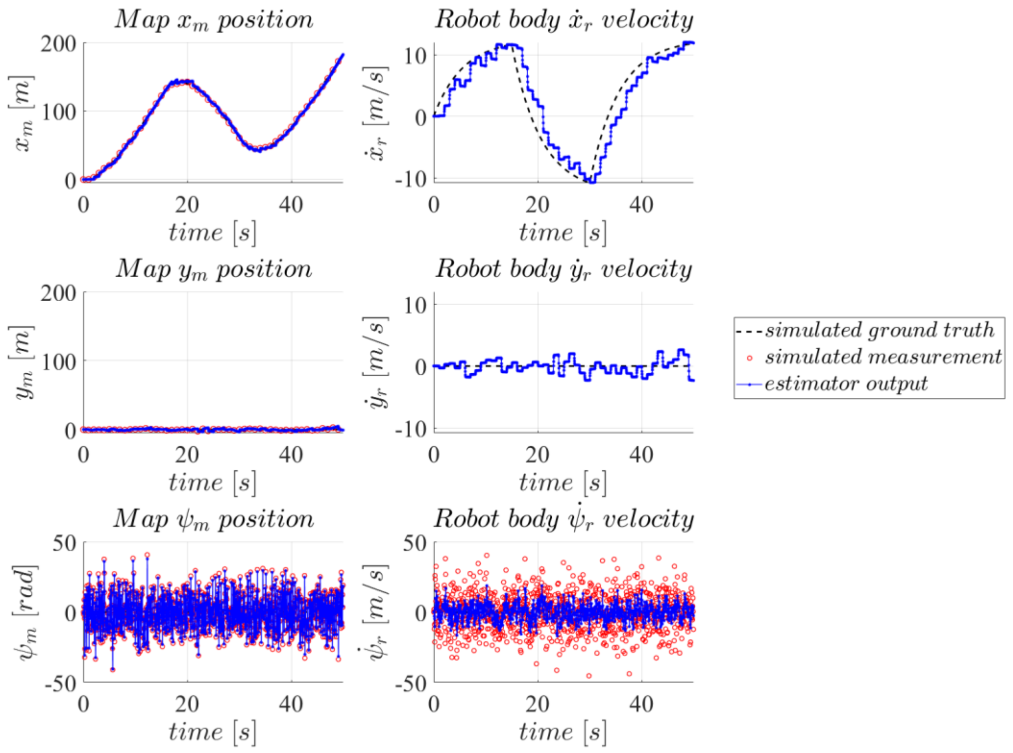

Figure 24: Straight-line simulation robot body frame states, with an open-loop linear velocity bias of 1.35.

The improved linear velocity estimation accuracy from the forward-backward-forward simulation in Figure 23, and the straight-line simulation in Figure 24 clearly demonstrate that the open-loop linear velocity bias helps. This analysis does not necessarily *prove* that this the sensor parameters are the only variables that influence the steady-state error in the linear velocity state, so I will continue using a linear velocity bias of 1.35 in the simulations going forward, while monitoring the linear velocity states for a change in steady-state error.

4.1.3. Circular motion

In this simulation, I impose a positive (counter-clockwise) torque input into the robot in addition to the typical forward thrust. This results in a continuous rotation. The map frame xy-position is given in Figure 25; the robot body frame states plotted versus time are given in Figure 26; and the hybrid frame states plotted versus time are given in Figure 27.

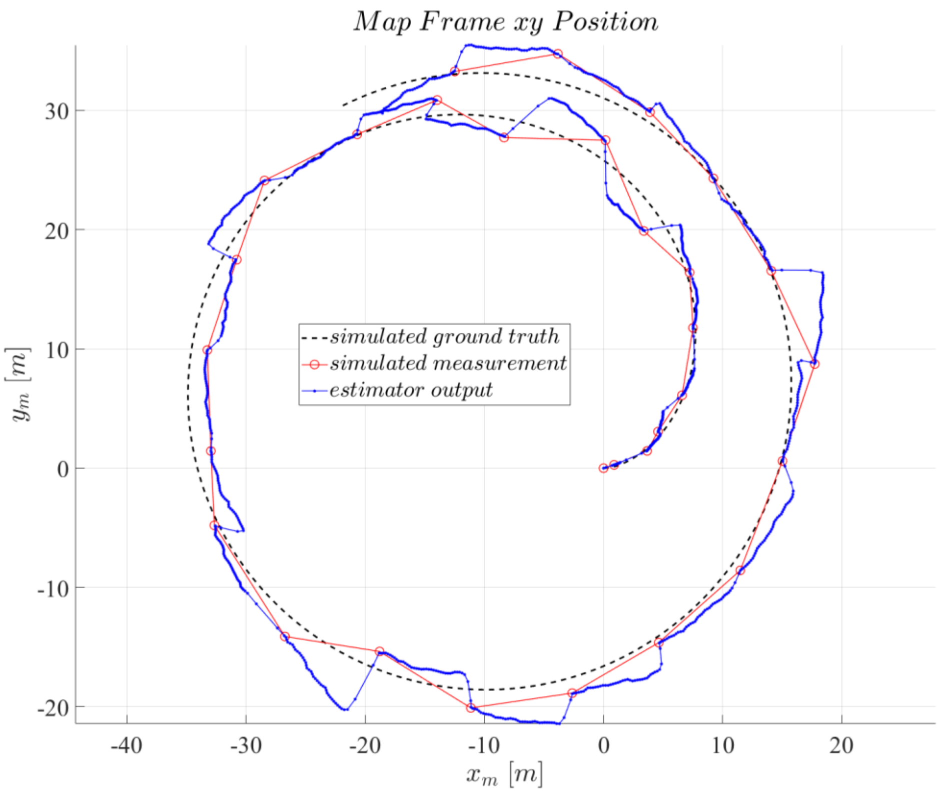

Figure 25: Circular motion simulation map frame xy-position “top-down” view.

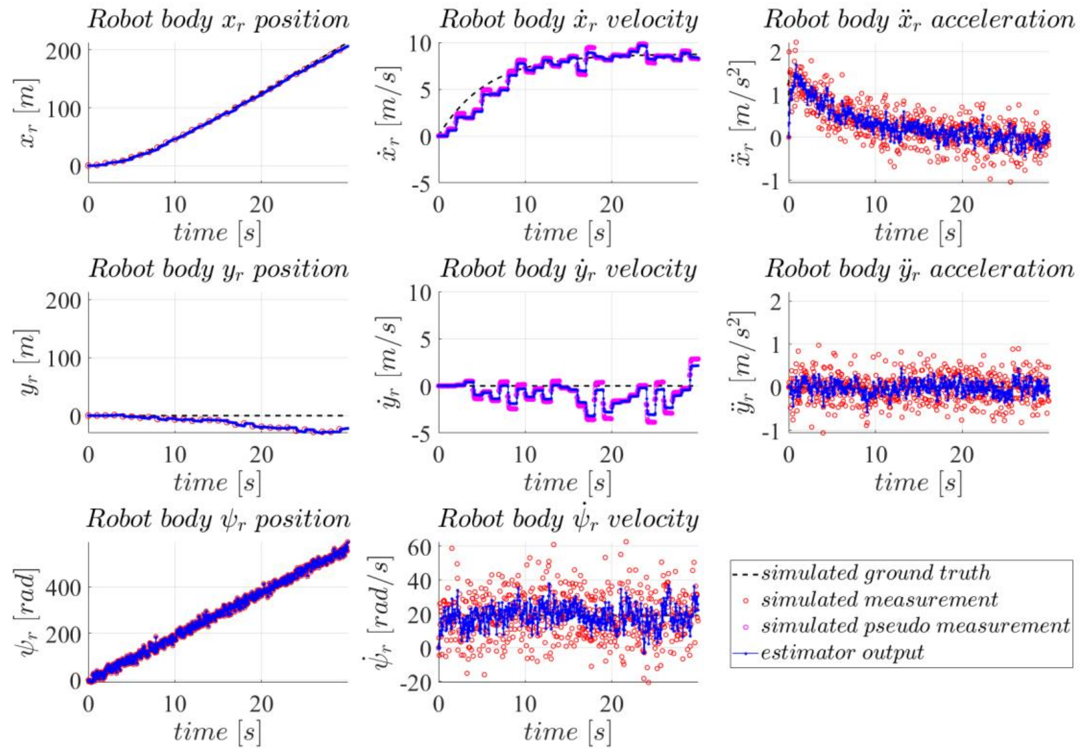

Figure 26: Circular motion simulation robot body frame states plotted versus time.

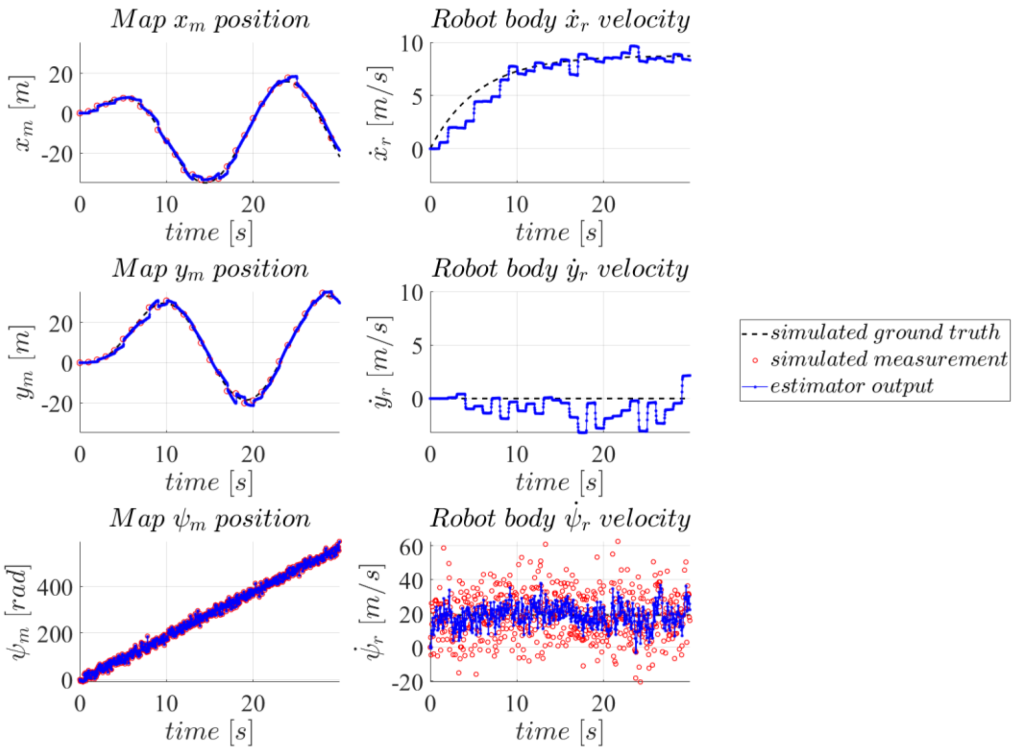

Figure 27: Circular motion simulation hybrid frame states plotted versus time.

This simulation clearly demonstrates how the fusion of ψ angular orientation with the and $\dot{x}_r$ and $\dot{y}_r$ linear velocity enables the estimator to predict the pose of the robot in between GNSS linear position updates. When the GNSS sensor ultimately does update, the state estimate is corrected, and the process continues repeats. Additionally, note that at the beginning of the simulation near position (0,0), the linear velocity estimate is the poorest because of the latency in the forward-difference linear velocity pseudo-measurement mentioned in the previous section; this is illustrated by the poor velocity prediction for the first few GNSS linear position updates. As the steady-state error in the linear velocity pseudo-measurements decreases, the predictive capability of the state estimator improves.

This is also a good opportunity to explore the effect of the sliding adaptive covariance gain on the system. As discussed in section 3.2, the state estimation algorithm employs the default sliding adaptive covariance profiling variables listed in equation (26) for the GNSS linear position measurements, and employ binary adaptive covariance profiling for the IMU measurements. For reference, the default sliding adaptive covariance profiling variables are $G_m = 10^5$; $t_{smooth} = 0.15 \cdot dt$; $t_{scale} = 0.1 \cdot dt$; $t_{1/2} = 0.1 \cdot t_{smooth}$ and $t_{1/2} = 0.1 \cdot t_{scale}$. This provides a small amount of smoothing on the abrupt jumps in GNSS data without significantly impacting the output. In Figure 28, Figure 29, and Figure 30, I run the same circular motion simulation again, but with sliding adaptive covariance profiling disabled; i.e. "binary" adaptive covariance profiling.

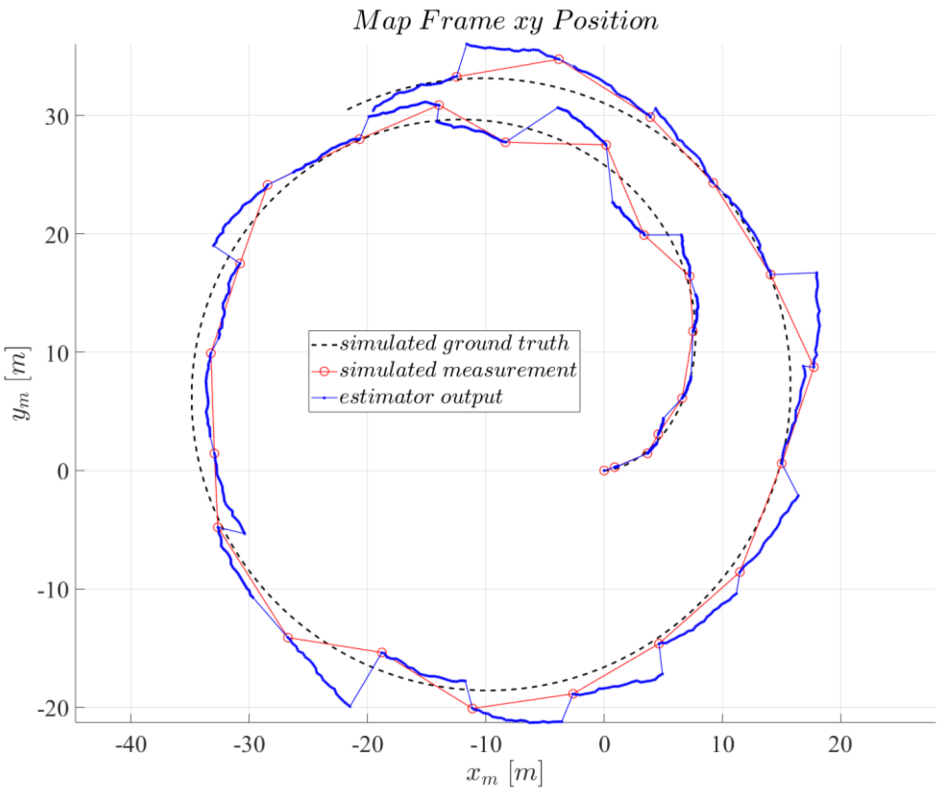

Figure 28: Circular motion simulation map frame xy-position "top-down" view, using binary adaptive covariance profiling.

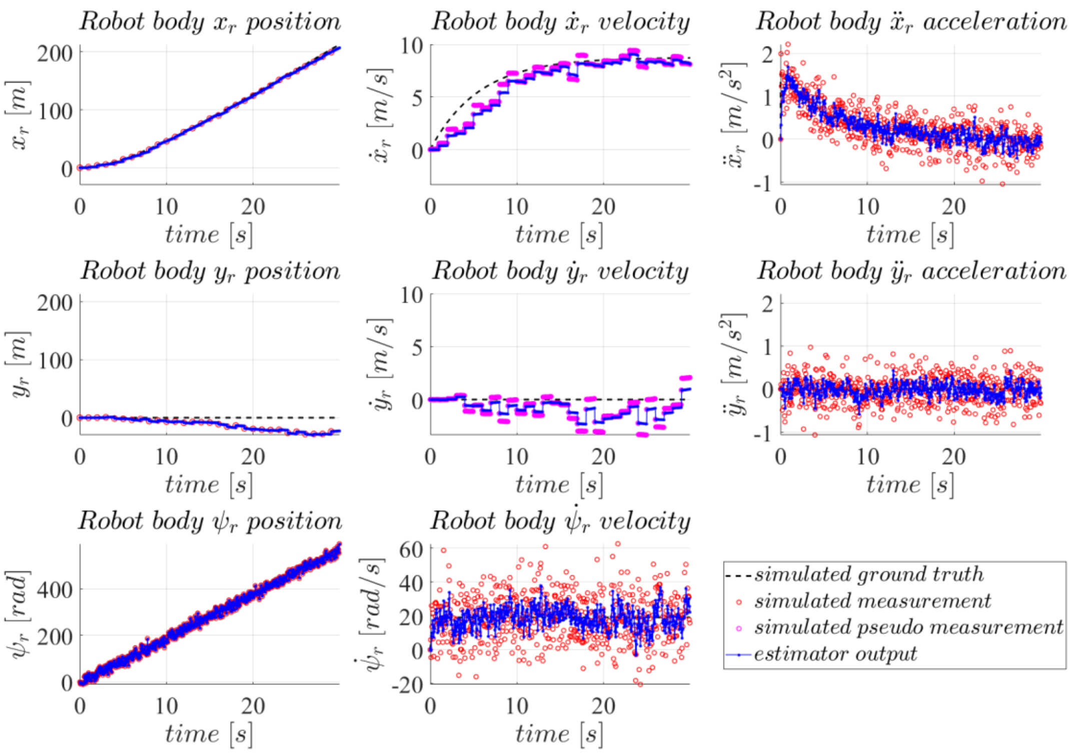

Figure 29: Circular motion simulation robot body frame states plotted versus time, using binary adaptive covariance profiling.

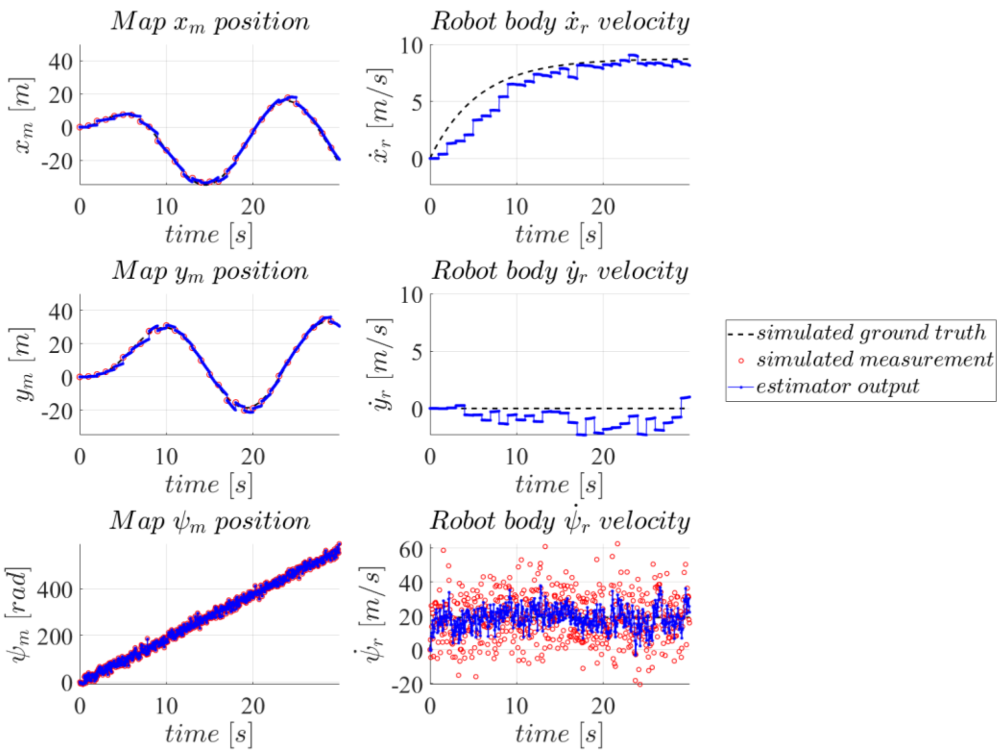

Figure 30: Circular motion simulation hybrid frame states plotted versus time, using binary adaptive covariance profiling.

The estimation outputs from the circular motion simulation in Figure 28, Figure 29, and Figure 30 demonstrate much "harder" jumps in the estimator output between measurement updates. Conversely, in Figure 31, Figure 32, and Figure 33, I run the same circular motion simulation again, but using more aggressive sliding adaptive covariance profiling variables, i.e. $G_m = 10^5$; $t_{smooth} = 0.5 \cdot dt$; $t_{scale} = 0.25 \cdot dt$; $t_{1/2} = 0.1 \cdot t_{smooth}$ and $t_{1/2} = 0.1 \cdot t_{scale}$.

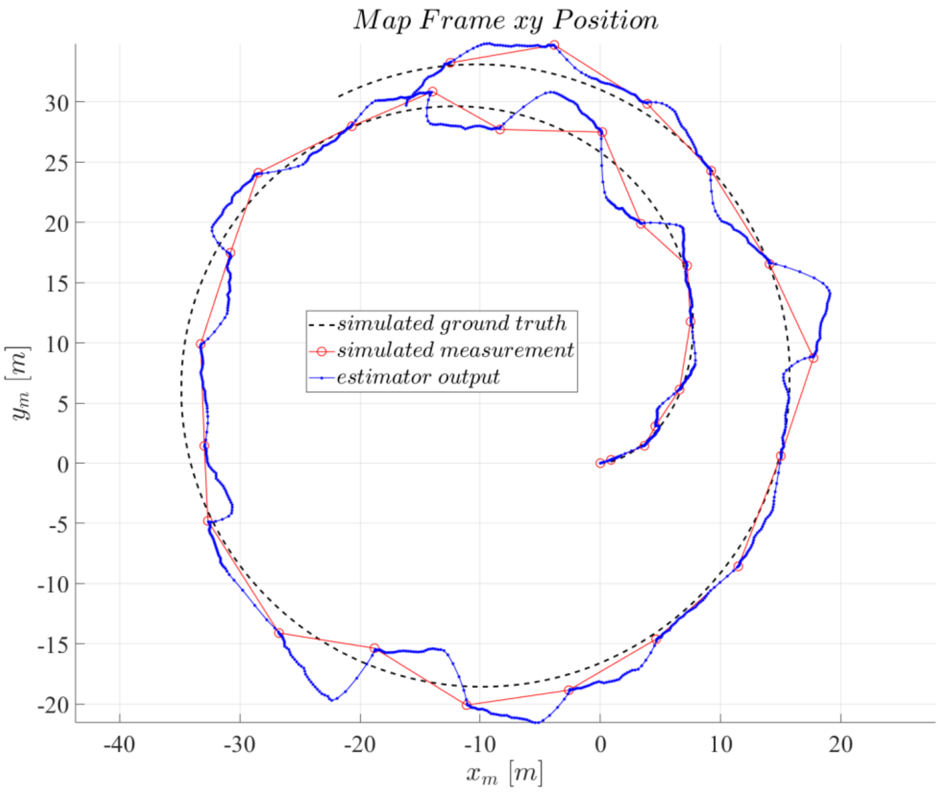

Figure 31: Circular motion simulation map frame xy-position "top-down" view, using more aggressive sliding adaptive covariance profiling smoothing constants.

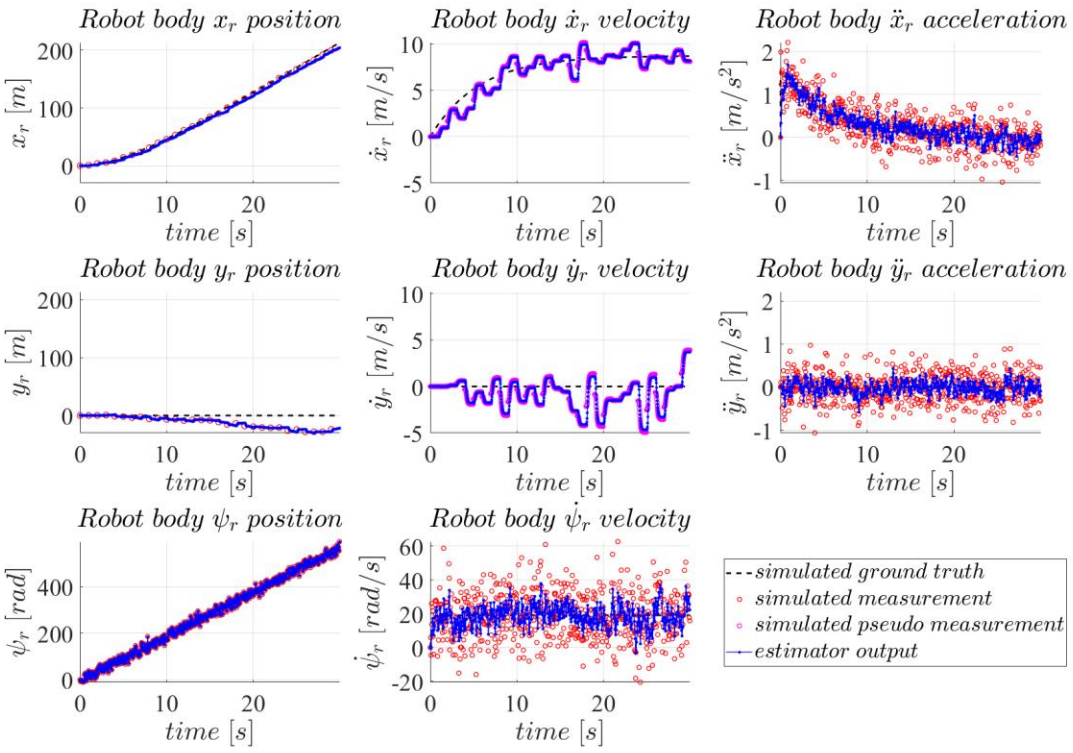

Figure 32: Circular motion simulation robot body frame states plotted versus time, using more aggressive sliding adaptive covariance profiling smoothing constants.

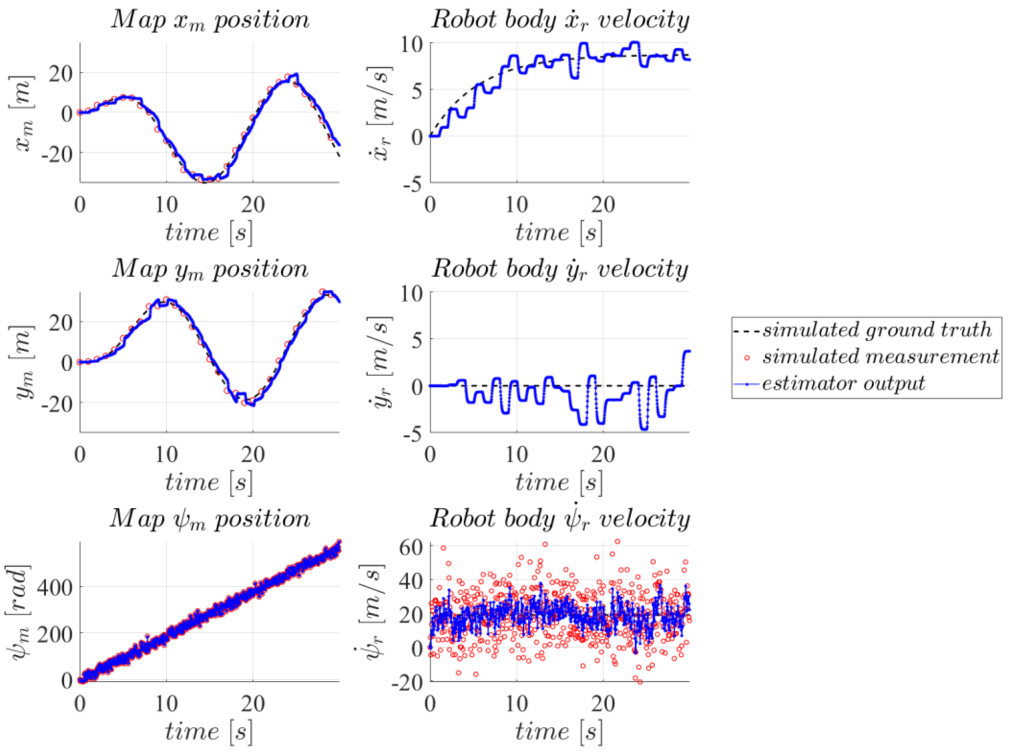

Figure 33: Circular motion simulation hybrid frame states plotted versus time, using more aggressive sliding adaptive covariance profiling smoothing constants.

Comparing the aggressive sliding adaptive covariance profiling in Figure 31, Figure 32, and Figure 33 to the binary adaptive covariance profiling in Figure 28, Figure 29, and Figure 30, the discrete jumps between GNSS measurements are much more smooth, leading to more “rounded” transitions between updates. There is not a necessarily “correct” answer regarding sliding adaptive covariance smoothing. Increasing the smoothing factor results in a more smoothed output but comes at the cost of latency in the estimator response, proportionally equal to the t_{smooth} time employed. Similar to our discussion of the number of elements n in the forward-difference calculation in section 4.1.1, the decrease in latency in the linear velocity pseudo-measurements also leads to a small increase in steady-state error; this can be seen in Figure 28, Figure 29, and Figure 30. In testing, I found that the default sliding adaptive covariance profiling variables in equation (26) demonstrated a good balance between smoothing, latency, and steady-state error; these are the values used for the remainder of this work.

Finally, also note the drift error in the robot frame $\dot{y}_r$ linear position state in Figure 26 (middle left). This occurs because robot frame linear positions are not directly measured, they are calculated by transforming the GNSS map frame linear position measurements based on the measured angular position ψ of the robot at GNSS discrete time $k-1$. So although the error in the GNSS map frame linear position measurements are bounded, the robot frame linear position measurements are not; they will drift due to systematic errors in either the map frame GNSS linear position data, or the IMU angular position data. Because this circular motion simulation only rotated left, the robot frame linear position error drifted in the negative sway direction. Although an unbounded error in a linear position state sounds like a significant shortcoming in a state estimation algorithm, this information is only used to calculate the forward-difference for a linear velocity pseudo-measurement, i.e. it is only needed in an incremental sense. If this were a typical Kalman filter state estimator employing frame transformations, transforming this drifted error back and forth between the map and robot frames would compound this error; however, the cascading, dual-frame nature of our state estimation algorithm prevents this error from being reintroduced back into the map frame (discussed in section 3.4).

4.1.4. Constant surge-sway-yaw

In this simulation, I impose a positive input force/torque on all three degrees-of-freedom on the robot frame, i.e. a surge, sway, and yaw input force. Combining all motion in all three-degrees-of-freedom provides the most challenging simulation for the estimator to predict motion, imperfect measurements introduce ambiguity in determining what motion belongs to what state variables. This is demonstrated in Figure 34, Figure 35, and Figure 36.

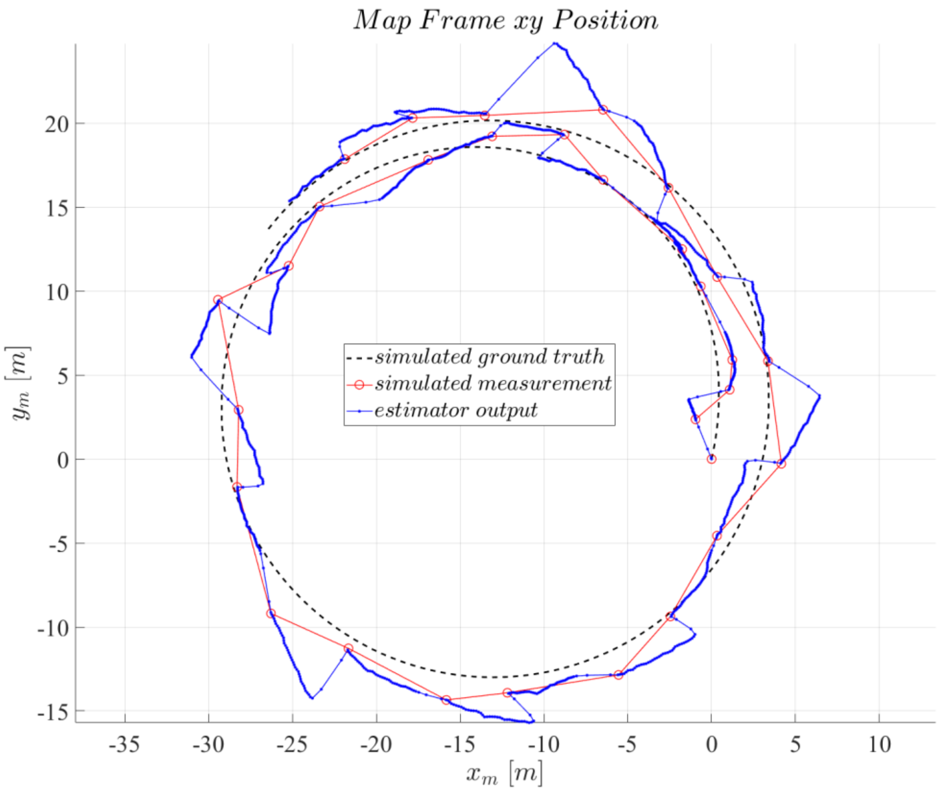

Figure 34: Constant surge-sway-yaw simulation map frame xy-position “top-down” view.

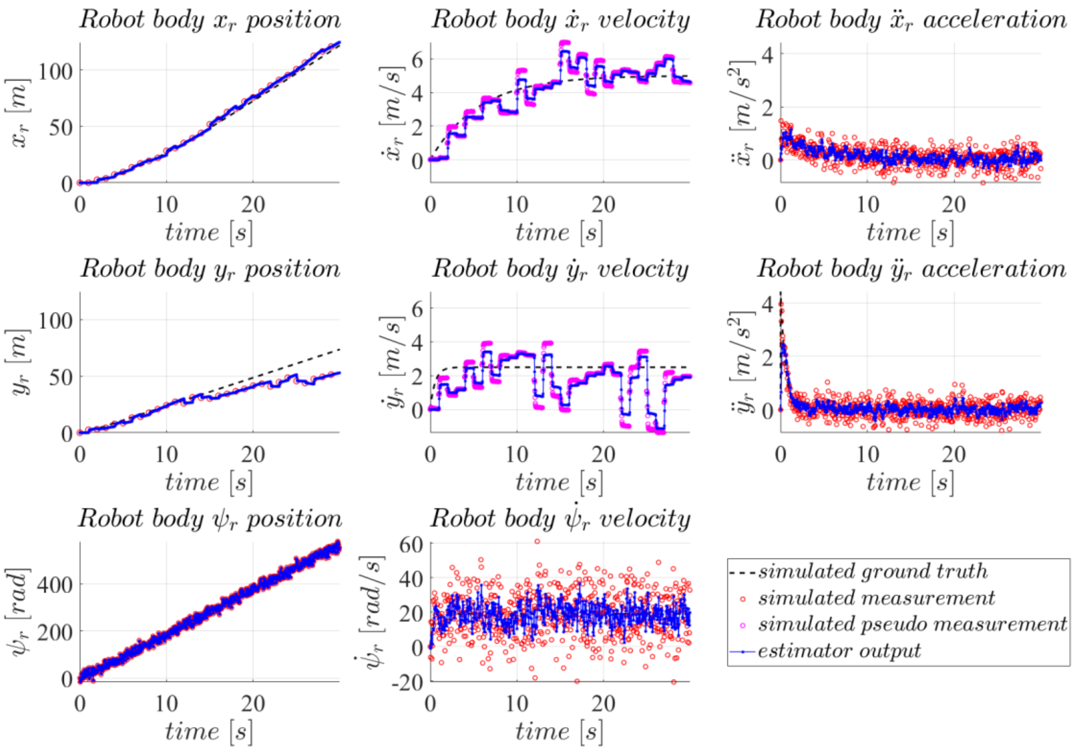

Figure 35: Constant surge-sway-yaw simulation robot body frame states plotted versus time.

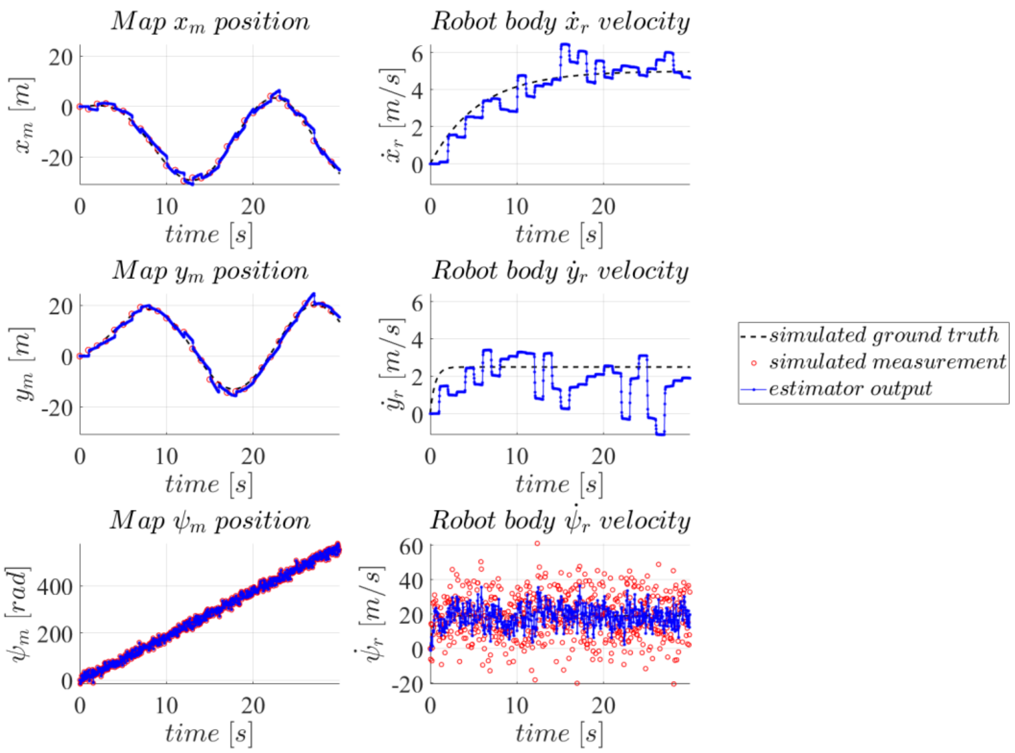

Figure 36: Constant surge-sway-yaw simulation hybrid frame states plotted versus time.

From Figure 34, the system had a harder time predicting the motion of the vehicle when compared to Figure 31. In particular, note how the y_r robot frame linear position drift error Figure 35 (middle left) manifests as a steady-state error in its respective $\dot{y}_r$ linear velocity (middle middle). This can be shown to be true by running the same simulation, but with an input torque in the opposite direction (resulting in a clockwise rotation). This is shown in Figure 37, Figure 38, and Figure 39.

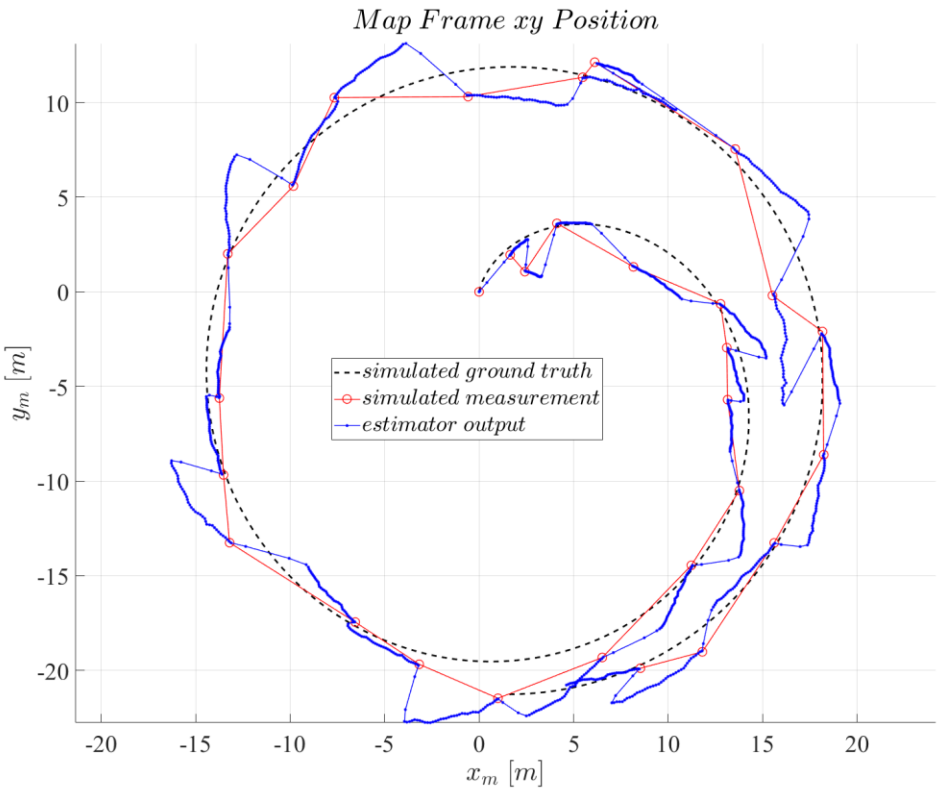

Figure 37: Constant surge-sway-negative-yaw simulation, map frame xy-position "top-down" view.

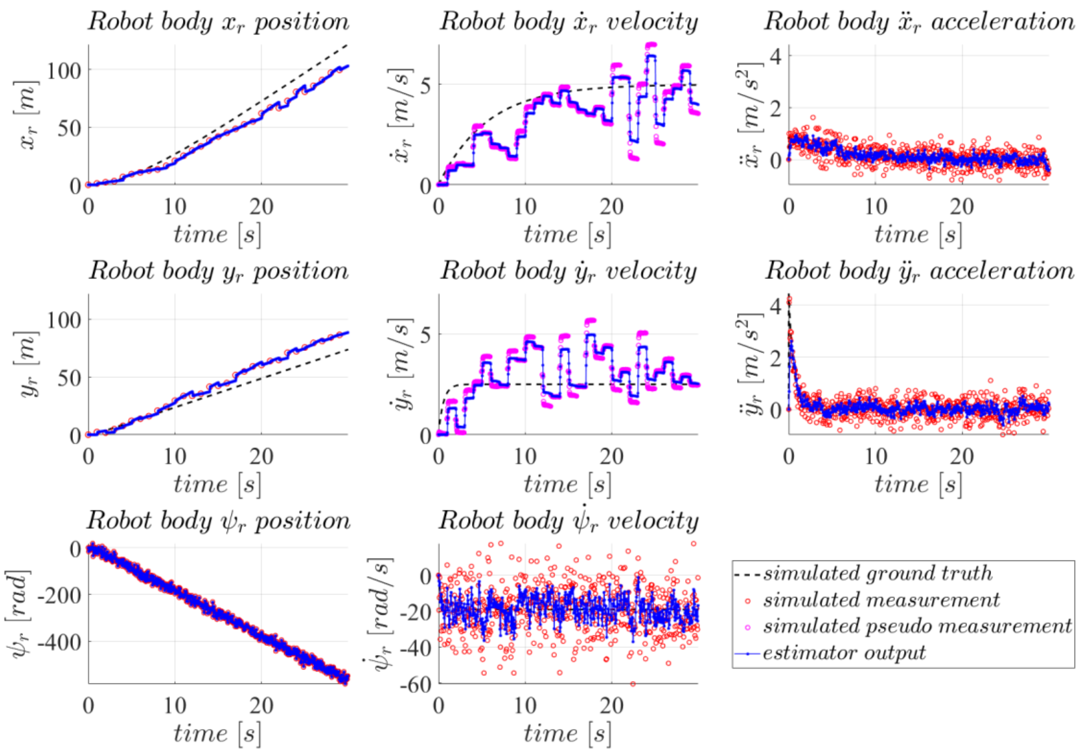

Figure 38: Constant surge-sway-negative-yaw simulation robot body frame states plotted versus time.

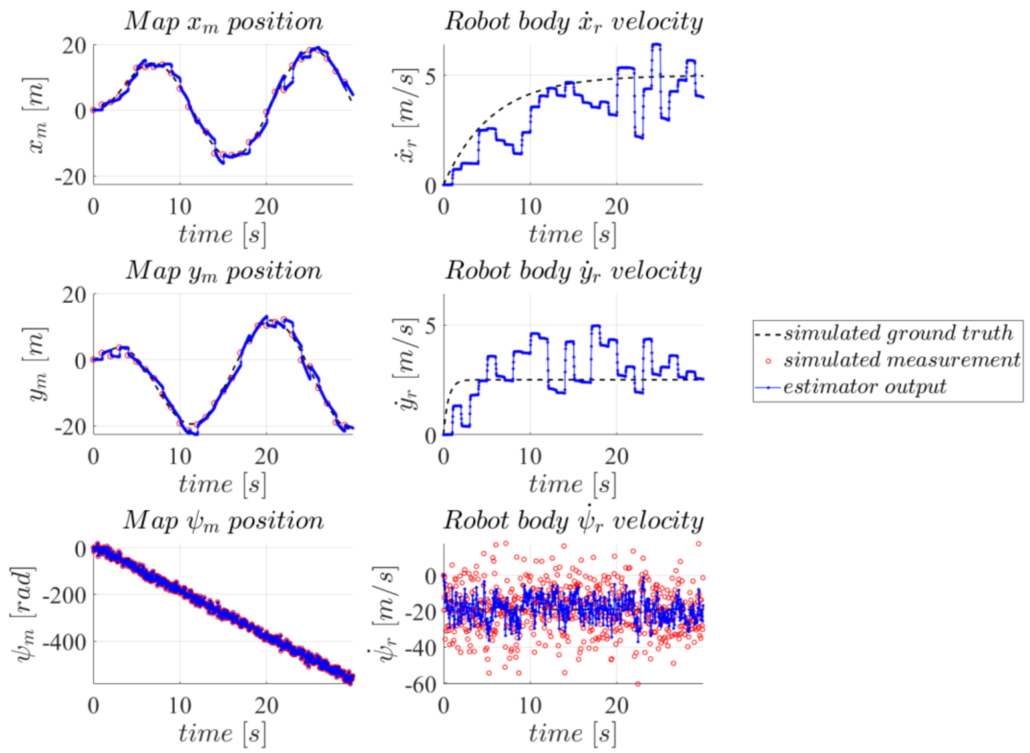

Figure 39: Constant surge-sway-negative-yaw simulation hybrid frame states plotted versus time.

Figure 38 demonstrates that the clockwise rotation causes the x_r surge linear position to drift negatively, and the y_r sway linear position state to drift positively. This results in an underestimating steady state error in the $\dot{x}_r$ linear velocity state, and an overestimating steady state error in the $\dot{y}_r$ linear velocity state, as expected.

4.1.5. Alternating surge-sway-yaw

In this simulation, I impose an input force/torque on all three degrees-of-freedom on the robot frame and alternate the polarities of the inputs every 15 s. This puts the estimator under a similarly taxing challenge as the previous simulation, while adding the effects of transient robot dynamics. This is shown in Figure 40, Figure 41, and Figure 42.

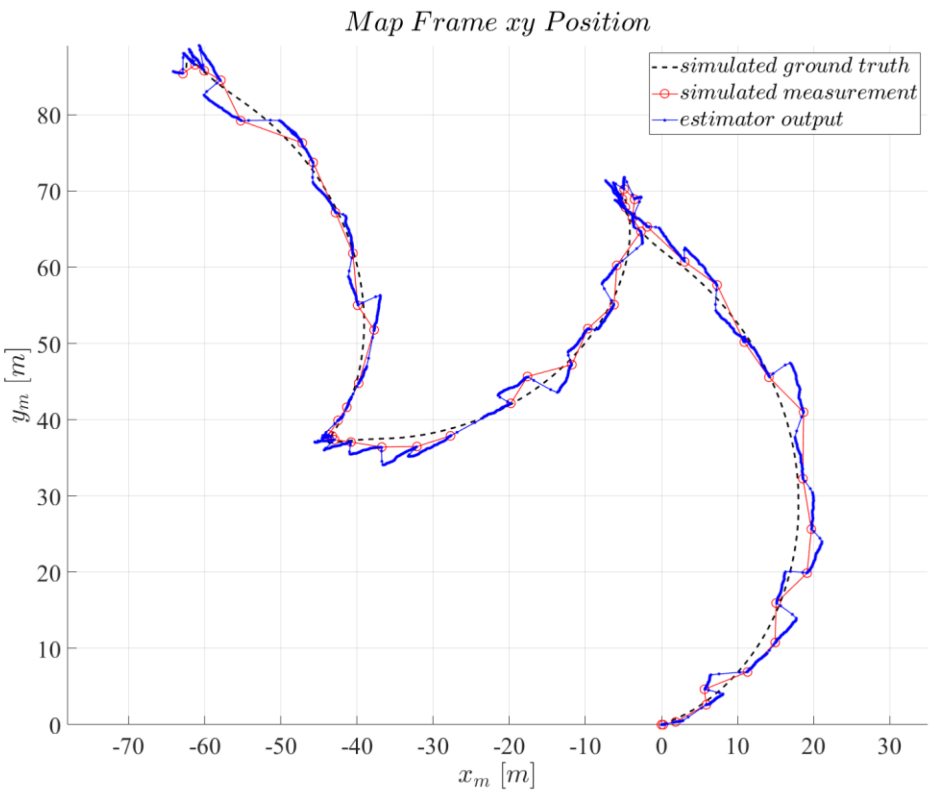

Figure 40: Alternating surge, sway, yaw simulation, map frame xy-position “top-down” view.

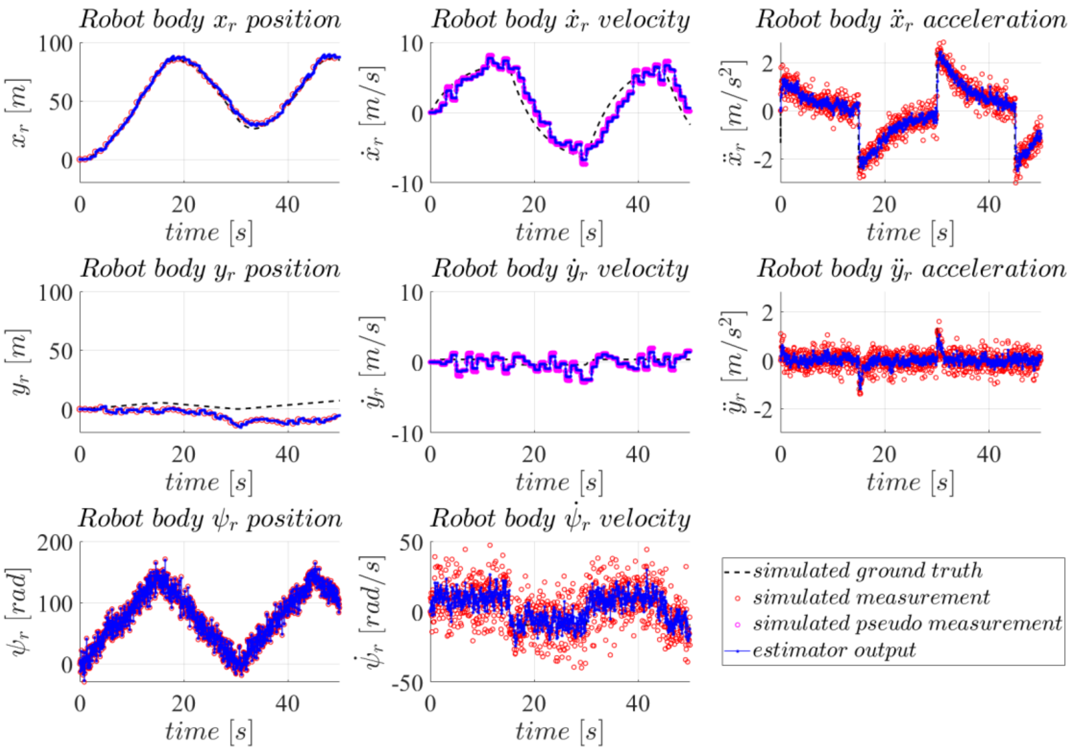

Figure 41: Alternating surge, sway, yaw simulation robot body frame states plotted versus time.

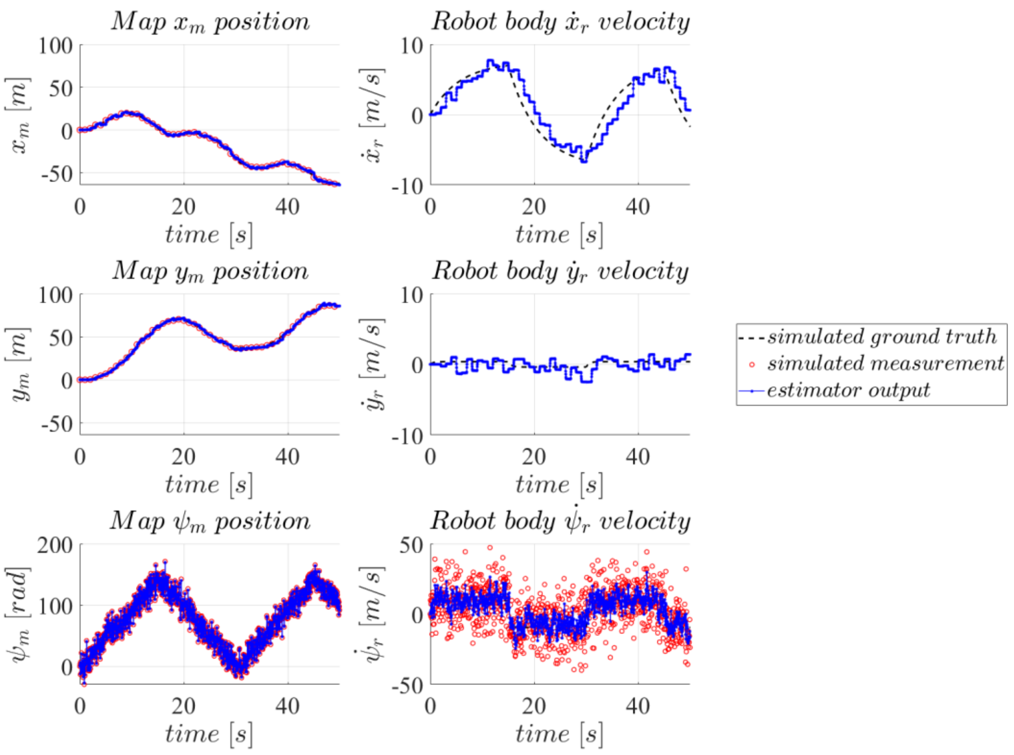

Figure 42: Alternating surge, sway, yaw simulation hybrid frame states plotted versus time.

In Figure 40, the transient effects of the changing forcing inputs highlight how the latency in the pseudo-linear velocity measurements results in the estimator taking some time to generate reasonably accurate pose predictions after a big change in the robot dynamics.

4.2. Estimator performance on real data

4.2.1. Experimental setup

In this section, I describe the performance of our state estimation system on data collected from real sensor systems. I used the Adafruit Ultimate GPS Version 3 Breakout [88], and the Redshift Labs UM7-LT Orientation Sensor [95]. These two sensors are shown in Figure 43.

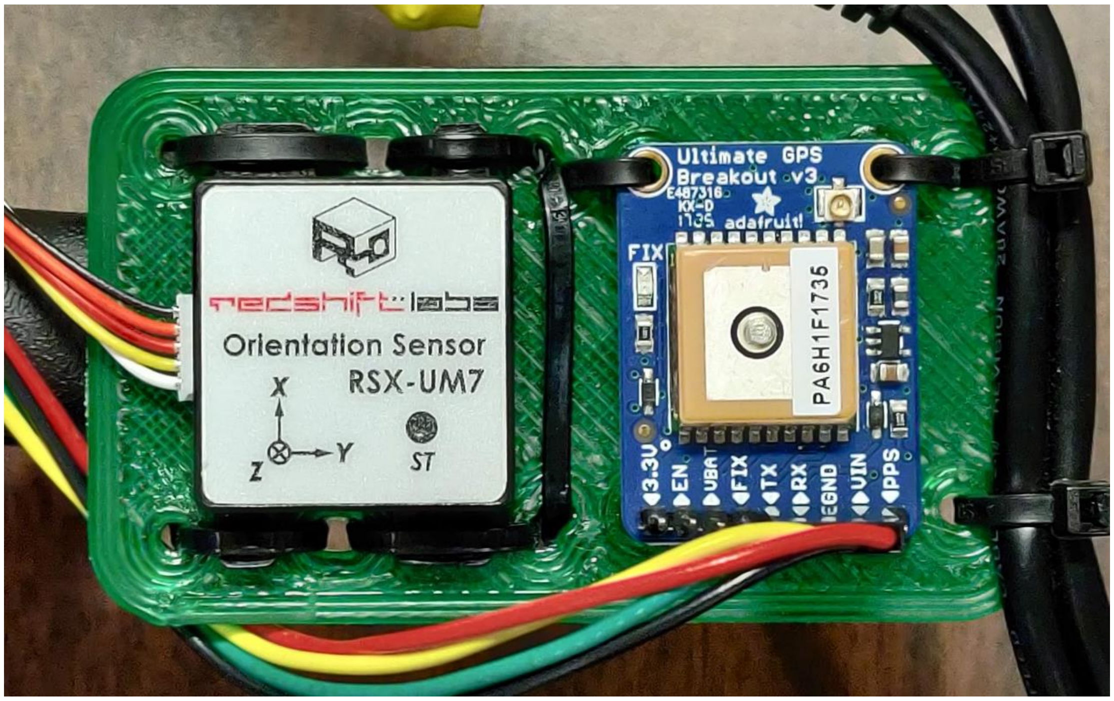

Figure 43: Redshift Labs UM7-LT sensor (left) and Adafruit Ultimate GPS Version 3 Breakout sensor (right).

These sensors are both connected via a FTDI FT242RL USB to serial breakout board to a computer running the Robot Operating System (ROS) [118]. To read data from the GNSS sensor, I use the ROS "nmea_navsat_driver" package [119], and to read data from the IMU, I use the ROS "um7" package [120]. The sensor data is collected in real time, and saved using the ROS "rosbag" package [121]; this allows a user to "record" sensor data in real time, and play it back at a later time as if those sensor systems were actively connected and outputting data. To visualize the data, in real time or in post-processing using rosbags, I use the ROS "plogjuggler" package [122].

4.2.2. Estimator performance on rectangular travel path

In this experiment, I test the improved state estimation algorithm on a "rectangular" travel path. By walking along a marked rectangular ground path with known dimensions, I can correlate the measurements and estimator outputs to a "ground truth" and calculate the accuracy of the measurements and the estimator output. The travel path is illustrated in Figure 44 (counter-clockwise) and Figure 45 (clockwise).

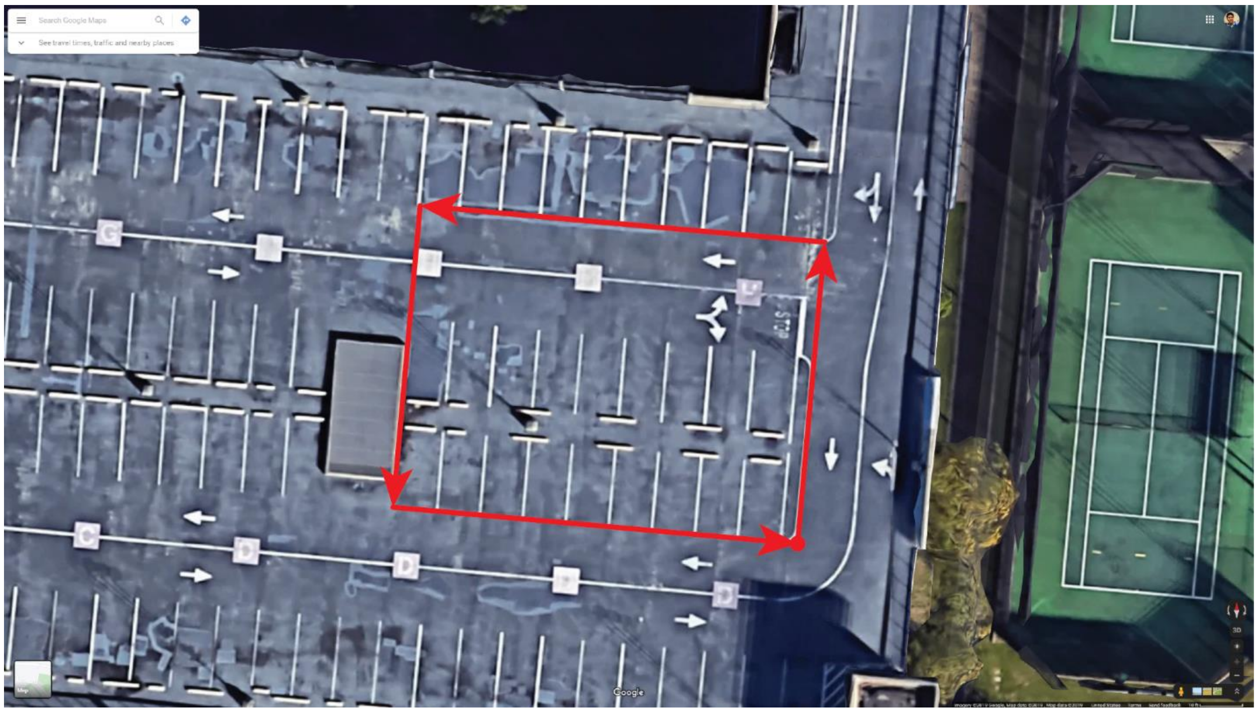
Figure 44: Counter-clockwise rectangular travel path. The long leg of travel is approximately 28 m, and the short leg of travel is approximately 22.5 m. Map data © 2018 Google. Reproduced with permission from Google Maps Geoguidelines, 2019 [123].

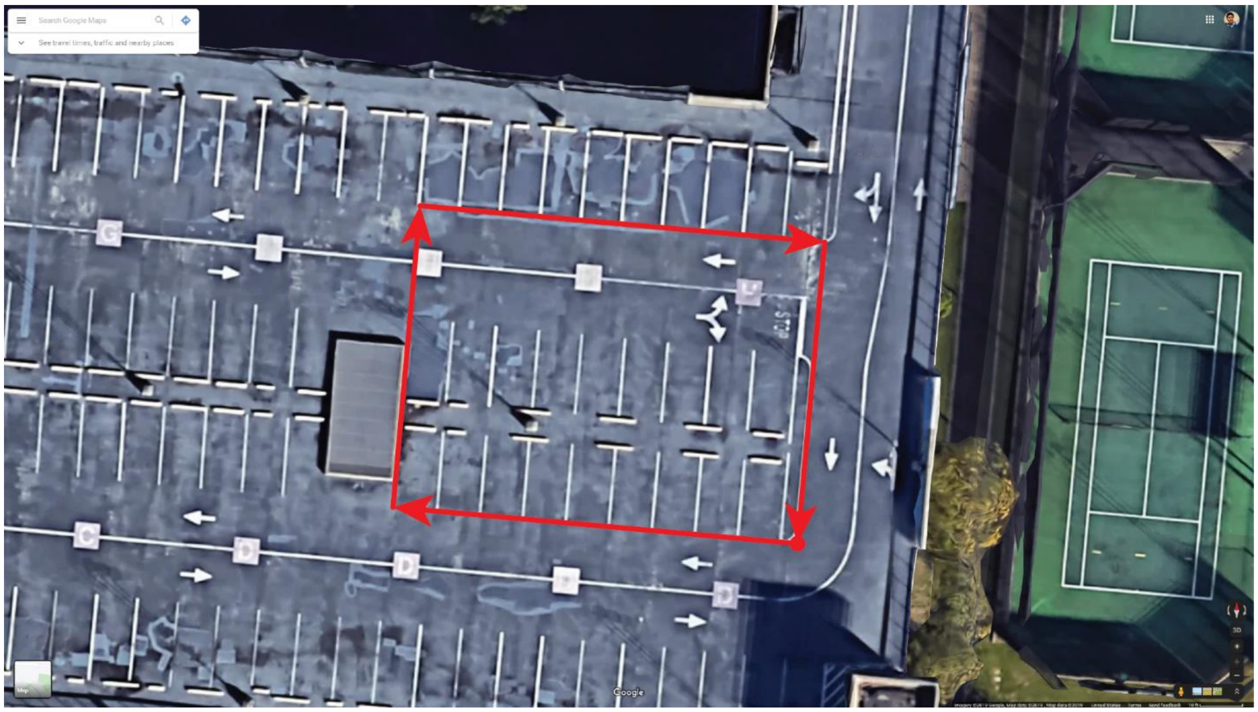
Figure 45: Counter-clockwise rectangular travel path. The long leg of travel is approximately 28 m, and the short leg of travel is approximately 22.5 m. Map data © 2018 Google. Reproduced with permission from Google Maps Geoguidelines, 2019 [123].

I collected data from three trials moving at a slow velocity ($\approx 1.5\ to\ 1.7\ [m/s]$), three trials at a faster velocity ($\approx 3.5\ to\ 3.7\ [m/s]$), for the counter-clockwise and clockwise directions (for a total of 12 different data collections).

The counter-clockwise "slow" ($\bar{v} \approx 1.3\ [m/s]$) trials one, two and three are shown in Figure 46 through Figure 54. The counter-clockwise "fast" ($\bar{v} \approx 2.8\ to\ 3.1\ [m/s]$) trials one, two, and three are shown in Figure 55 through Figure 63. The clockwise "slow" ($\bar{v} \approx 1.3\ to\ 1.5\ [m/s]$) trials one, two, and three are shown in Figure 64 through Figure 72. The clockwise "fast" ($\bar{v} \approx 2.8\ to\ 3.1\ [m/s]$) trials one, two, and three are shown in Figure 73 through Figure 81.

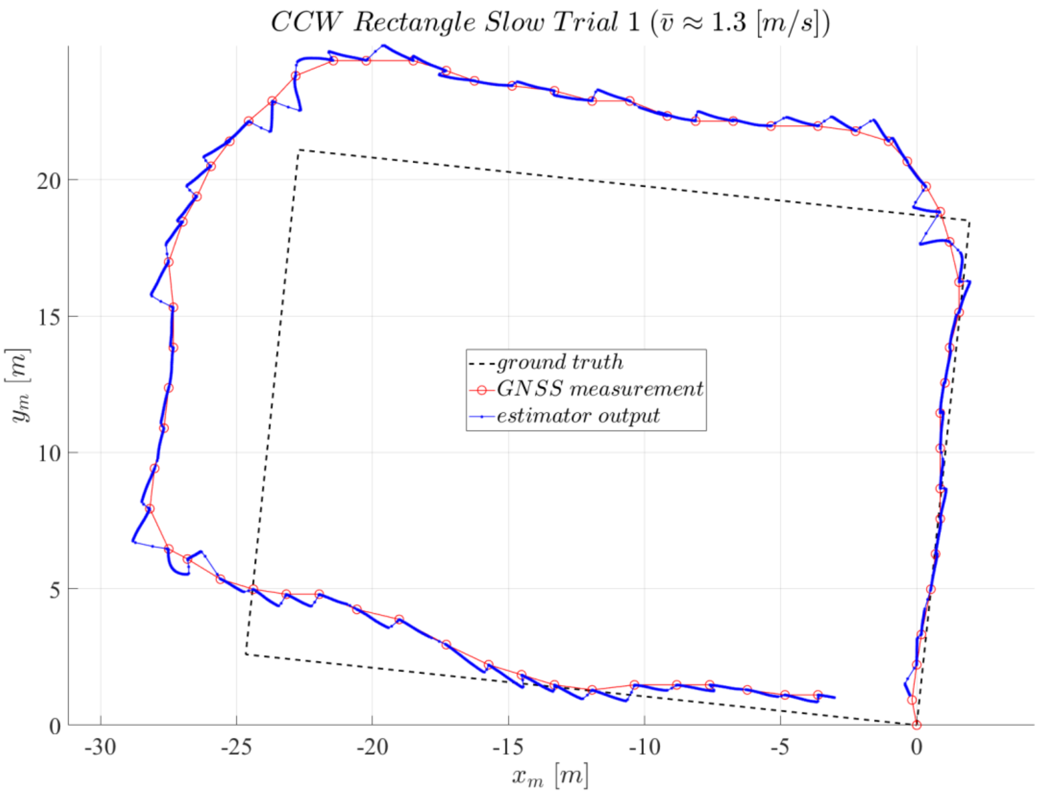

Figure 46: "Top-down" (x_m versus y_m) counter-clockwise rectangle slow trial 1. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx 1.3\ [m/s]$.

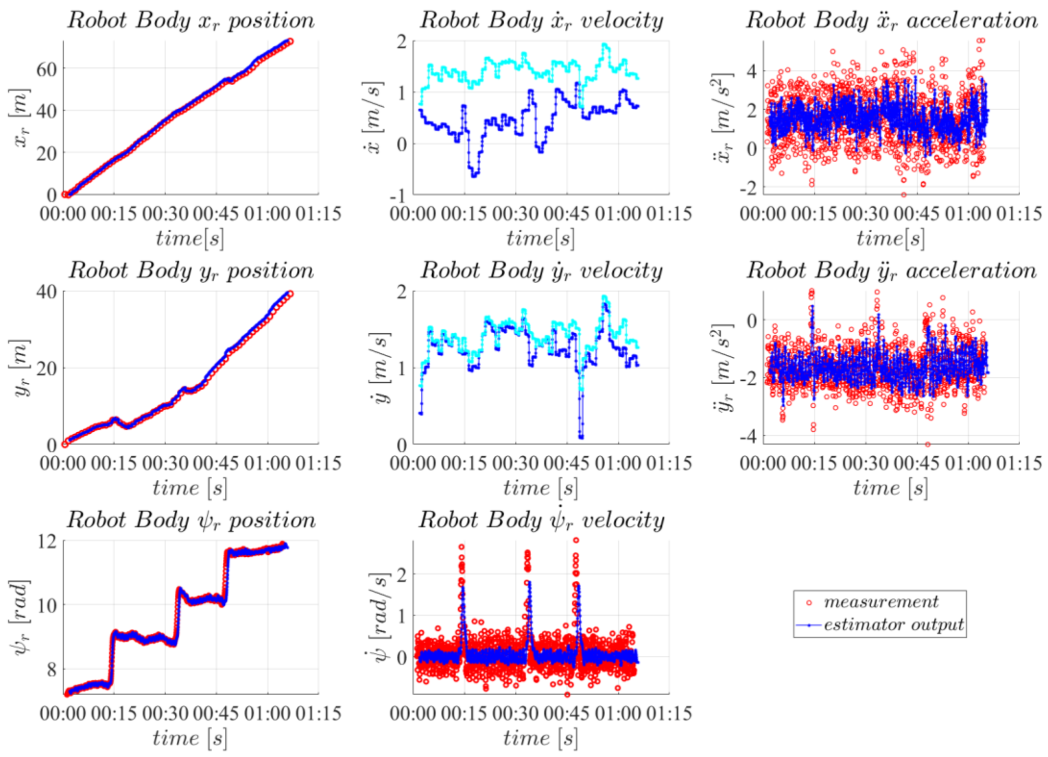

Figure 47: Robot body frame states plotted versus time for the counter-clockwise rectangle slow trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.3 $[m/s]$.

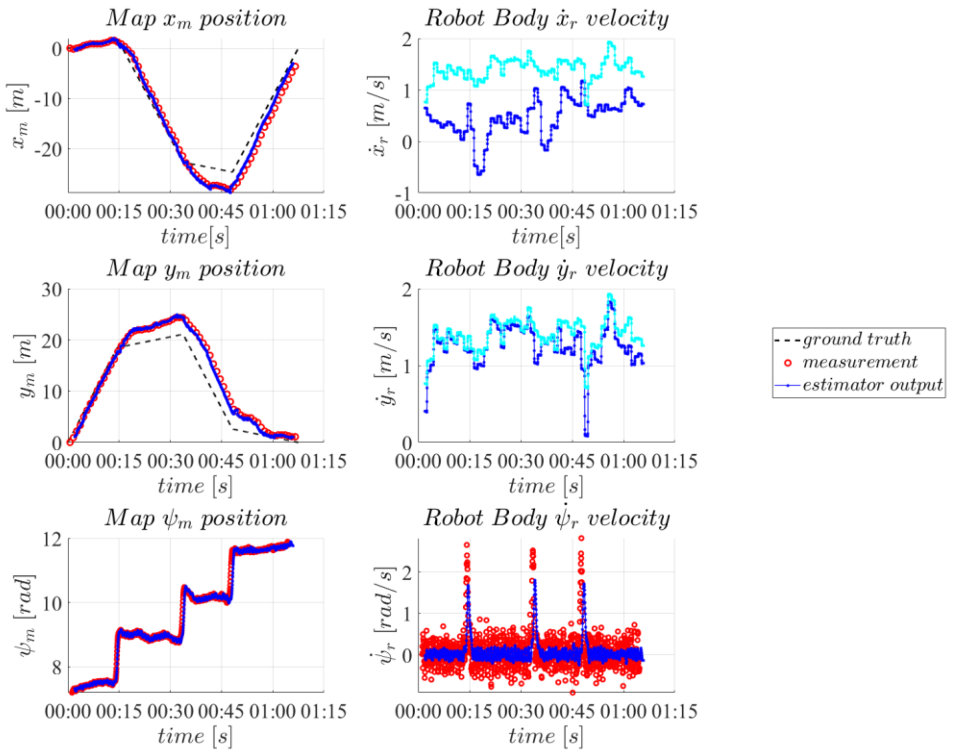

Figure 48: Hybrid frame states plotted versus time for the counter-clockwise rectangle slow trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.3 $[m/s]$.

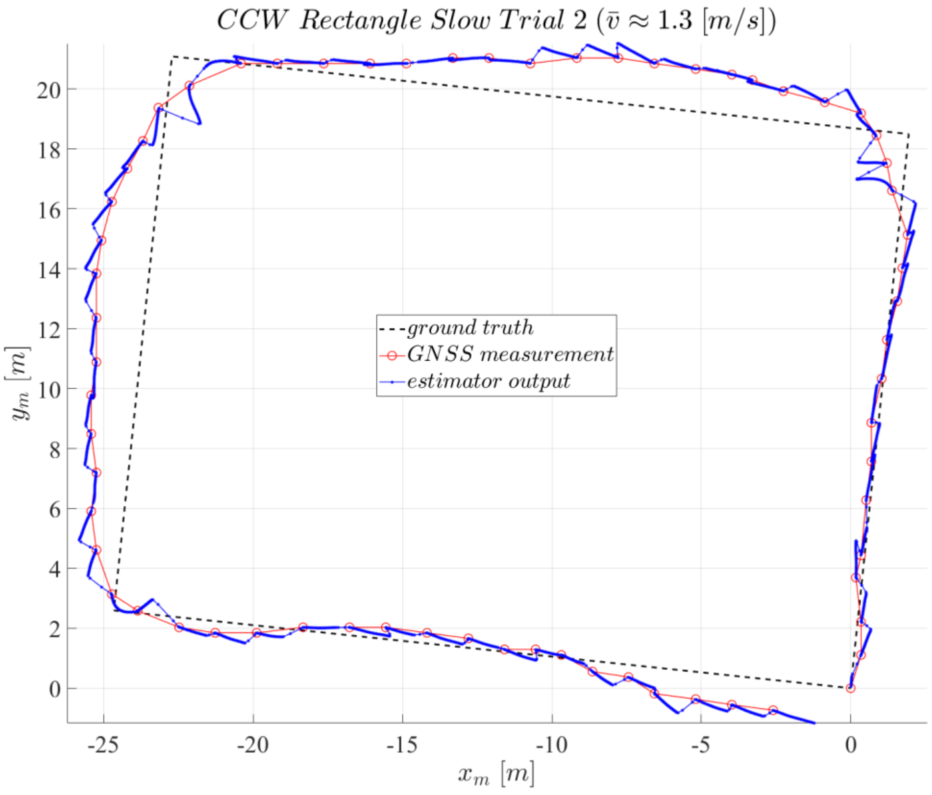

Figure 49: "Top-down" (x_m versus y_m) counter-clockwise rectangle slow trial 2. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx 1.3\ [m/s]$.

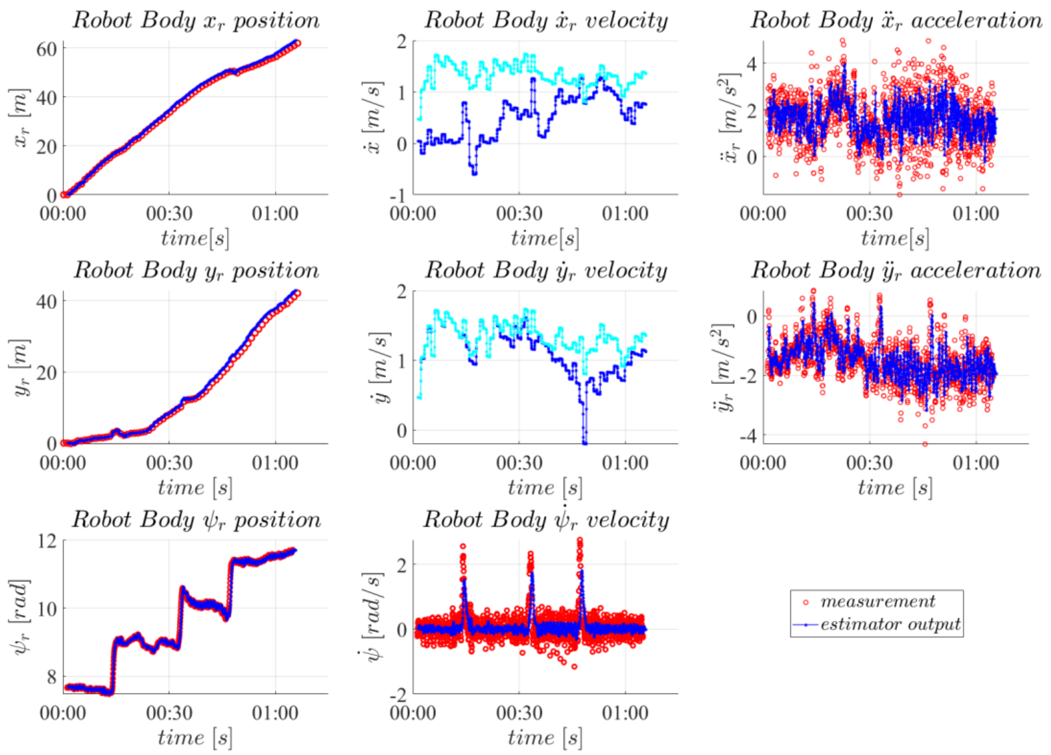

Figure 50: Robot body frame states plotted versus time for the counter-clockwise rectangle slow trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.3 $[m/s]$.

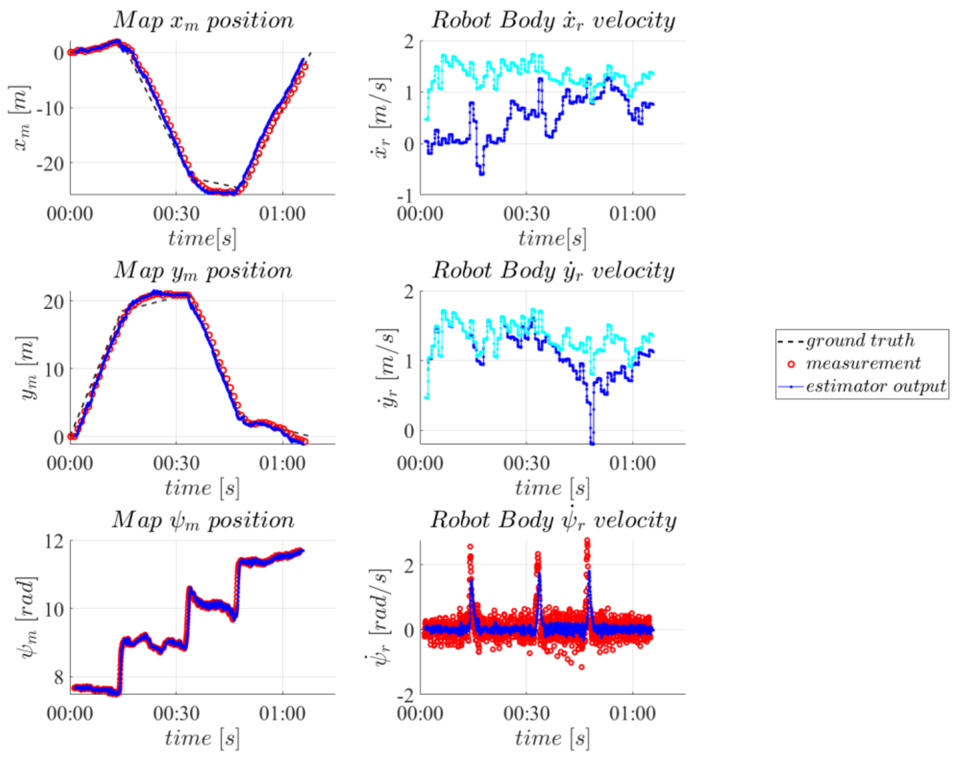

Figure 51: Hybrid frame states plotted versus time for the counter-clockwise rectangle slow trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.3 $[m/s]$.

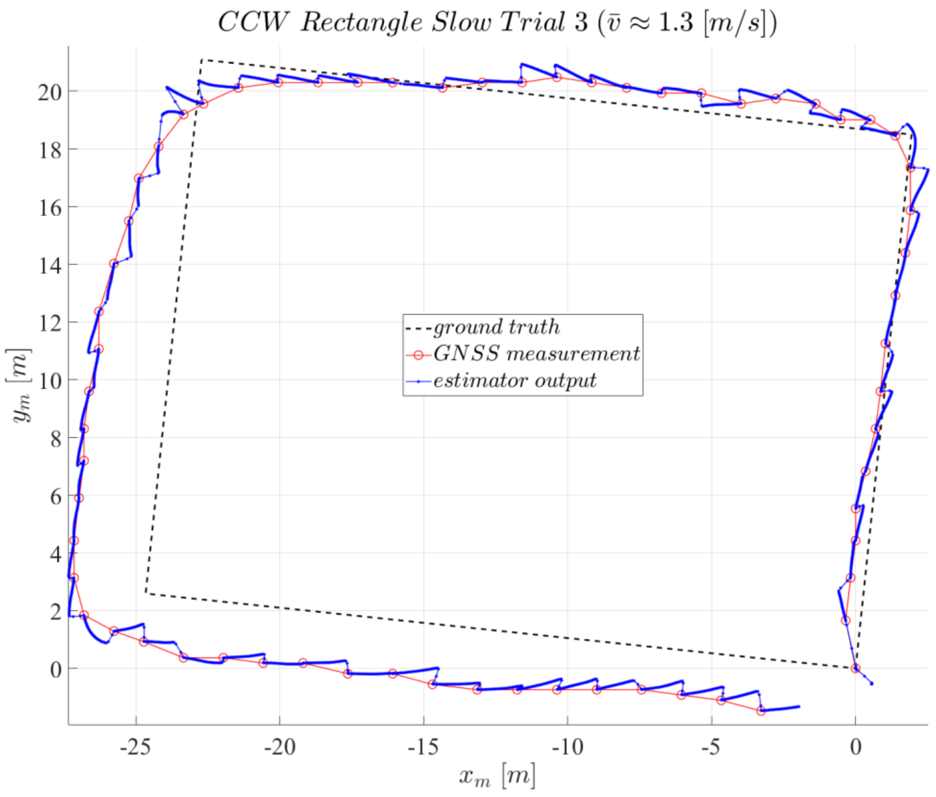

Figure 52: "Top-down" (x_m versus y_m) counter-clockwise rectangle slow trial 3. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx$ 1.3 $[m/s]$.

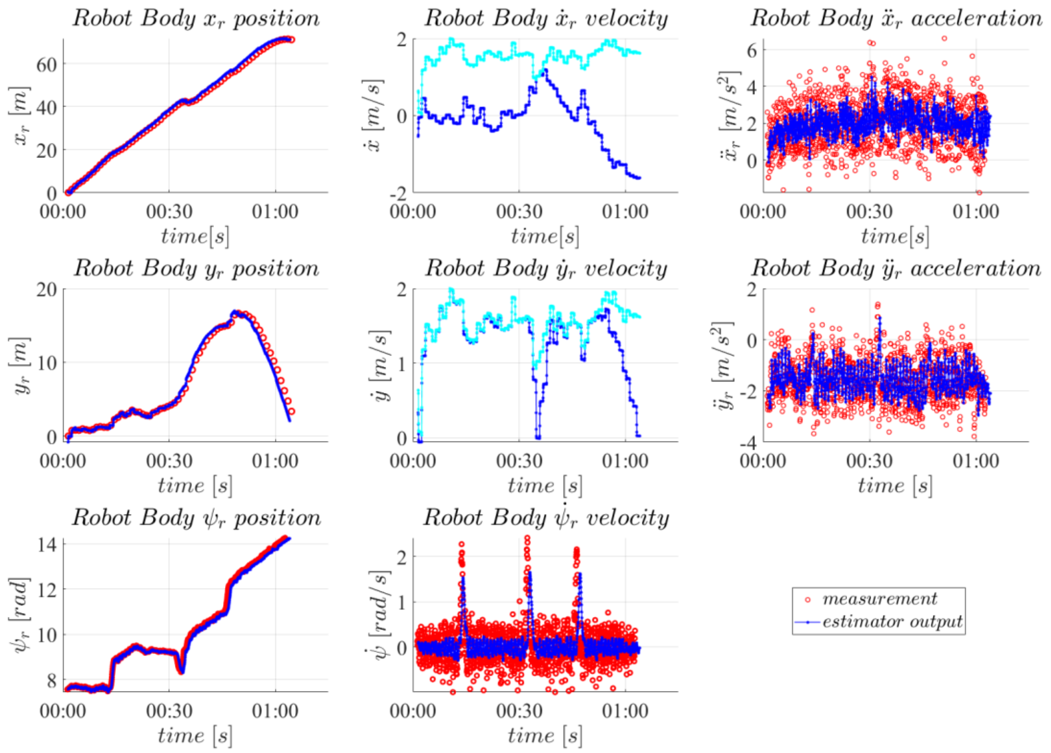

Figure 53: Robot body frame states plotted versus time for the counter-clockwise rectangle slow trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.3 $[m/s]$.

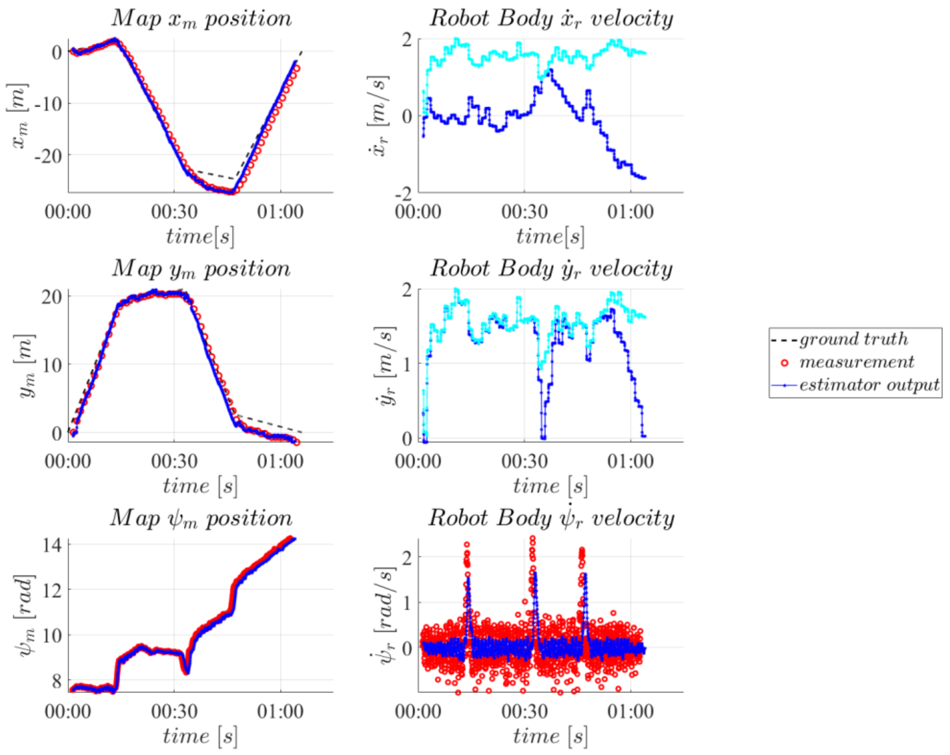

Figure 54: Hybrid frame states plotted versus time for the counter-clockwise rectangle slow trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.3 $[m/s]$.

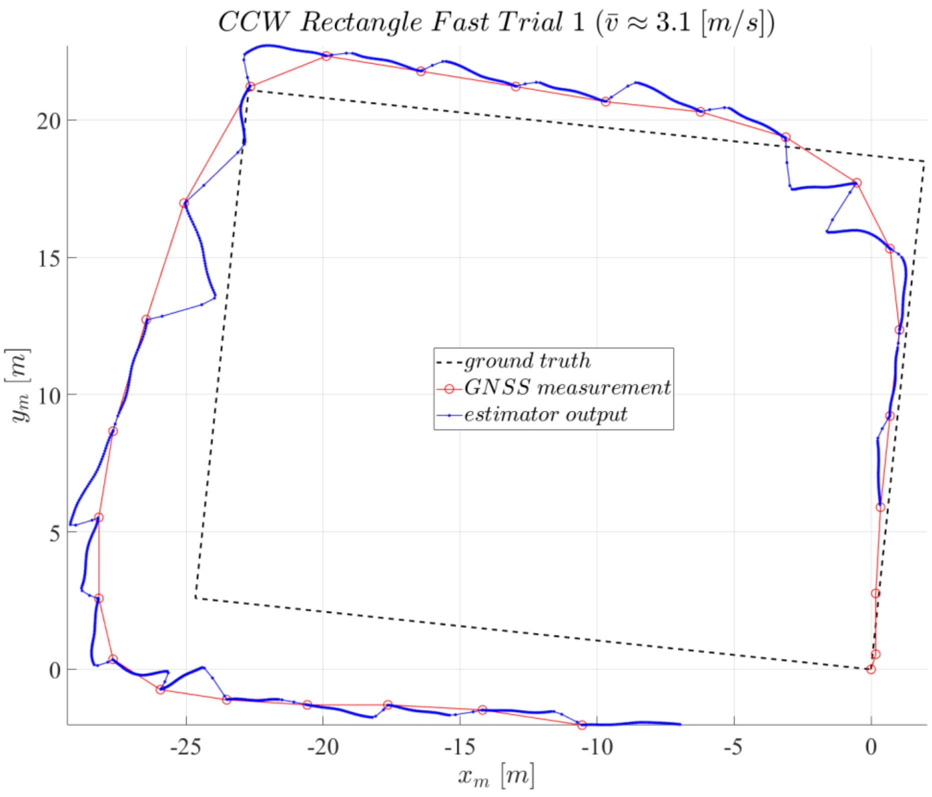

Figure 55: "Top-down" (x_m versus y_m) counter-clockwise rectangle fast trial 1. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx 3.1\ [m/s]$.

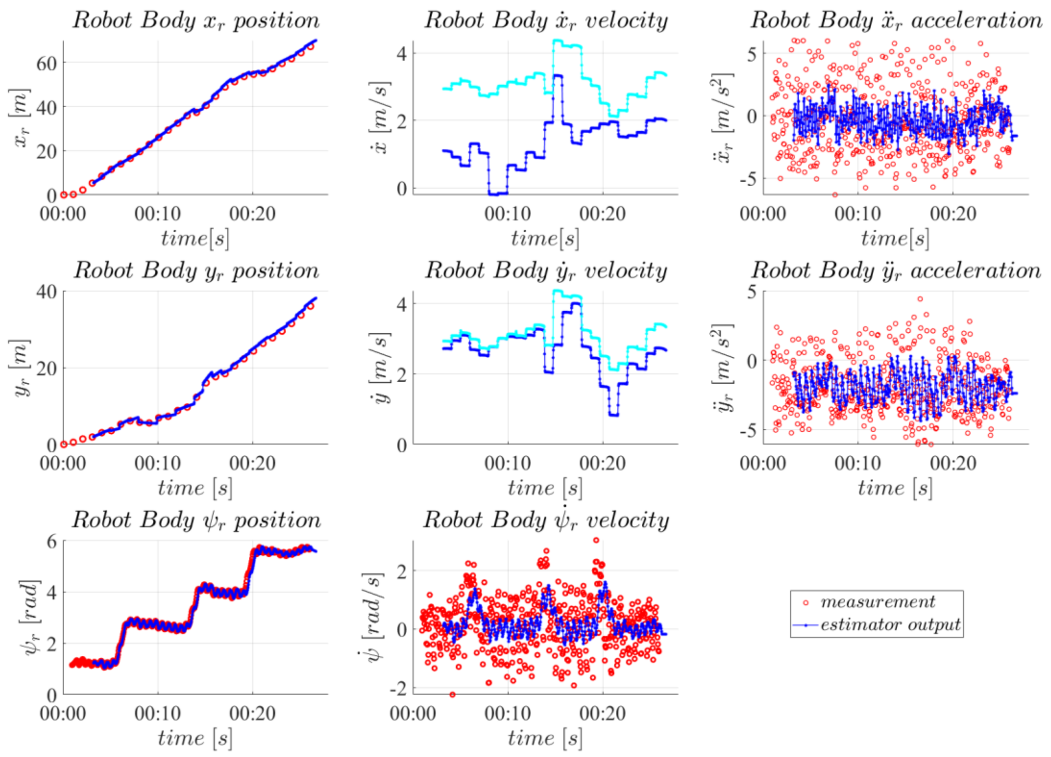

Figure 56: Robot body frame states plotted versus time for the counter-clockwise rectangle fast trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 3.1 $[m/s]$.

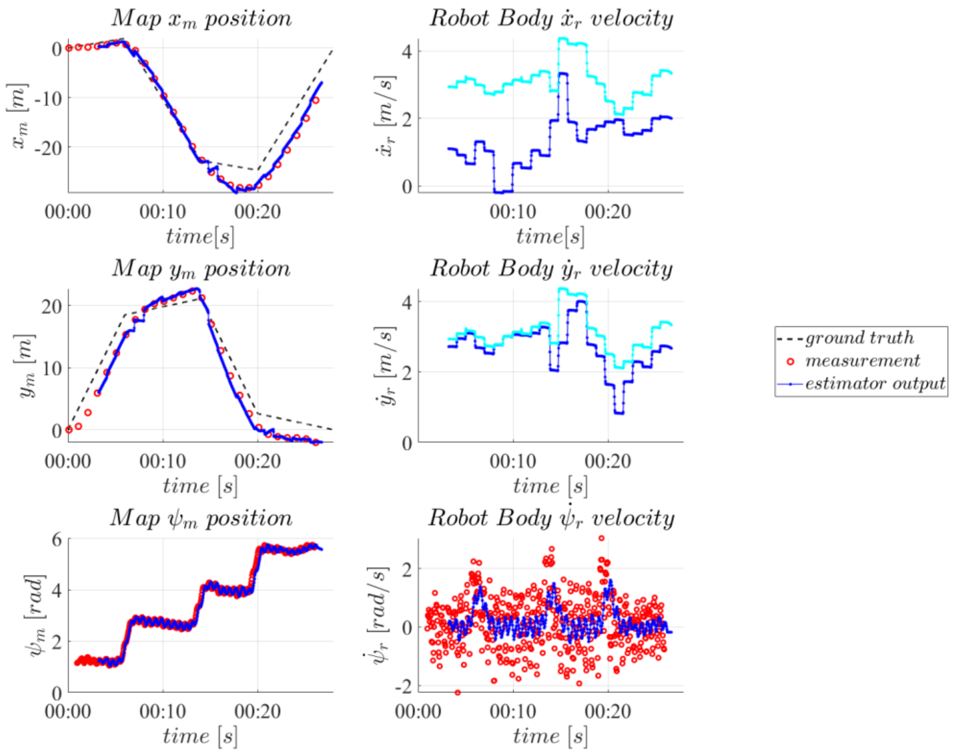

Figure 57: Hybrid frame states plotted versus time for the counter-clockwise rectangle fast trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 3.1 $[m/s]$.

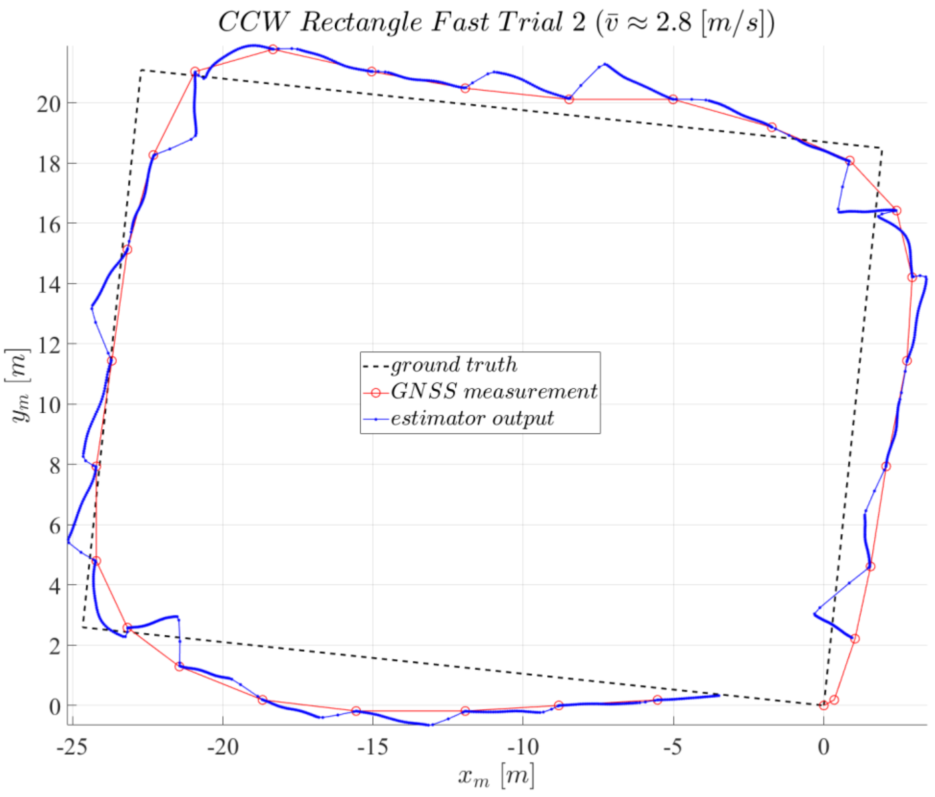

Figure 58: "Top-down" (x_m versus y_m) counter-clockwise rectangle fast trial 2. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was ≈ 2.8 [m/s].

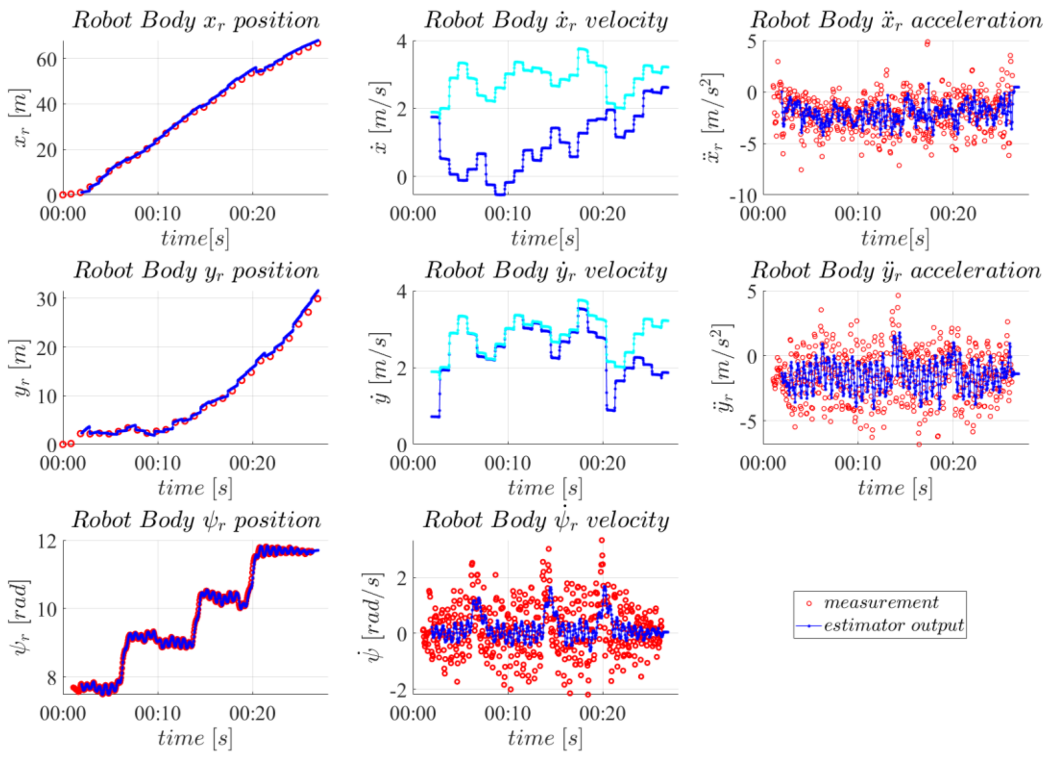

Figure 59: Robot body frame states plotted versus time for the counter-clockwise rectangle fast trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 2.8 $[m/s]$.

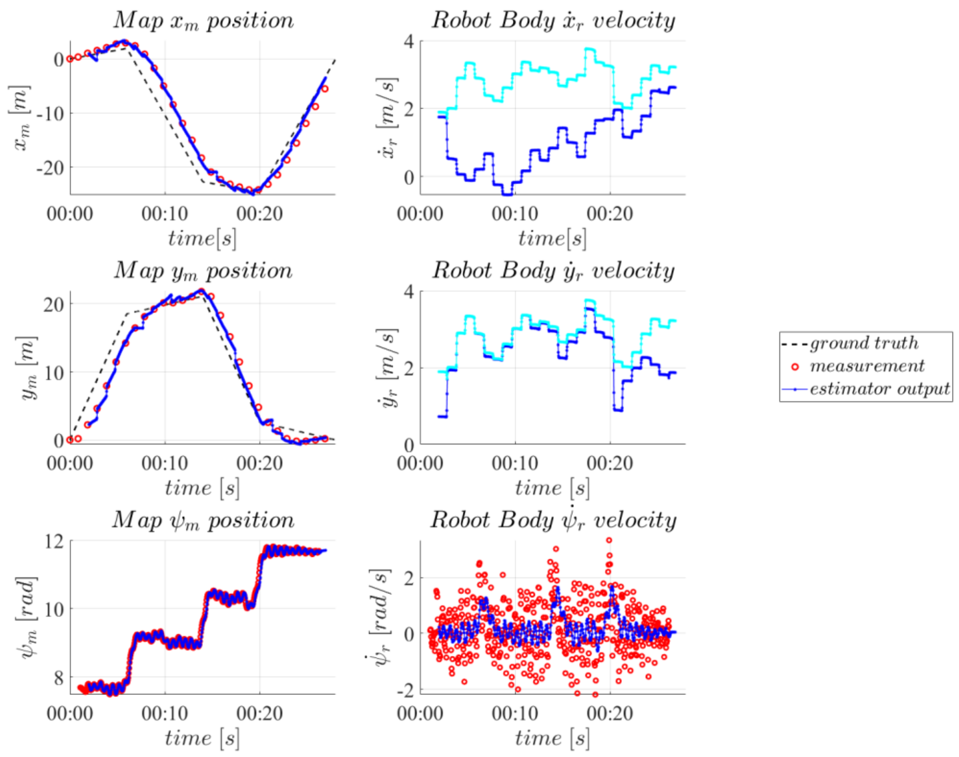

Figure 60: Hybrid frame states plotted versus time for the counter-clockwise rectangle fast trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 2.8 $[m/s]$.

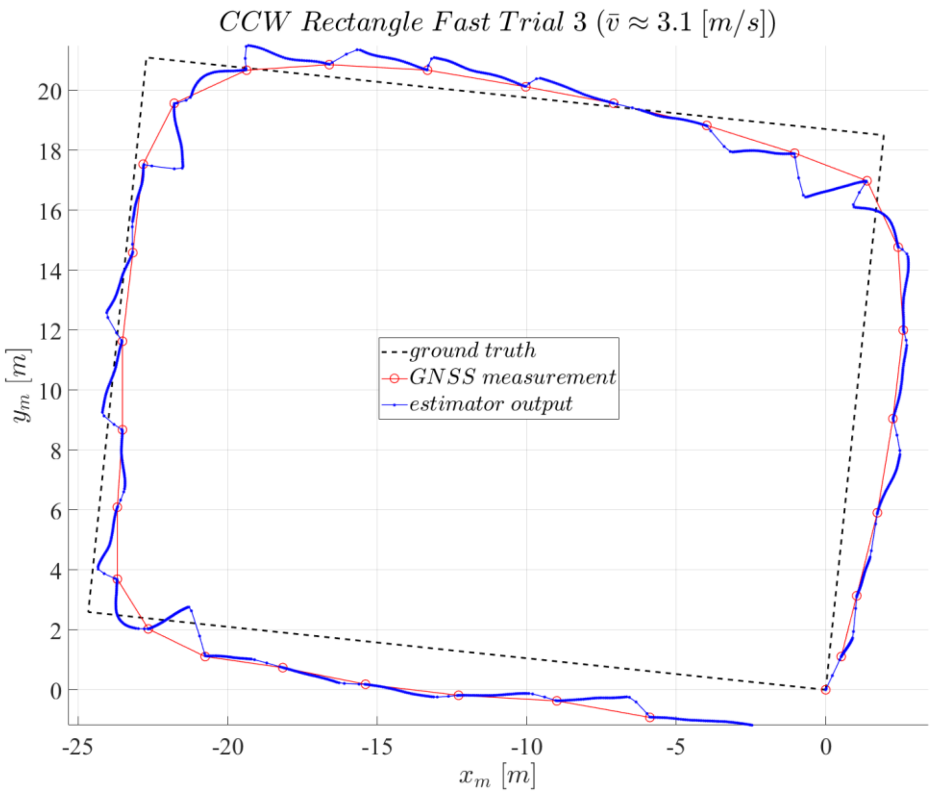

Figure 61: “Top-down” (x_m versus y_m) counter-clockwise rectangle fast trial 3. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx$ 3.1 $[m/s]$.

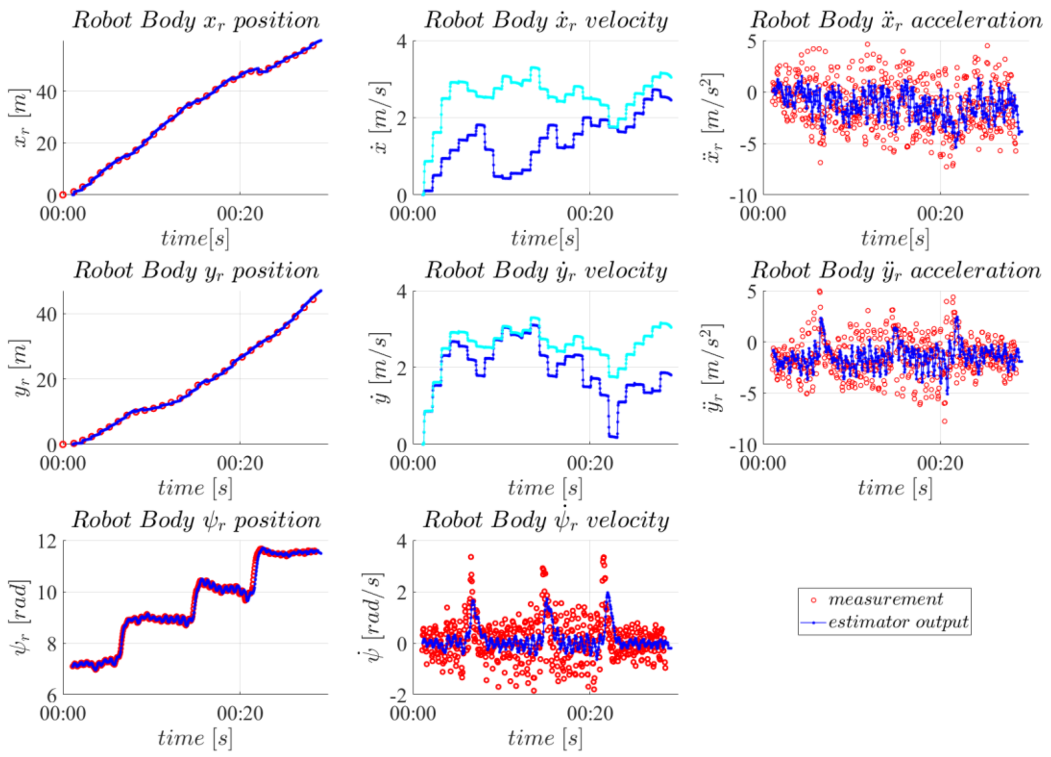

Figure 62: Robot body frame states plotted versus time for the counter-clockwise rectangle fast trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 3.1 $[m/s]$.

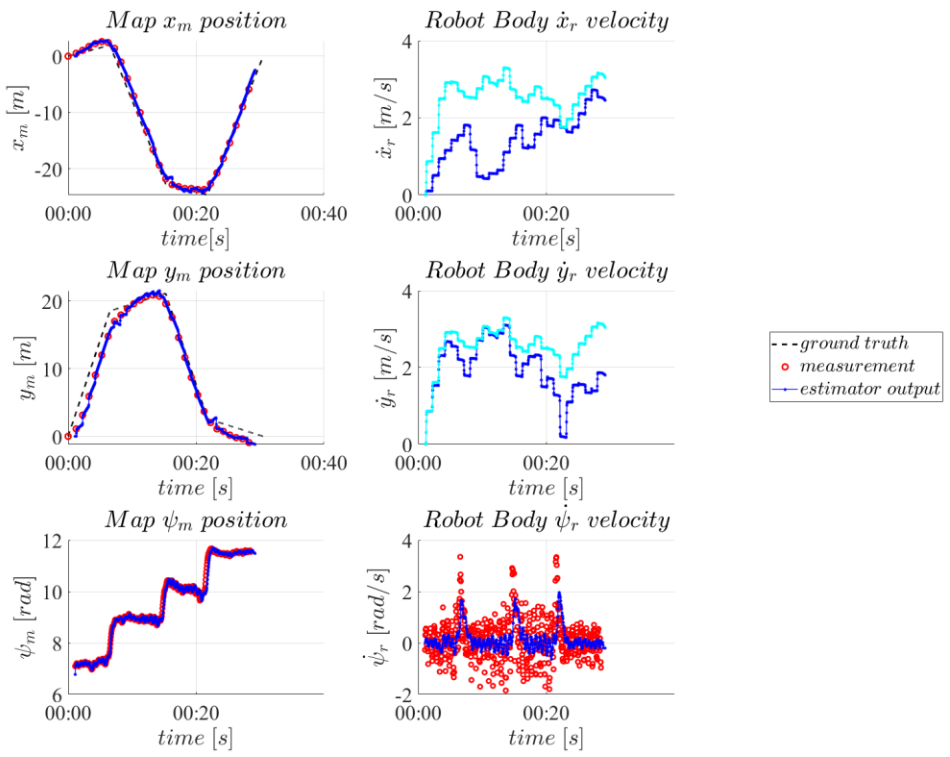

Figure 63: Hybrid frame states plotted versus time for the counter-clockwise rectangle fast trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 3.1 $[m/s]$.

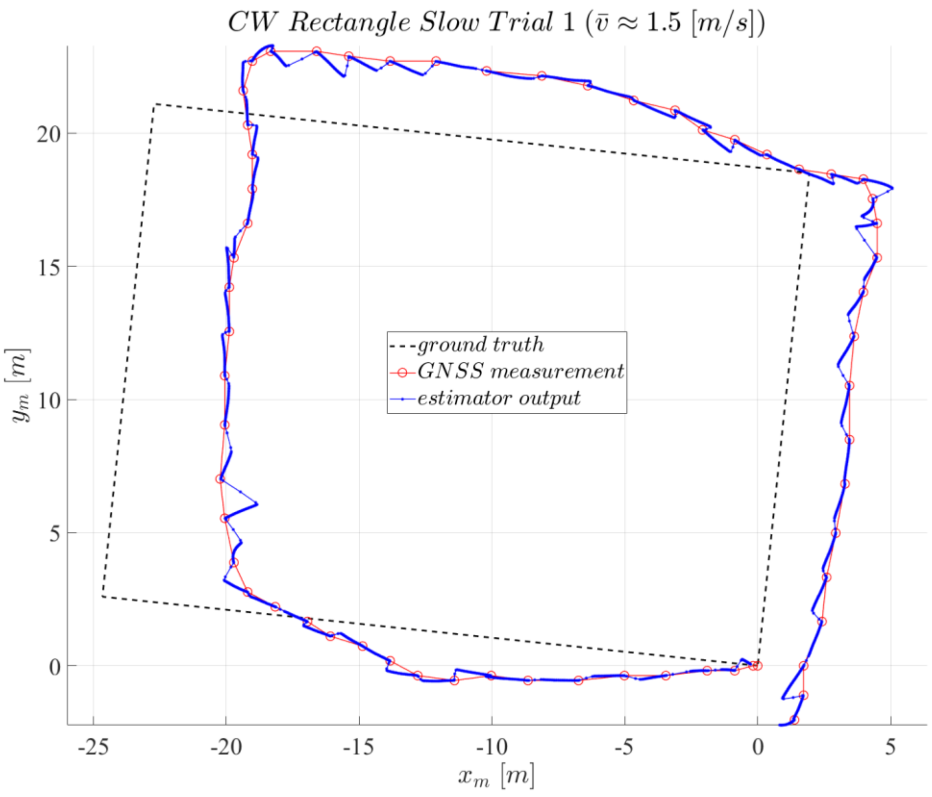

Figure 64: "Top-down" (x_m versus y_m) clockwise rectangle slow trial 1. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx 1.4\ [m/s]$.

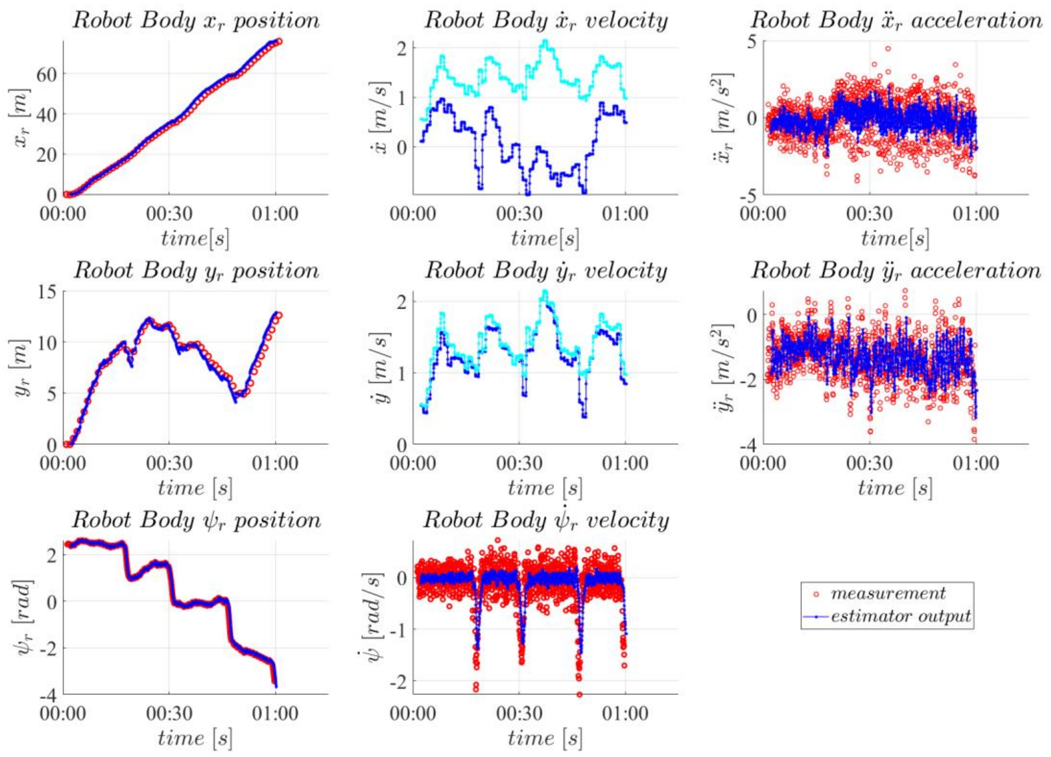

Figure 65: Robot body frame states plotted versus time for the clockwise rectangle slow trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.4 $[m/s]$.

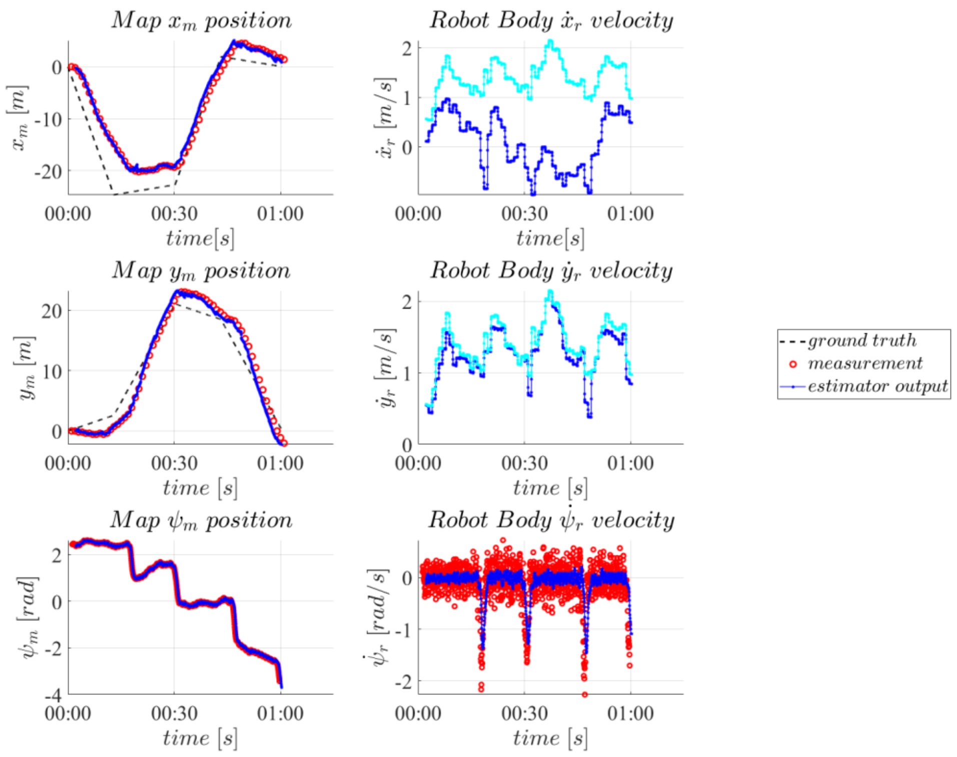

Figure 66: Hybrid frame states plotted versus time for the clockwise rectangle slow trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was $\approx 1.4\ [m/s]$.

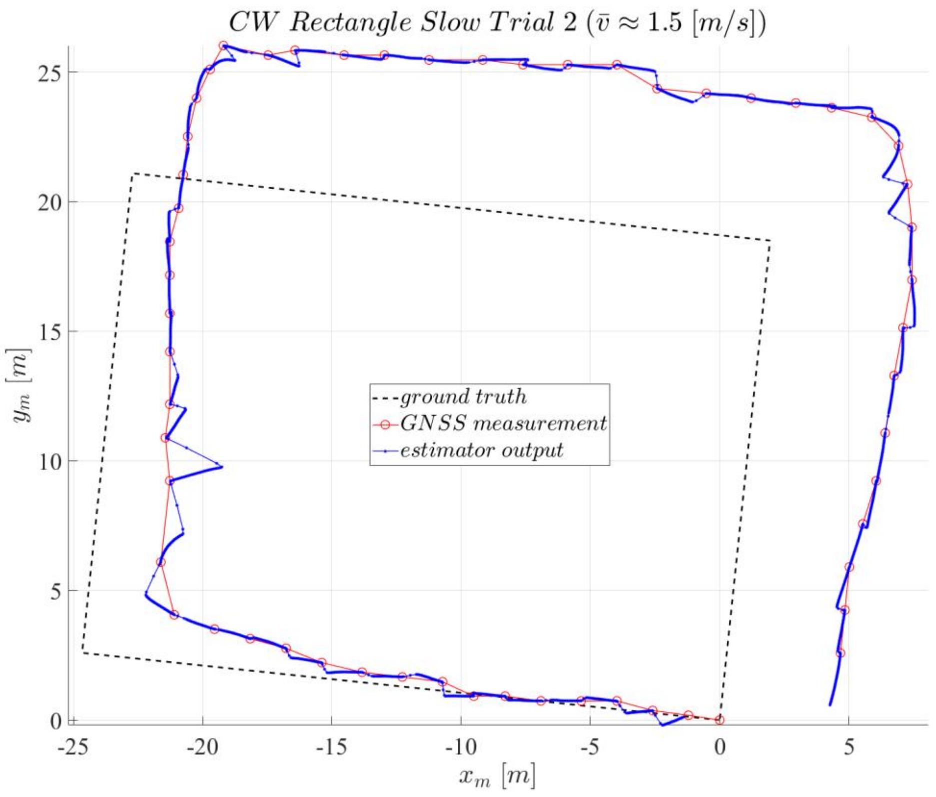

Figure 67: “Top-down” (x_m versus y_m) clockwise rectangle slow trial 2. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx 1.5\ [m/s]$.

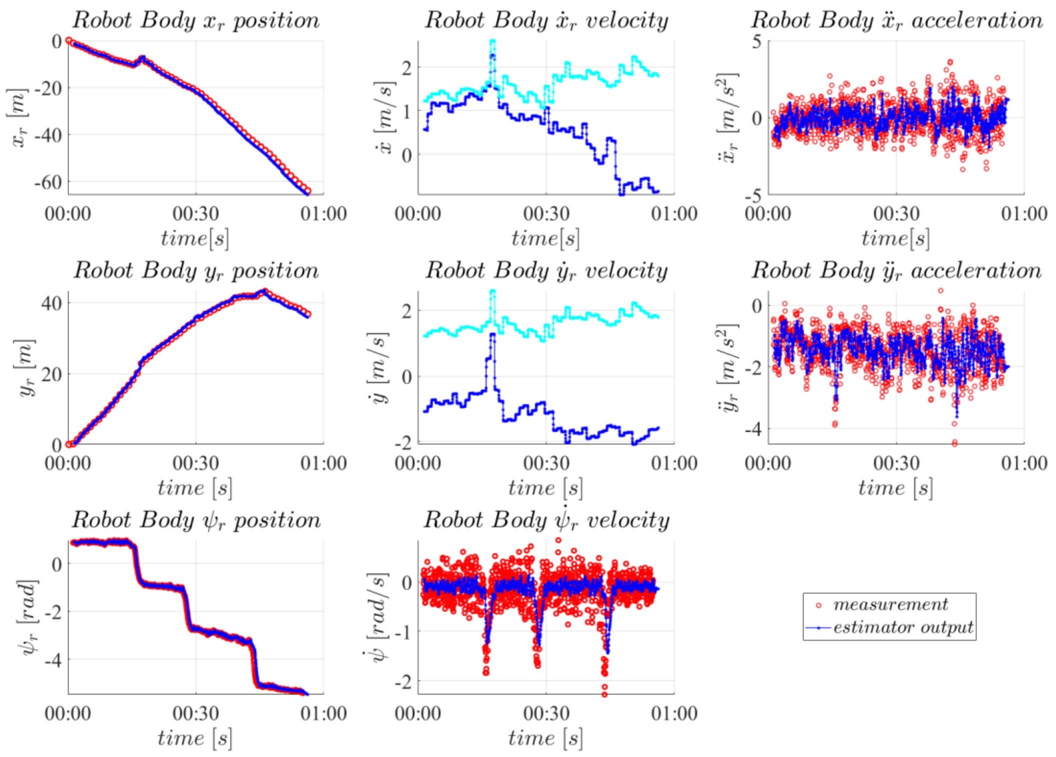

Figure 68: Robot body frame states plotted versus time for the clockwise rectangle slow trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.5 $[m/s]$.

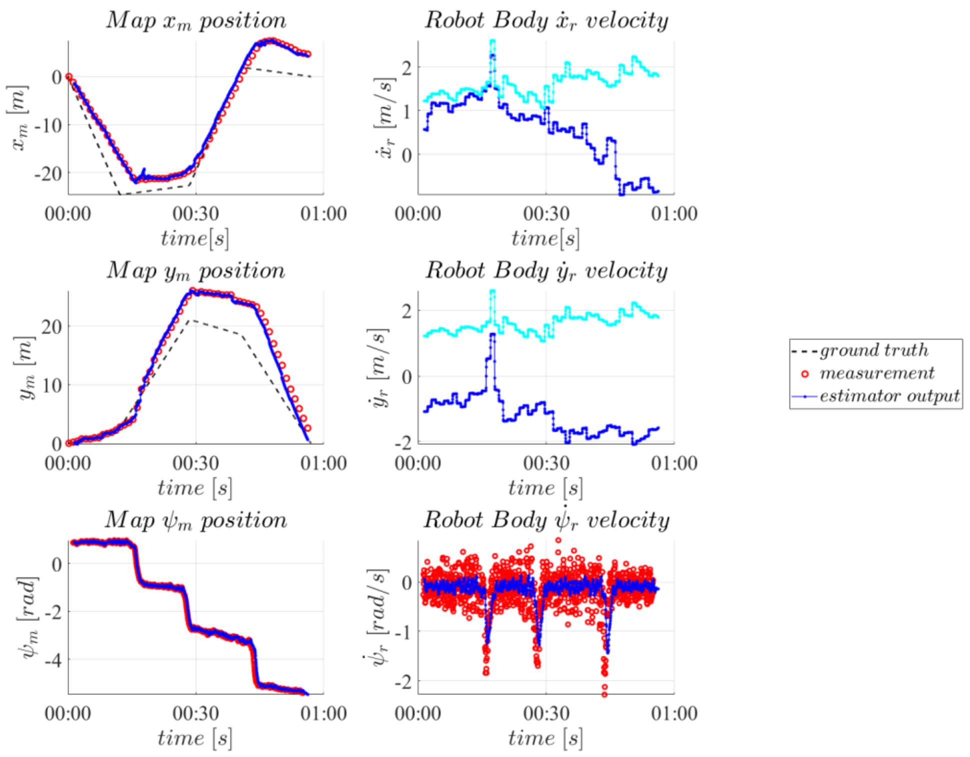

Figure 69: Hybrid frame states plotted versus time for the clockwise rectangle slow trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.5 $[m/s]$.

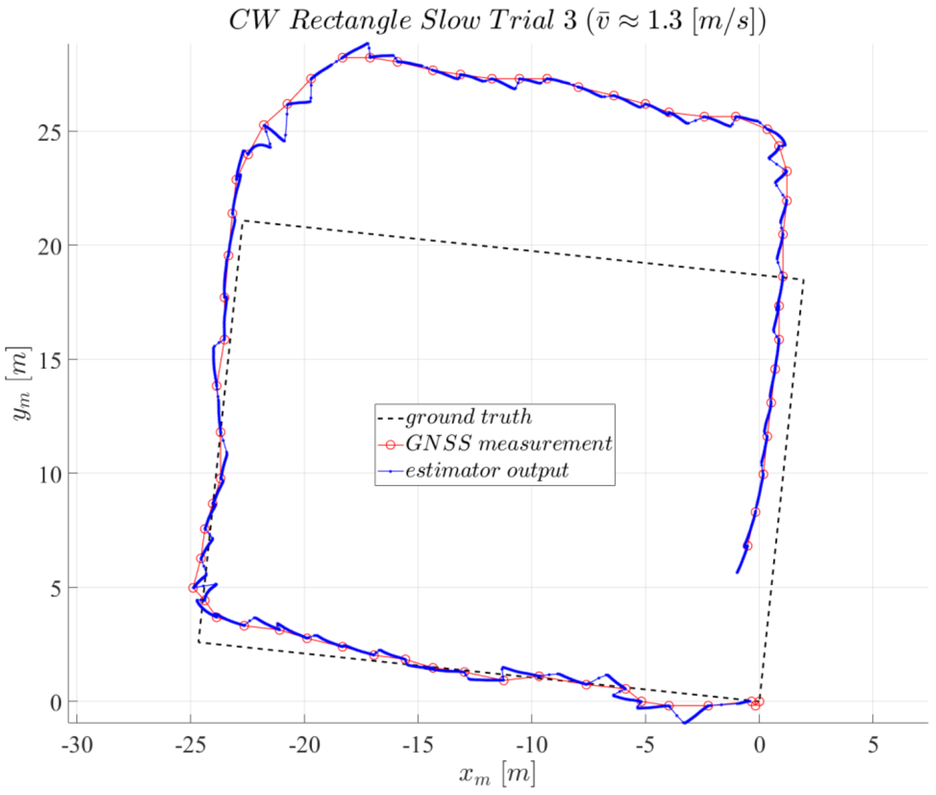

Figure 70: "Top-down" (x_m versus y_m) clockwise rectangle slow trial 3. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx 1.3\ [m/s]$.

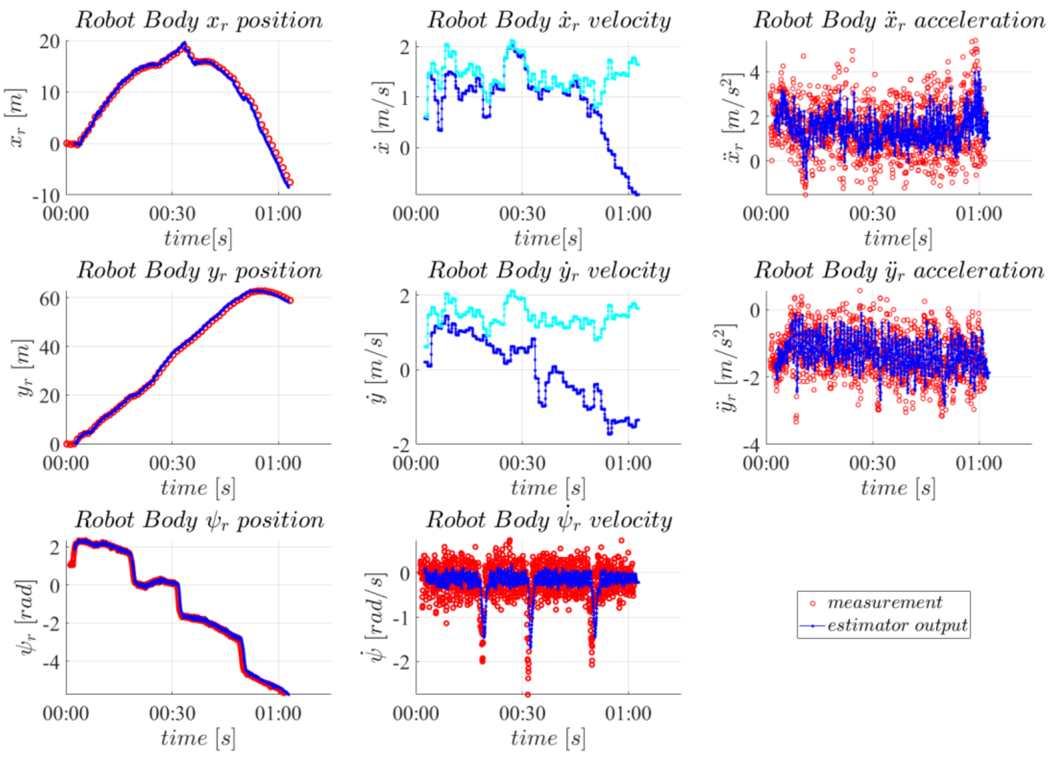

Figure 71: Robot body frame states plotted versus time for the clockwise rectangle slow trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.3 $[m/s]$.

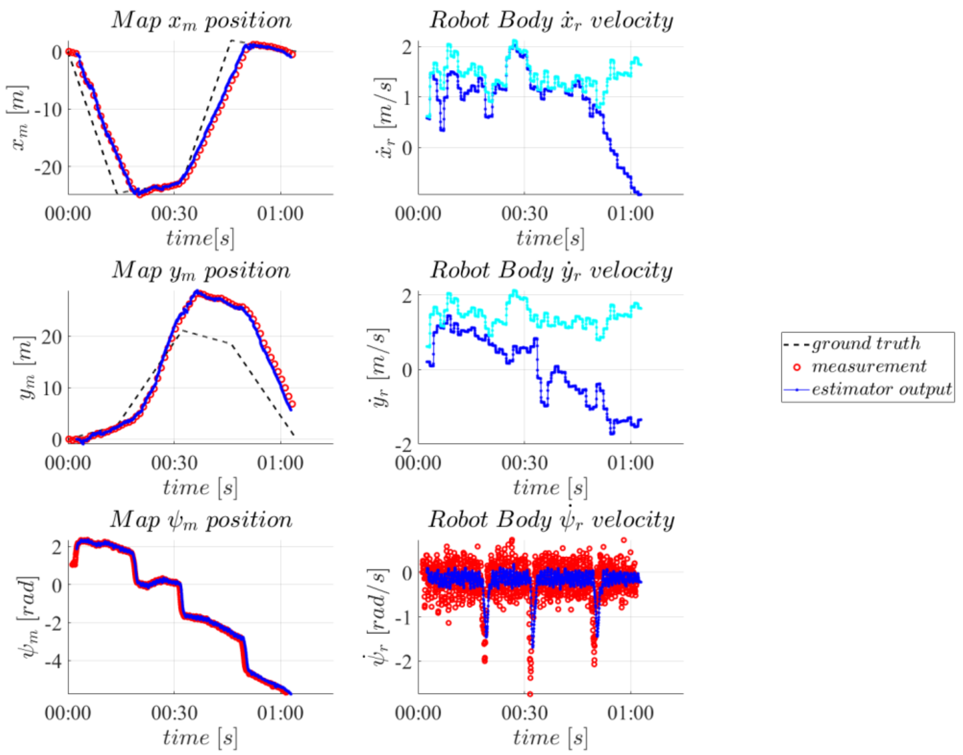

Figure 72: Hybrid frame states plotted versus time for the clockwise rectangle slow trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was $\approx 1.3\ [m/s]$.

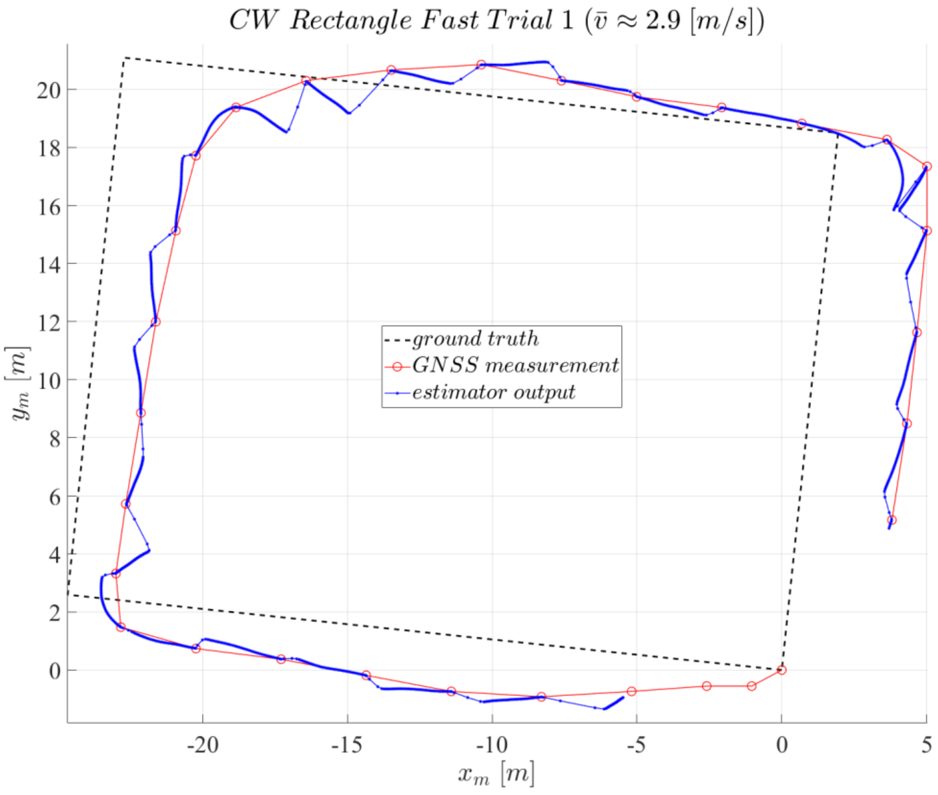

Figure 73: “Top-down” (x_m versus y_m) clockwise rectangle fast trial 1. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was ≈ 2.9 $[m/s]$.

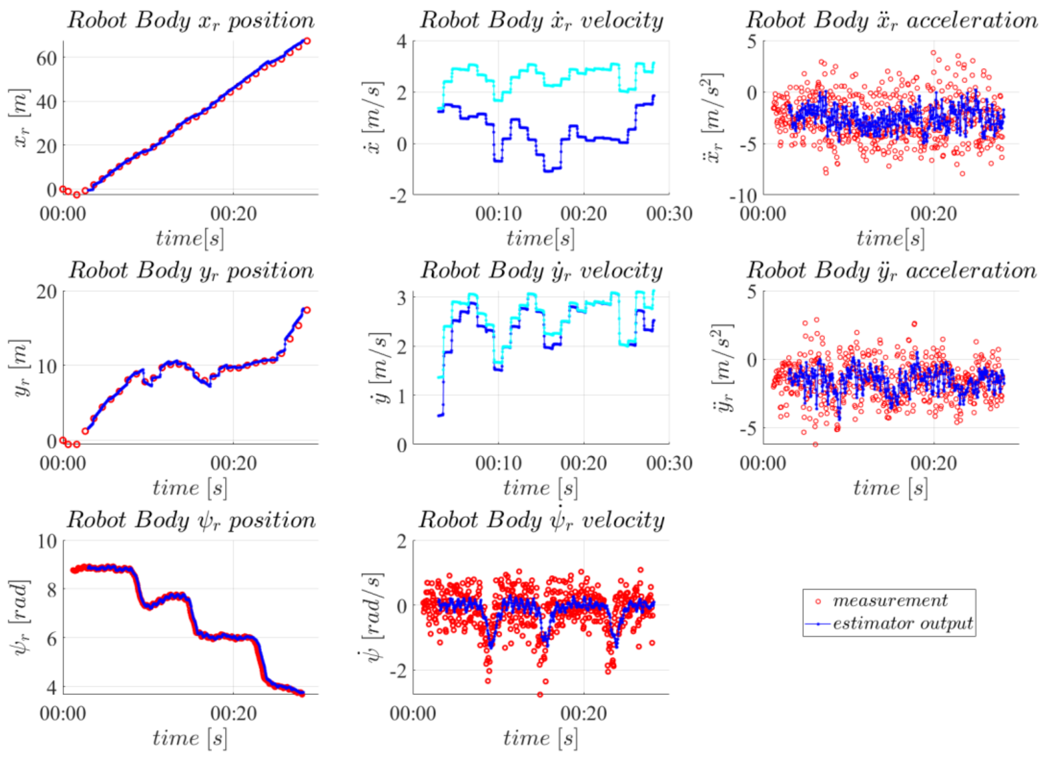

Figure 74: Robot body frame states plotted versus time for the clockwise rectangle fast trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 2.9 $[m/s]$.

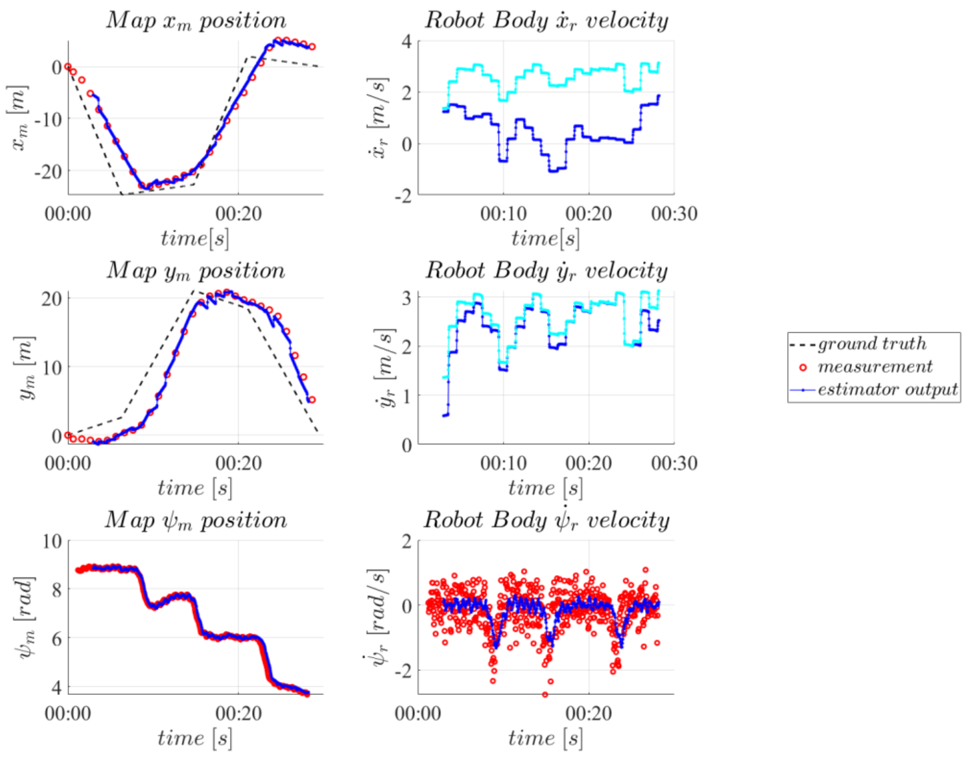

Figure 75: Hybrid frame states plotted versus time for the clockwise rectangle fast trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was $\approx 2.9\ [m/s]$.

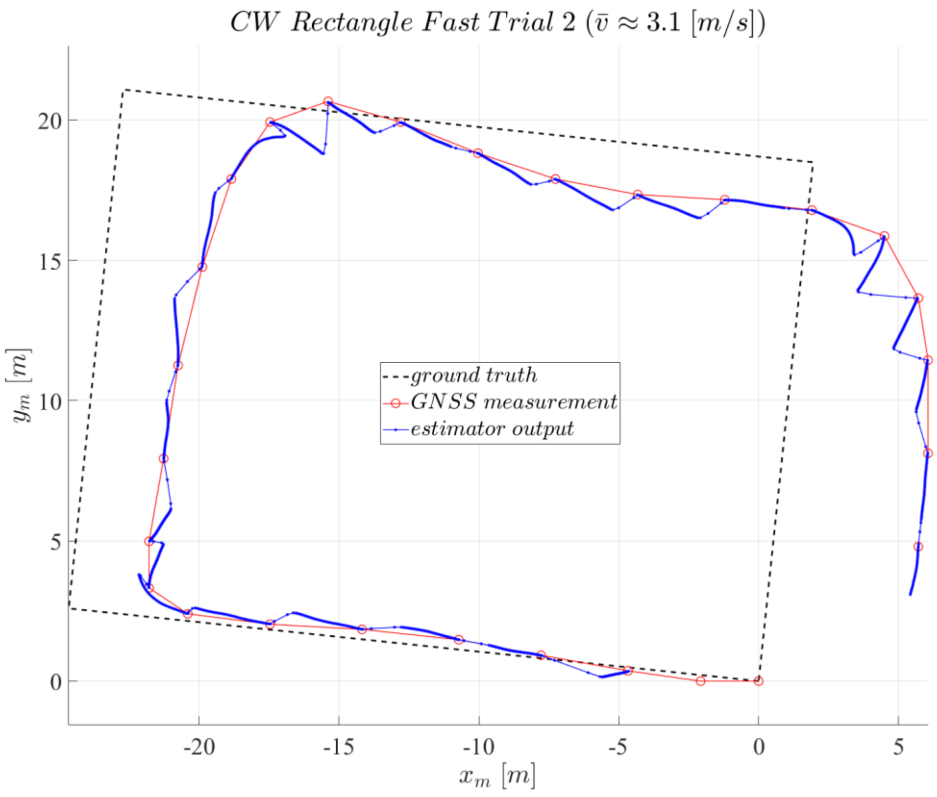

Figure 76: "Top-down" (x_m versus y_m) clockwise rectangle fast trial 2. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx 3.1\ [m/s]$.

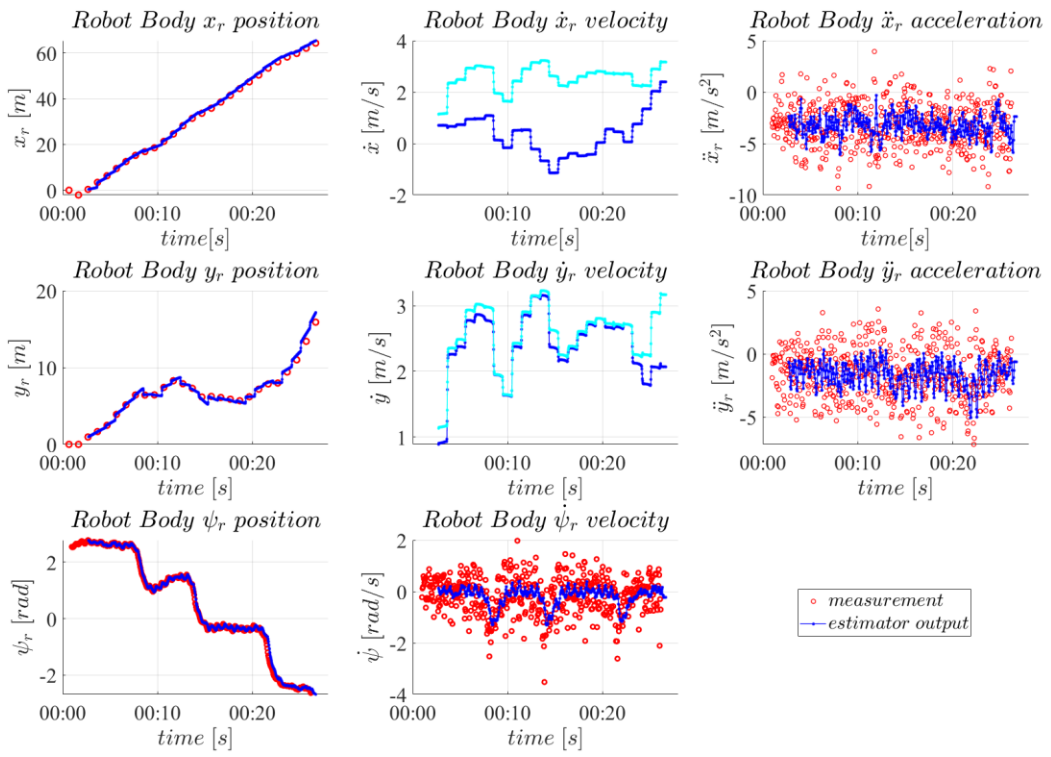

Figure 77: Robot body frame states plotted versus time for the clockwise rectangle fast trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 3.1 $[m/s]$.

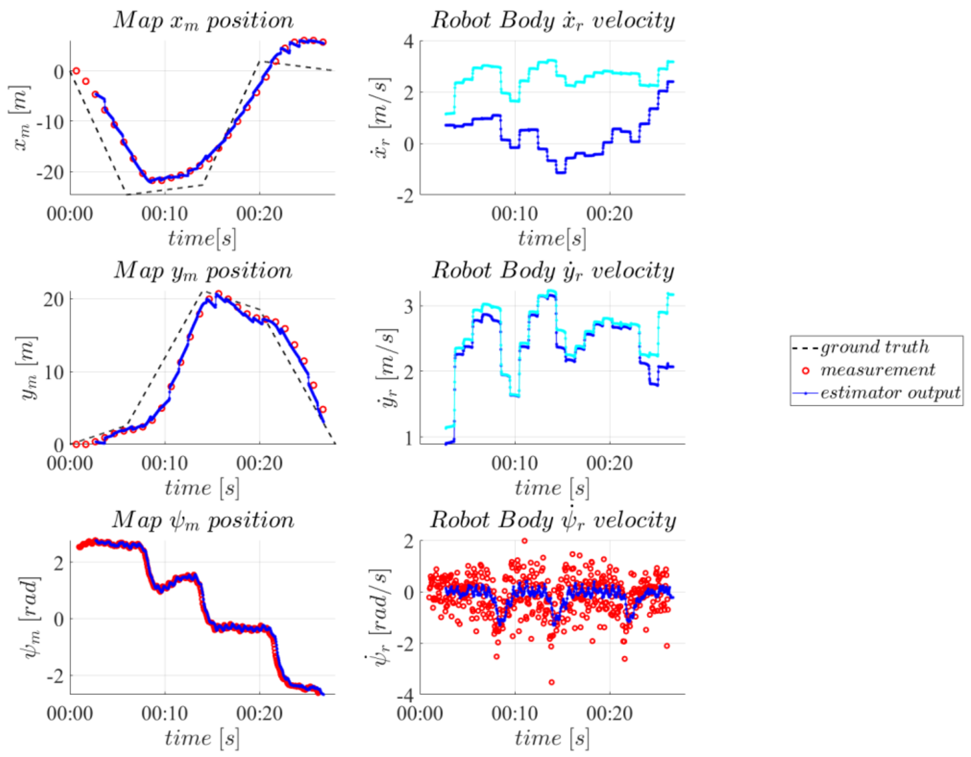

Figure 78: Hybrid frame states plotted versus time for the clockwise rectangle fast trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 3.1 $[m/s]$.

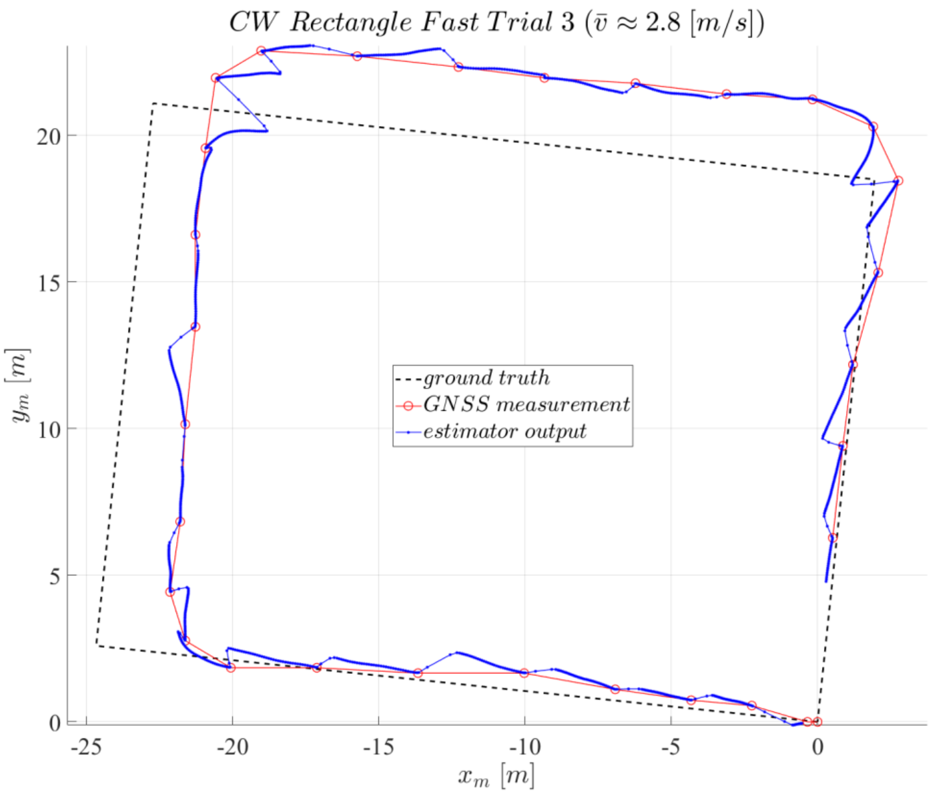

Figure 79: “Top-down” (x_m versus y_m) clockwise rectangle fast trial 3. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was ≈ 2.8 $[m/s]$.

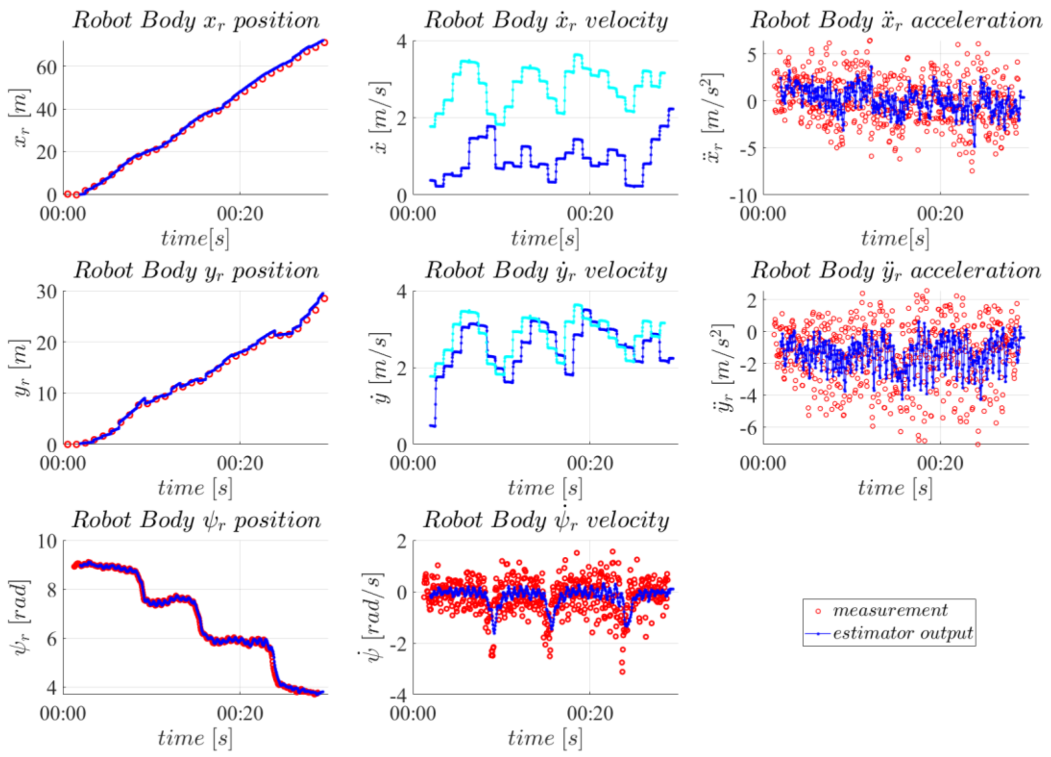

Figure 80: Robot body frame states plotted versus time for the clockwise rectangle fast trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 2.8 $[m/s]$.

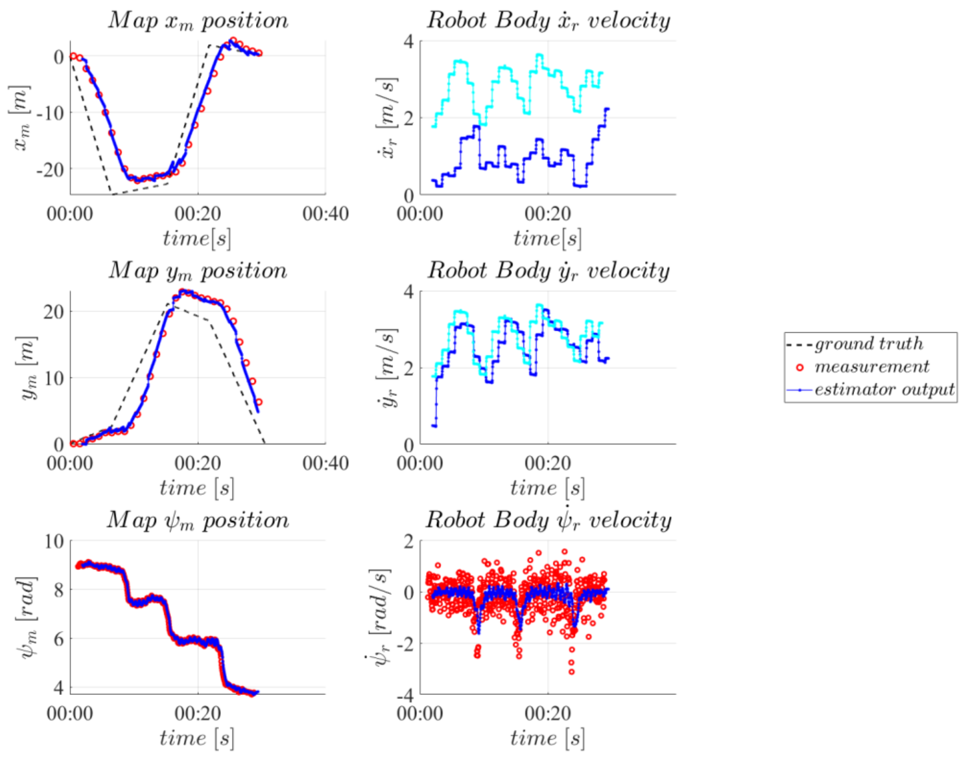

Figure 81: Hybrid frame states plotted versus time for the clockwise rectangle fast trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was $\approx 2.8\ [m/s]$.

As previously discussed, I am not particularly interested in the absolute accuracy of the linear position pose measurements from the state estimator, since the error of the linear position estimates is floored by the error in the GNSS linear position measurements. Instead, I am more interested in evaluating the ability of the state estimator to "dead-reckon" state estimates in between the infrequent linear position measurements from the GNSS, i.e., how well the estimator predicts the next measurement. The two methods I use to evaluate this error reflect this ideology. In the first method, I calculate the error of each GNSS measurement (converted to the map frame) relative to the known ground truth; I repeat this process for the linear position pose estimates. Taking the mean of the GNSS measurement error, and the linear position pose estimates allows us to compare the error of linear position pose estimates against the measurement error. In the second method, I interpolate the GNSS linear position measurements to the same time scale as the linear position estimates, and calculate the error in between them. In the first method, I gain understanding of the error performance of the full state estimation algorithm, and the contributive benefit of the probabilistic state estimator. In the second method, I directly evaluate the state estimator's ability to dead-reckon the next linear position measurement. These errors are compiled in Table 8 for method one, and Table 9 for method two.

Table 8: GNSS and Estimator Linear Position Mean Error Magnitude Against Ground Truth and Change in Error Between Them

Trial	GNSS mean error magnitude against ground truth $\pm$ 1 σ	Estimator mean error magnitude against ground truth $\pm$ 1 σ	Change in mean error magnitude from GNSS to Estimator [%]	
CCW Slow 1	3.21 $\pm$ 2.15 m	2.83 $\pm$ 1.90 m	-11.8 %	-1.3 %
CCW Slow 2	1.19 $\pm$ 0.99 m	1.50 $\pm$ 0.92 m	26.1 %	
CCW Slow 3	1.72 $\pm$ 1.16 m	1.41 $\pm$ 1.11 m	-18.2 %	
CCW Fast 1	3.12 $\pm$ 2.30 m	3.35 $\pm$ 1.99 m	7.2 %	3.8 %
CCW Fast 2	2.70 $\pm$ 2.17 m	2.50 $\pm$ 2.18 m	-7.6 %	
CCW Fast 3	1.45 $\pm$ 1.14 m	1.62 $\pm$ 1.27 m	11.7 %	
CW Slow 1	4.00 $\pm$ 2.71 m	4.09 $\pm$ 2.48 m	2.2 %	-0.9 %
CW Slow 2	4.88 $\pm$ 3.43 m	4.81 $\pm$ 3.16 m	-1.3 %	
CW Slow 3	5.49 $\pm$ 4.39 m	5.29 $\pm$ 4.01 m	-3.7 %	
CW Fast 1	4.54 $\pm$ 2.95 m	4.59 $\pm$ 2.84 m	1.2 %	-2.1 %
CW Fast 2	4.58 $\pm$ 2.94 m	4.55 $\pm$ 3.00 m	-0.8 %	
CW Fast 3	5.52 $\pm$ 4.13 m	5.16 $\pm$ 3.99 m	-6.5 %	

Table 9: Estimator Linear Position Mean Error Magnitude Against Interpolated GNSS Measurements

Trial	Estimator mean error magnitude versus interpolated GNSS measurements $\pm$ 1 σ	
CCW Slow 1	0.72 $\pm$ 0.63 m	0.80 $\pm$ 0.71 m
CCW Slow 2	0.84 $\pm$ 0.74 m	
CCW Slow 3	0.82 $\pm$ 0.74 m	
CCW Fast 1	0.72 $\pm$ 0.61 m	0.62 $\pm$ 0.55 m
CCW Fast 2	0.77 $\pm$ 0.68 m	
CCW Fast 3	0.38 $\pm$ 0.38 m	
CW Slow 1	0.93 $\pm$ 0.88 m	0.66 $\pm$ 0.78 m
CW Slow 2	0.75 $\pm$ 0.84 m	
CW Slow 3	0.69 $\pm$ 0.62 m	
CW Fast 1	0.60 $\pm$ 0.57 m	0.79 $\pm$ 0.60 m
CW Fast 2	0.62 $\pm$ 0.53 m	
CW Fast 3	0.77 $\pm$ 0.69 m	

4.2.3. Estimator performance on circular travel path

In this experiment, I repeat the same experiment as the last sub-section, but this time walking along a marked circle with dimensions. Using these known dimensions, I can correlate the measurements and estimator outputs to a “ground truth” and calculate the accuracy of the measurements and the estimator output. The circular travel path is illustrated in Figure 82.

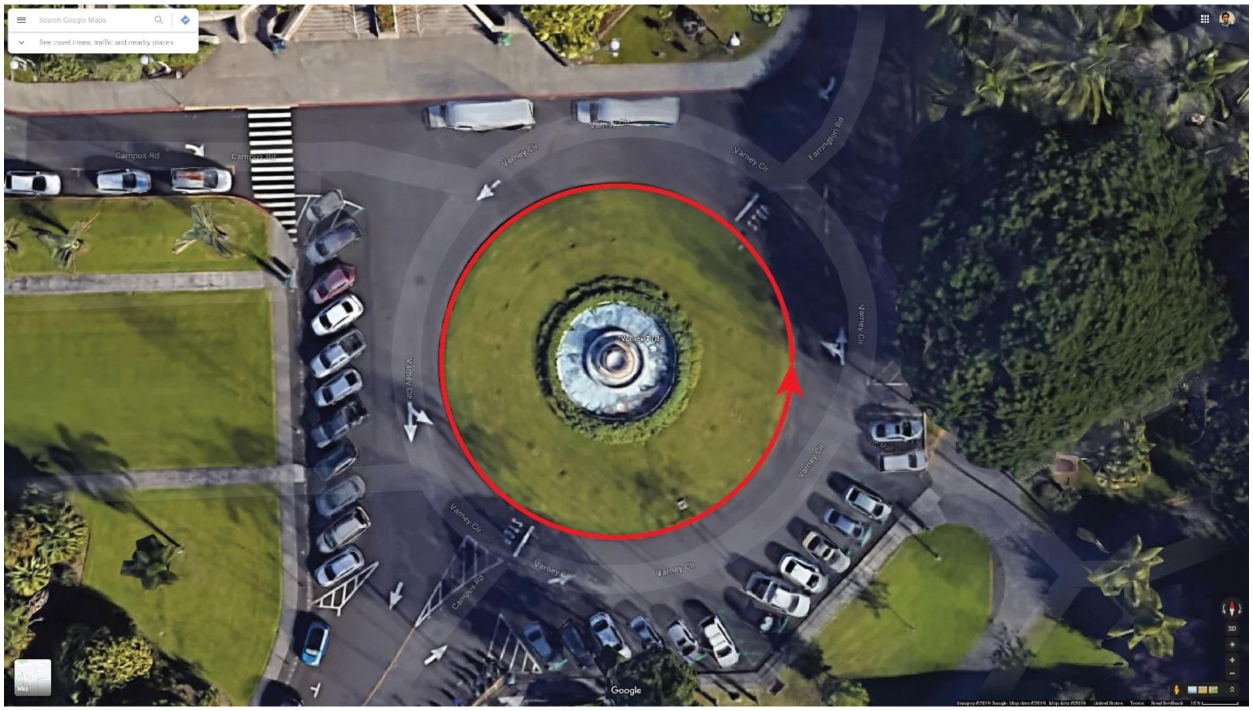

Figure 82: Counter-clockwise circular travel path. The diameter of the circle is 100 ft (30.48 m). Map data © 2018 Google. Reproduced with permission from Google Maps Geoguidelines, 2019 [123].

The counter-clockwise “slow” ($\bar{v} \approx 1.2\ to\ 1.3\ [m/s]$) trials one, two and three are shown in Figure 83 through Figure 91. The counter-clockwise “fast” ($\bar{v} \approx 2.6\ [m/s]$) trials one, two, and three are shown in Figure 92 through Figure 100.

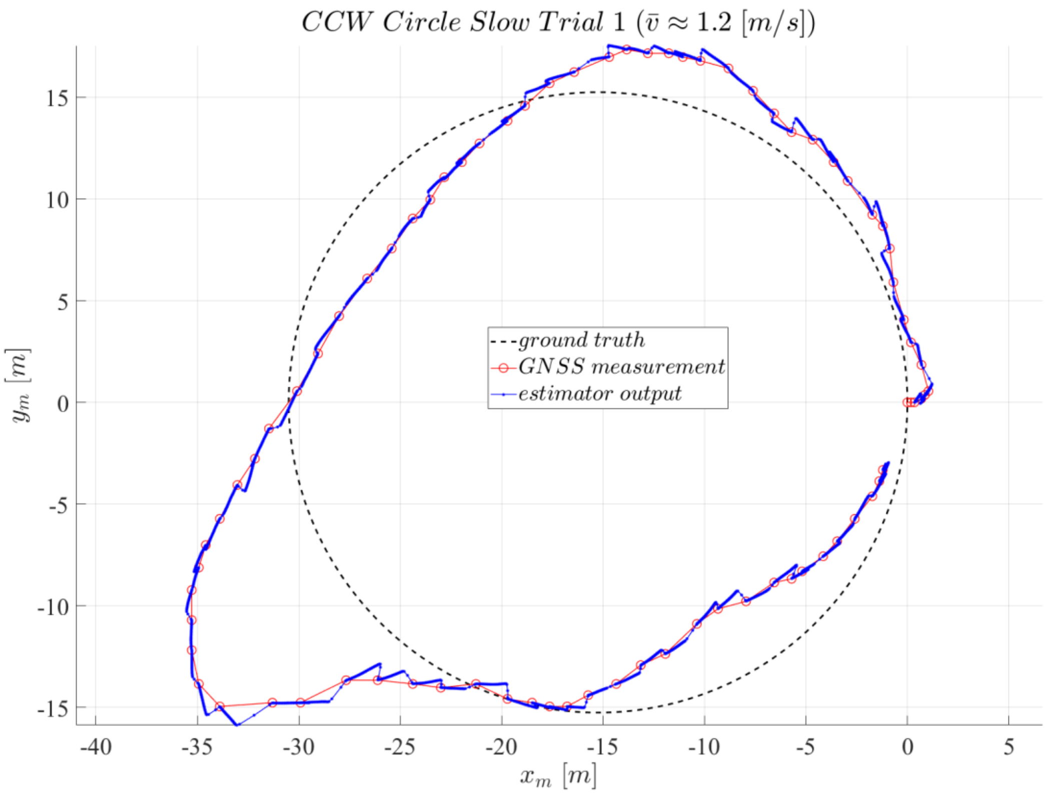

Figure 83: "Top-down" (x_m versus y_m) counterclockwise circle slow trial 1. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was ≈ 1.2 $[m/s]$.

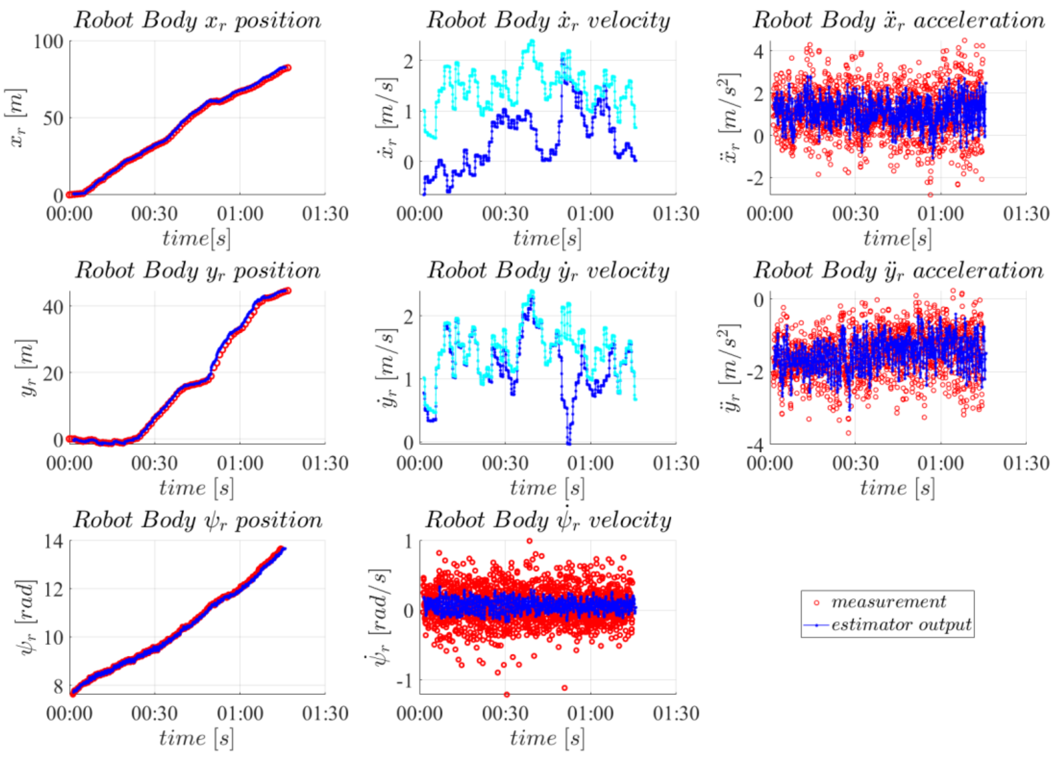

Figure 84: Robot body frame states plotted versus time for the counterclockwise circle slow trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.2 $[m/s]$.

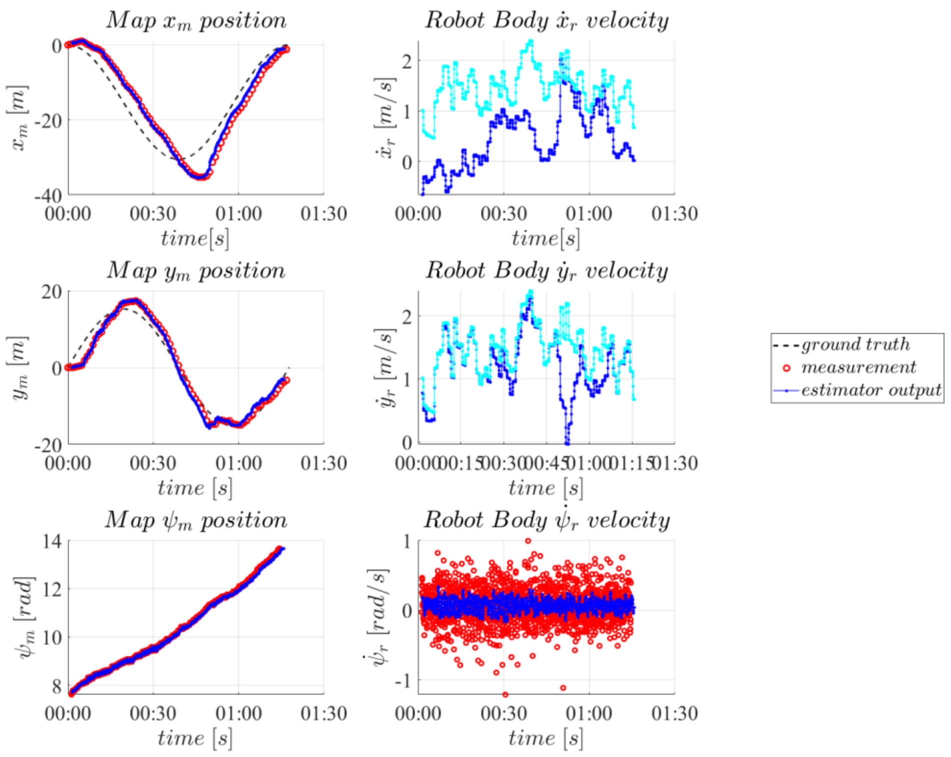

Figure 85: Hybrid frame states plotted versus time for the counterclockwise circle slow trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.2 $[m/s]$.

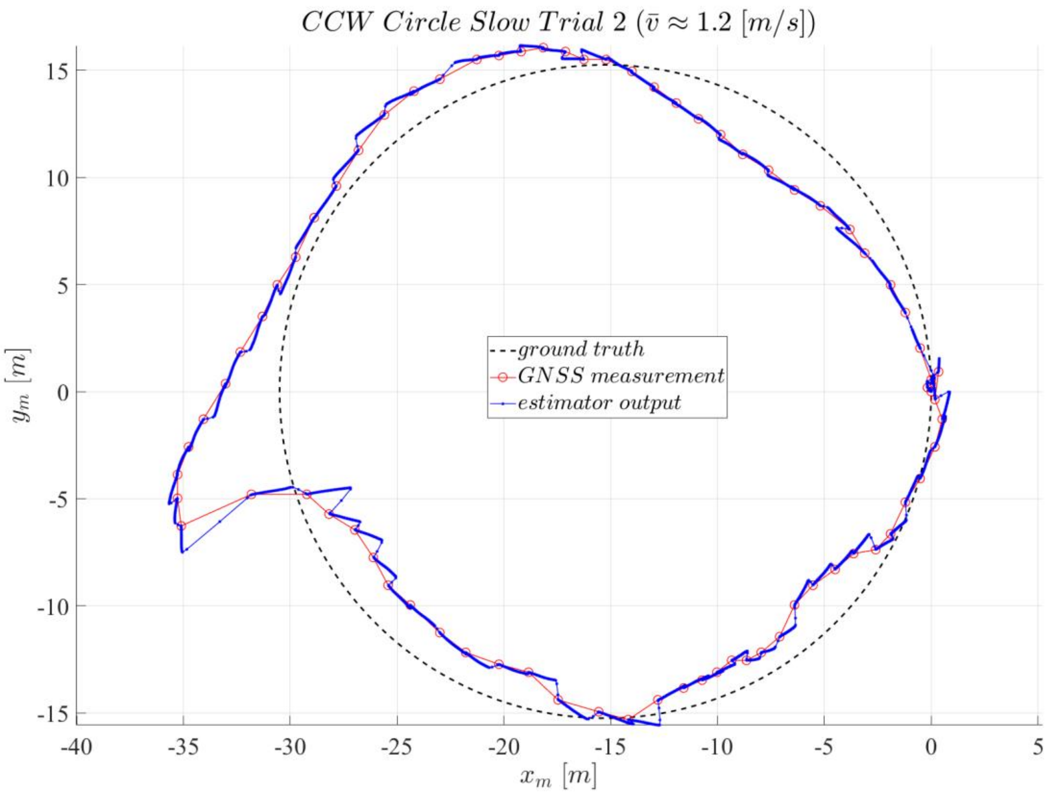

Figure 86: "Top-down" (x_m versus y_m) counterclockwise circle slow trial 2. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx 1.2\ [m/s]$.

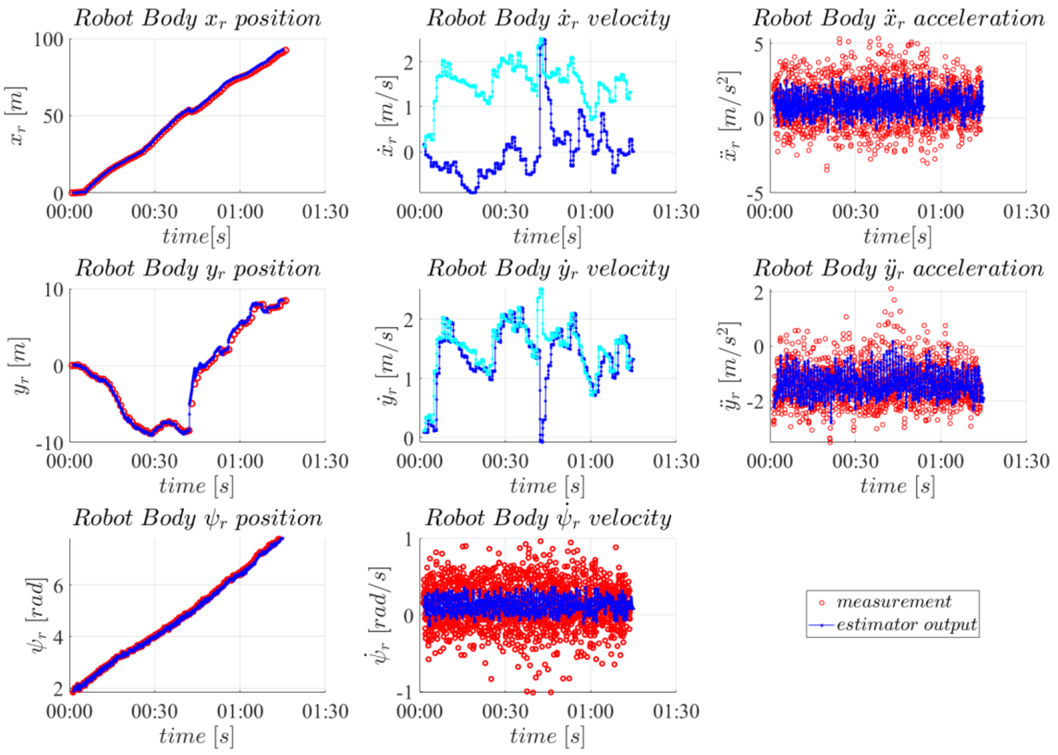

Figure 87: Robot body frame states plotted versus time for the counterclockwise circle slow trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.2 $[m/s]$.

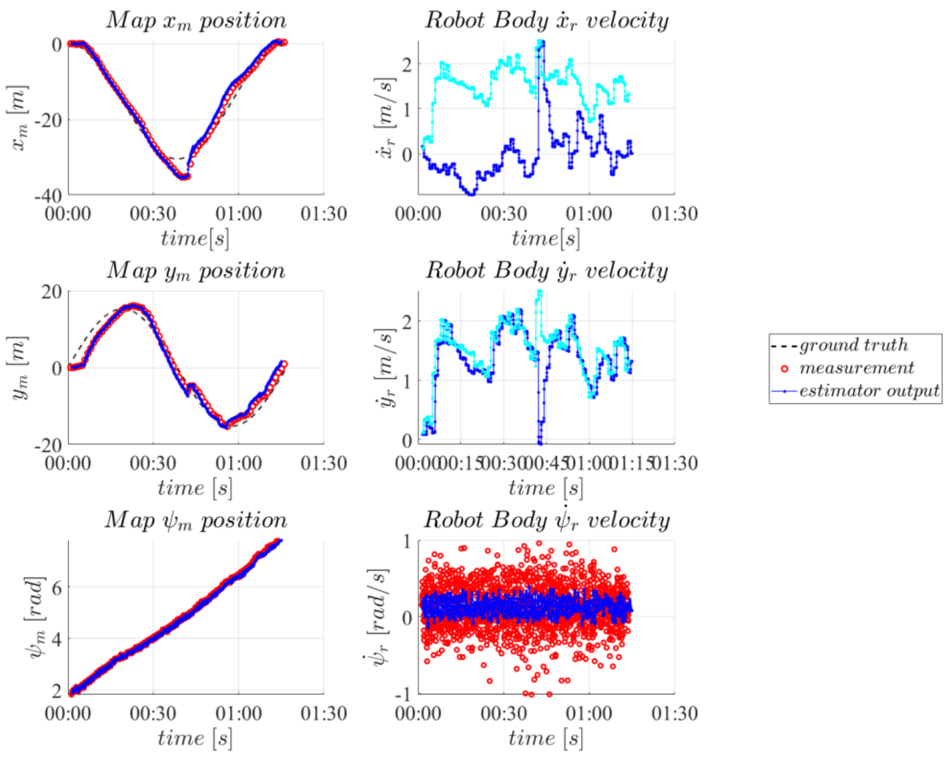

Figure 88: Hybrid frame states plotted versus time for the counterclockwise circle slow trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.2 $[m/s]$.

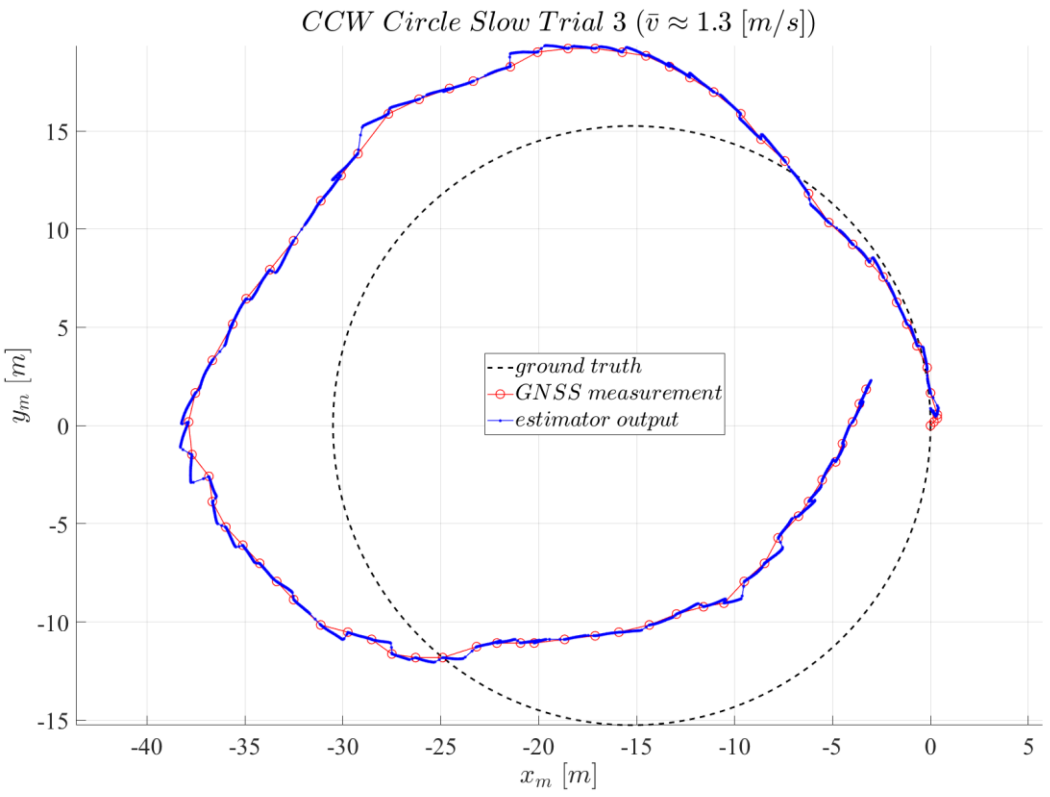

Figure 89: "Top-down" (x_m versus y_m) counterclockwise circle slow trial 3. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was $\approx 1.3\ [m/s]$.

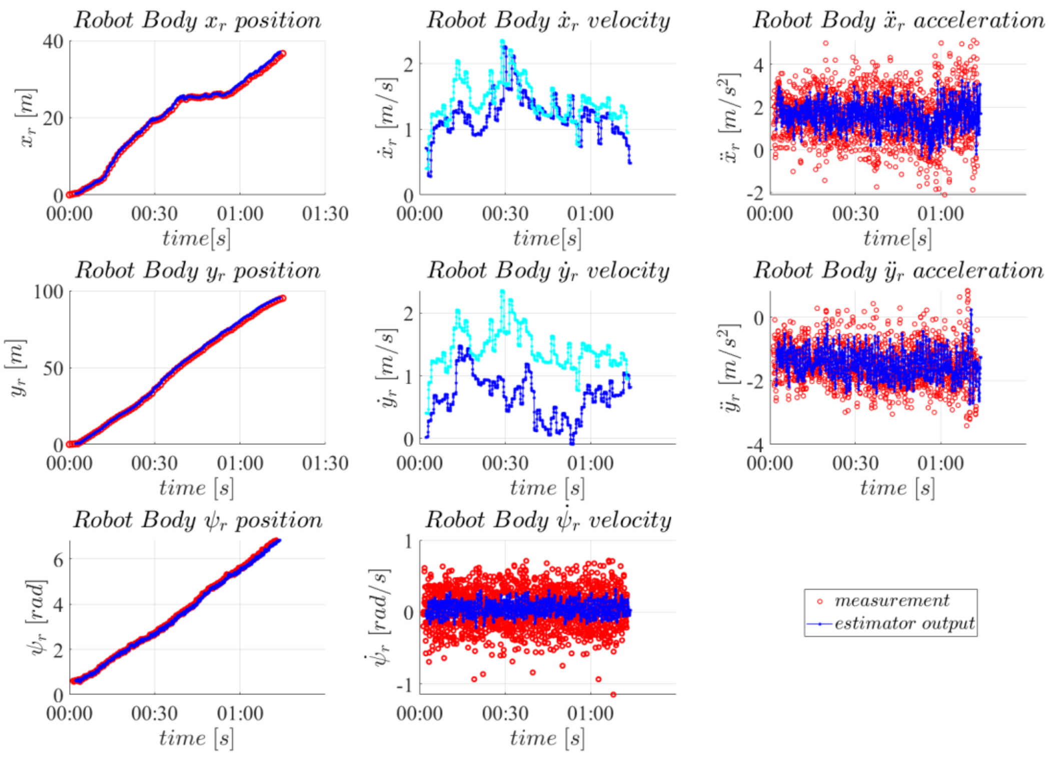

Figure 90: Robot body frame states plotted versus time for the counterclockwise circle slow trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.3 $[m/s]$.

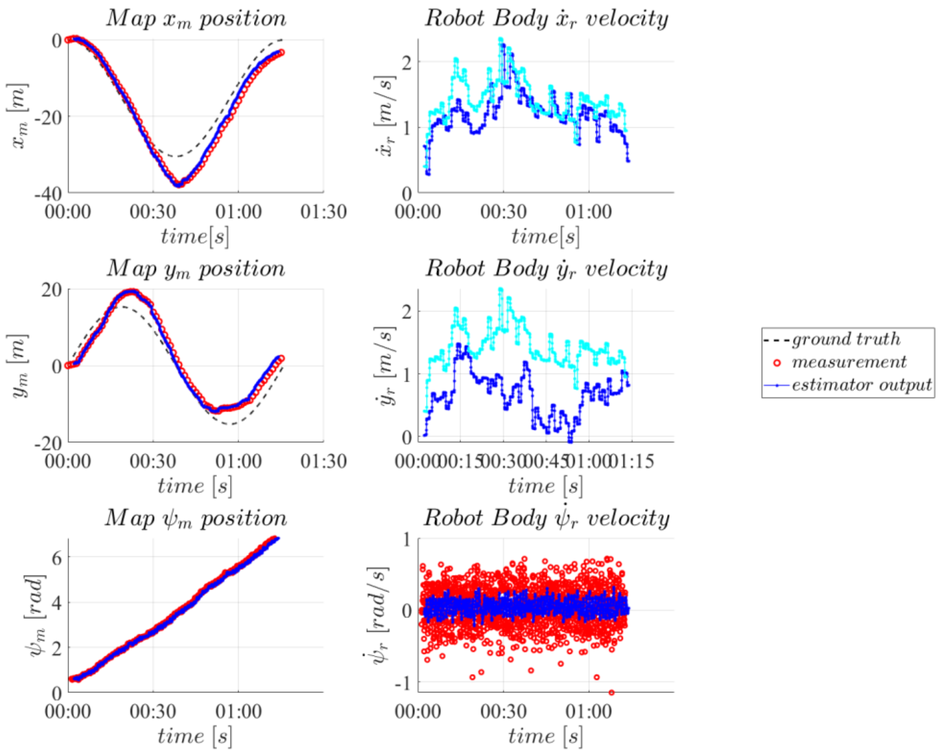

Figure 91: Hybrid frame states plotted versus time for the counterclockwise circle slow trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 1.3 $[m/s]$.

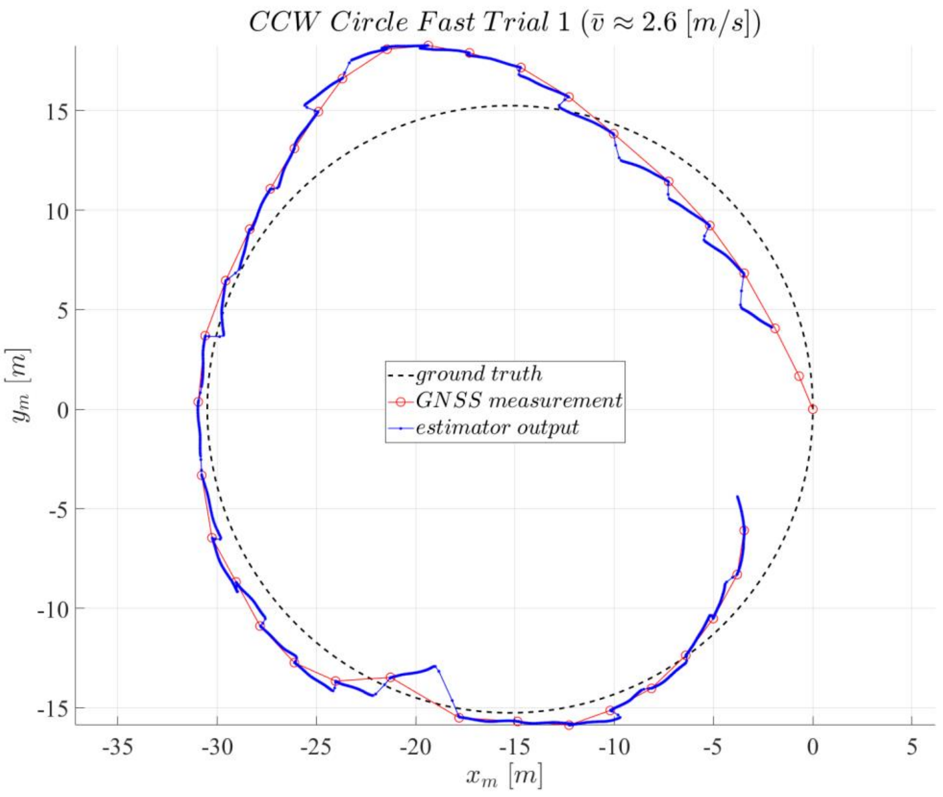

Figure 92: “Top-down” (x_m versus y_m) counterclockwise circle fast trial 1. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was ≈ 2.6 $[m/s]$.

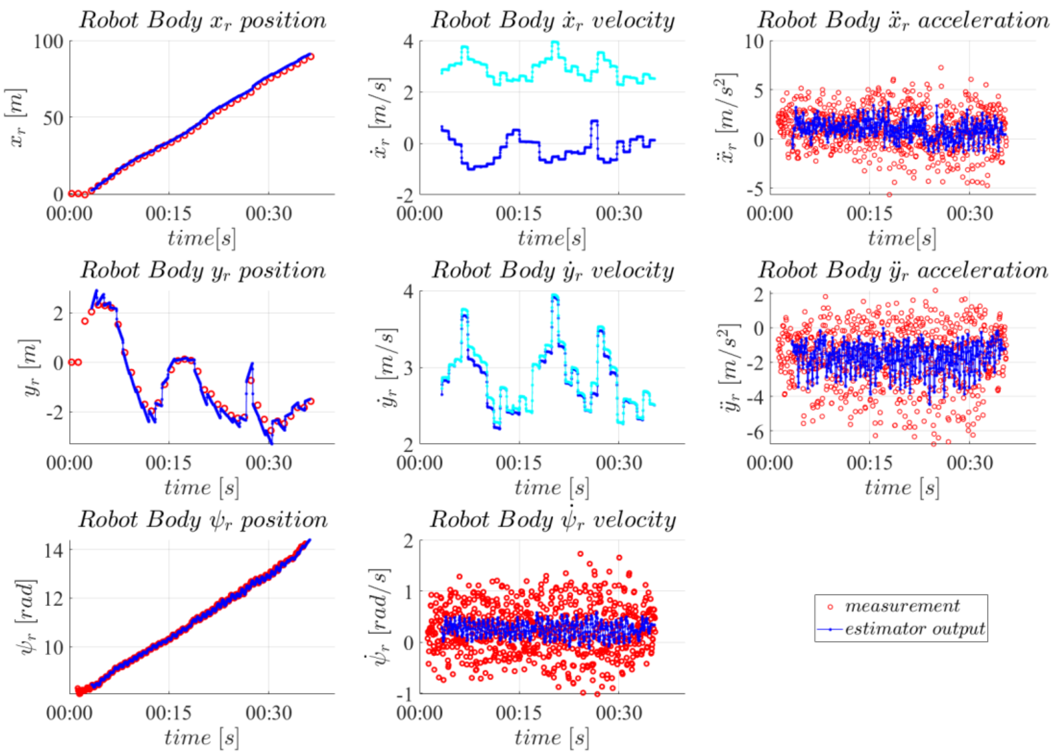

Figure 93: Robot body frame states plotted versus time for the counterclockwise circle fast trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 2.6 $[m/s]$.

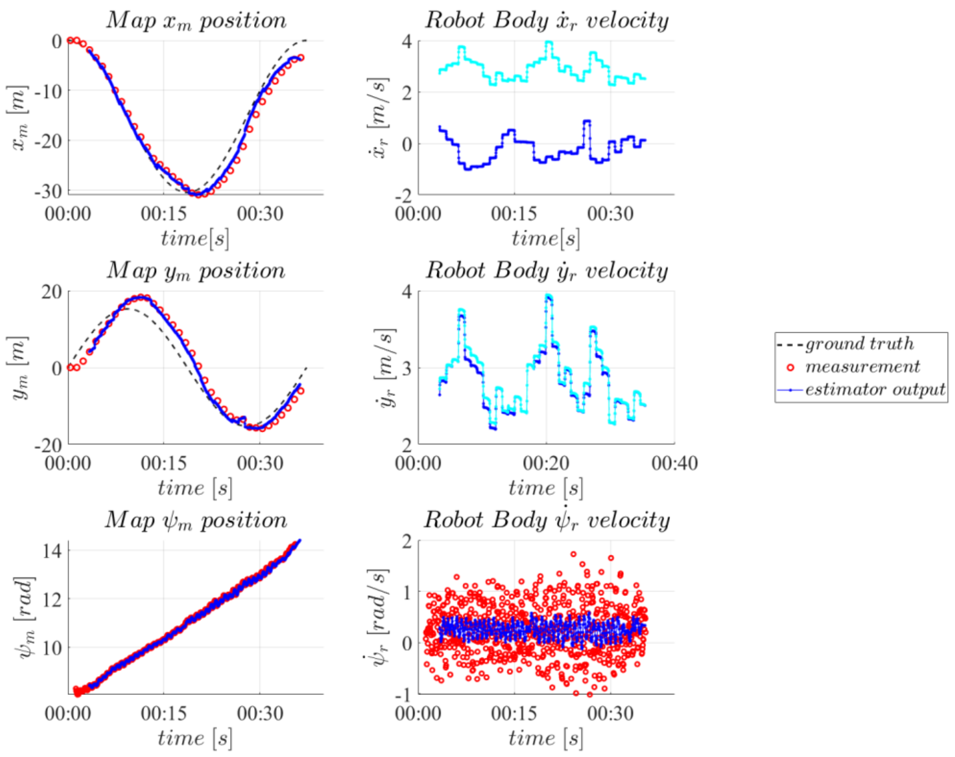

Figure 94: Hybrid frame states plotted versus time for the counterclockwise circle fast trial 1. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 2.6 $[m/s]$.

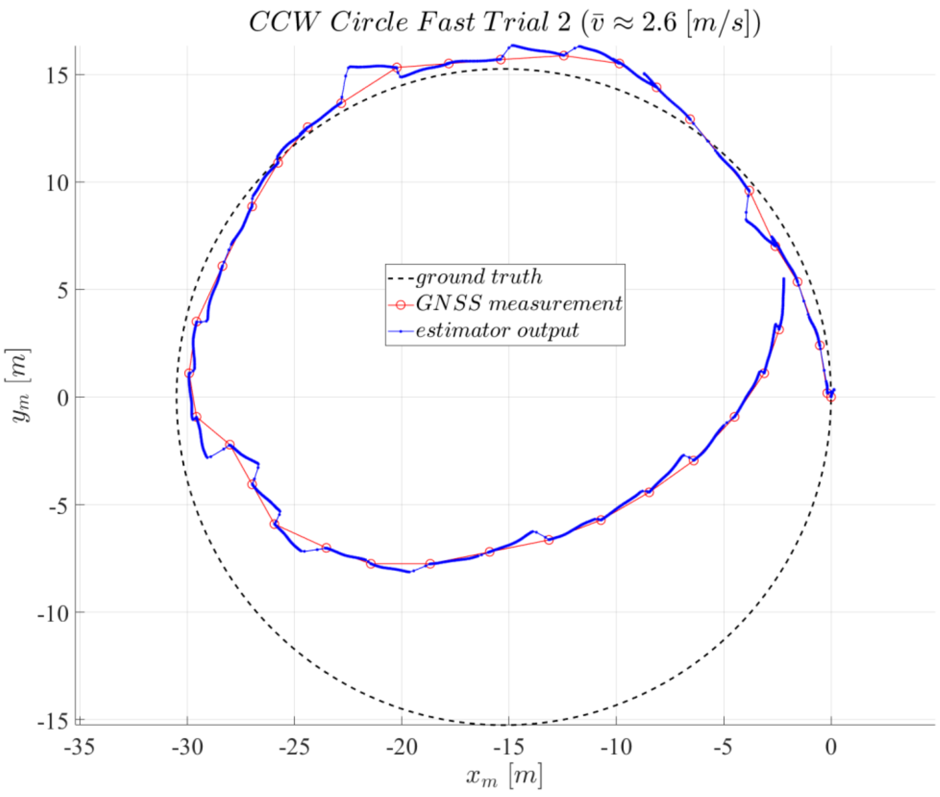

Figure 95: "Top-down" (x_m versus y_m) counterclockwise circle fast trial 2. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was ≈ 2.6 $[m/s]$.

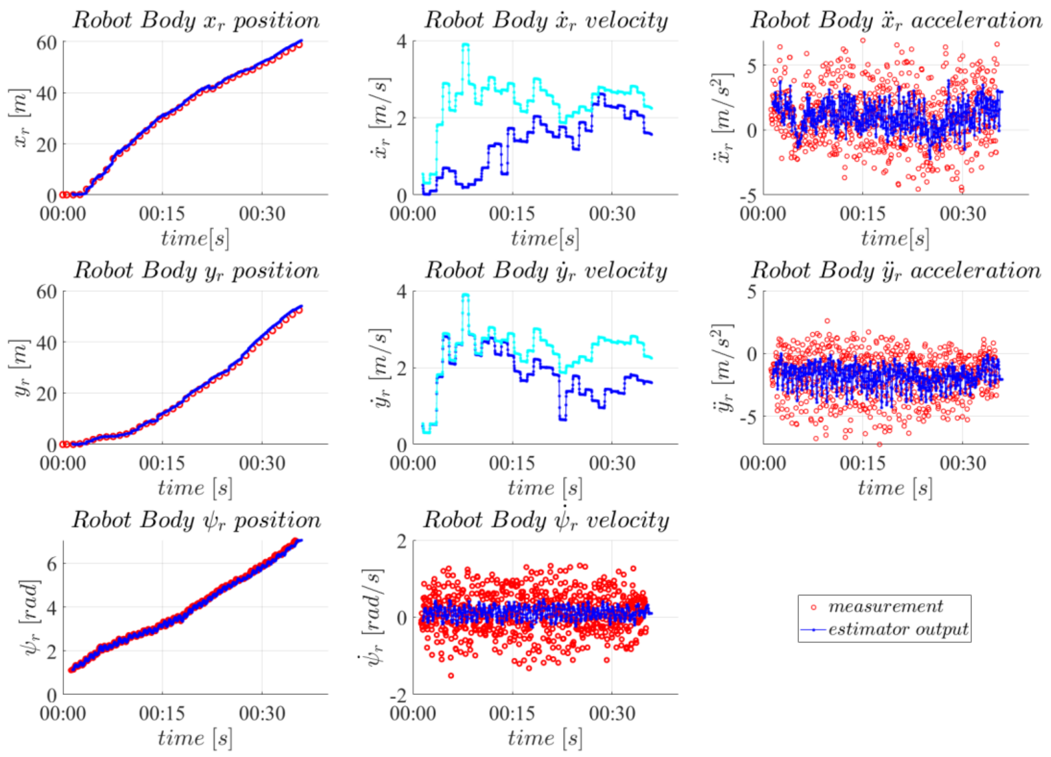

Figure 96: Robot body frame states plotted versus time for the counterclockwise circle fast trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 2.6 $[m/s]$.

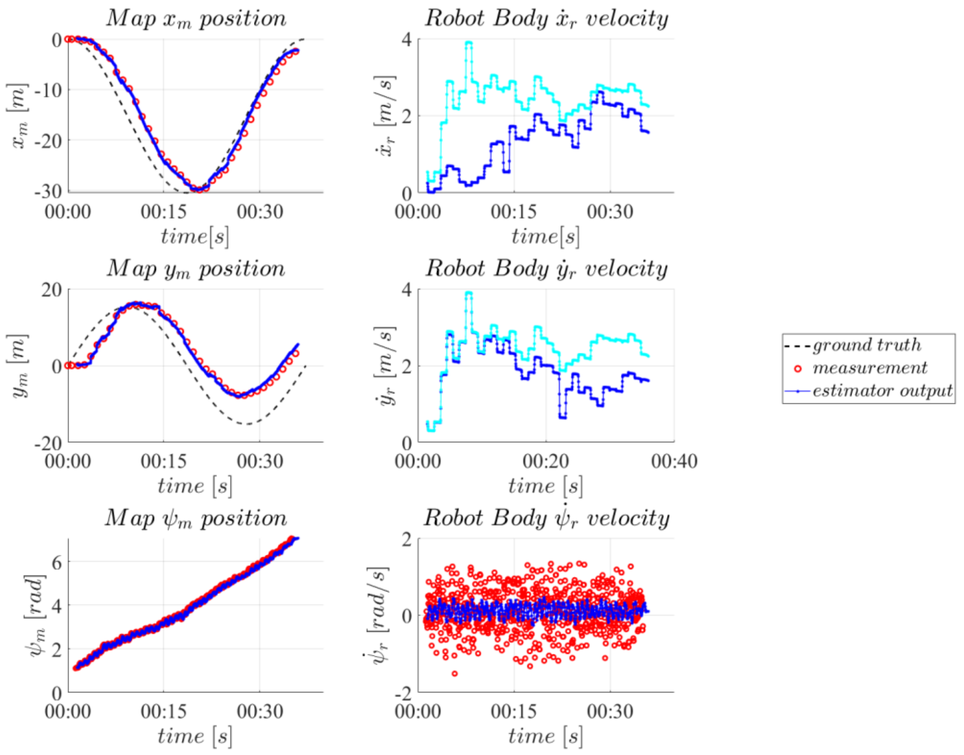

Figure 97: Hybrid frame states plotted versus time for the counterclockwise circle fast trial 2. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 2.6 $[m/s]$.

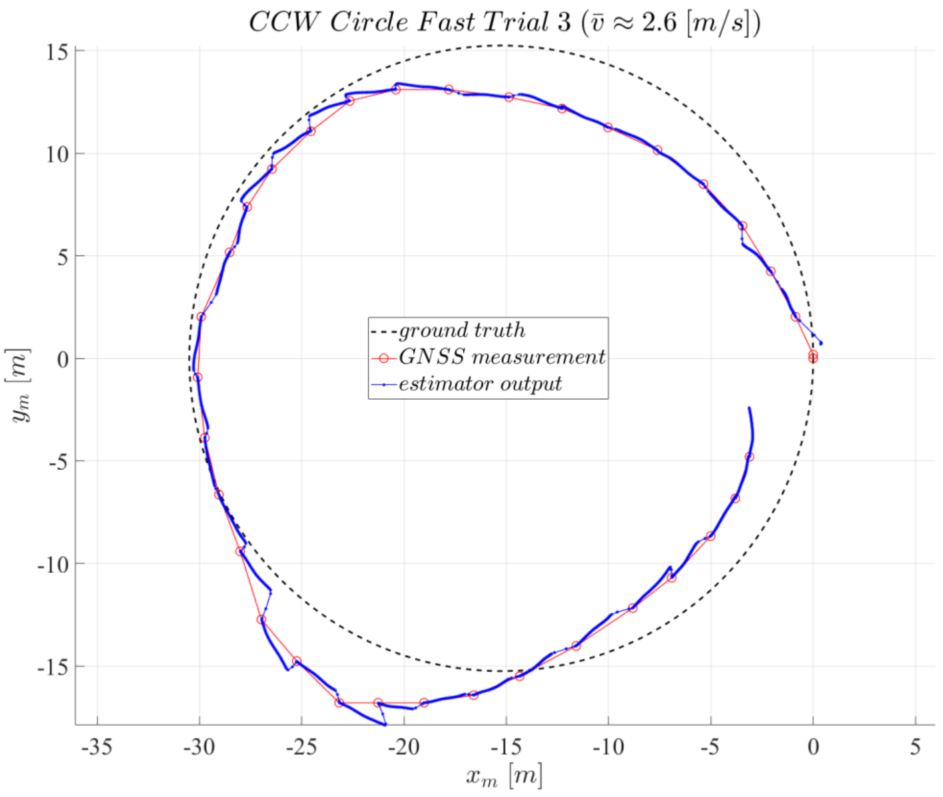

Figure 98: “Top-down” (x_m versus y_m) counterclockwise circle fast trial 3. The dashed black line represents the known ground truth, the red circle-connected line represents GNSS measurements, and the blue dot-connected line represents the estimator pose output. The average velocity for this trial was ≈ 2.6 $[m/s]$.

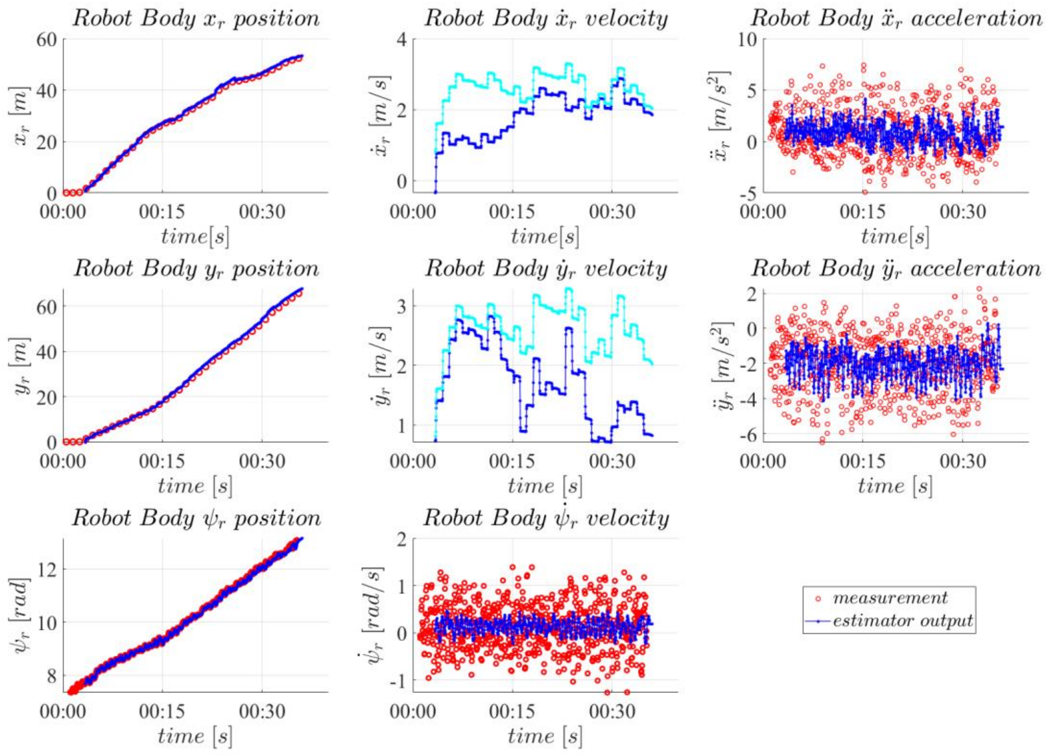

Figure 99: Robot body frame states plotted versus time for the counterclockwise circle fast trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was $\approx 2.6\ [m/s]$.

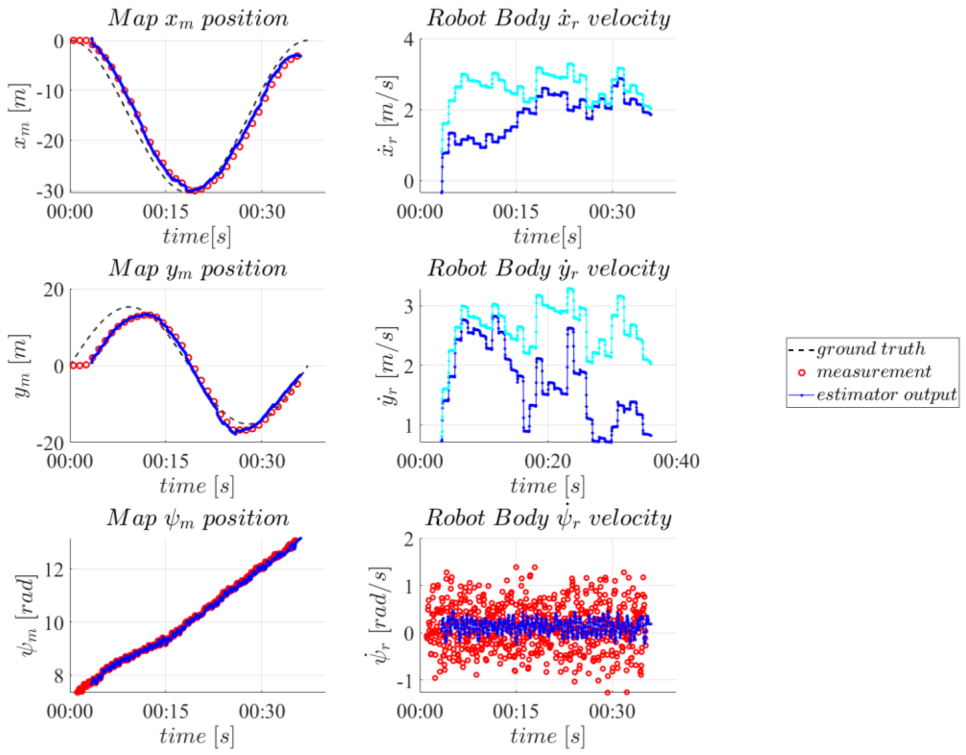

Figure 100: Hybrid frame states plotted versus time for the counterclockwise circle fast trial 3. The red circle-connected line represents GNSS measurements, the blue dot-connected line represents the estimator pose output, the dashed black line represents the known ground truth, and the cyan dot-connected line represents the absolute linear velocity (only in the x_r, y_r linear velocity states). The average velocity for this trial was ≈ 2.6 $[m/s]$.

Following the same error characterization methods discussed in the previous sub-section, I define the error between the linear position GNSS measurements and ground truth, and linear position estimates and ground truth in Table 10, and the error between the linear position estimates and interpolated GNSS measurements in Table 11.

Table 10: GNSS and Estimator Linear Position Mean Error Magnitude Against Ground Truth and Change in Error Between Them

Trial	GNSS mean error magnitude against ground truth ± 1 σ	Estimator mean error magnitude against ground truth ± 1 σ	Change in mean error magnitude from GNSS to Estimator [%]	
CCW Slow 1	4.62 ± 2.48 m	4.32 ± 2.36 m	-6.4 %	3.0%
CCW Slow 2	2.35 ± 1.67 m	2.82 ± 1.96 m	19.9 %	
CCW Slow 3	4.57 ± 2.44 m	4.36 ± 2.56 m	-4.4 %	
CCW Fast 1	3.66 ± 2.14 m	2.70 ± 1.80 m	-26.1 %	-11.2%

CCW Fast 2	5.99 $\pm$ 3.06 m	6.16 $\pm$ 3.10 m	2.9 %	
CCW Fast 3	3.02 $\pm$ 2.19 m	2.70 $\pm$ 2.04 m	-10.5 %	

Table 11: Estimator Linear Position Mean Error Magnitude Against Interpolated GNSS Measurements

Trial	Estimator mean error magnitude versus interpolated GNSS measurements $\pm$ 1 σ	
CCW Slow 1	0.92 $\pm$ 0.73 m	0.93 $\pm$ 0.71 m
CCW Slow 2	0.96 $\pm$ 0.73 m	
CCW Slow 3	0.91 $\pm$ 0.68 m	
CCW Fast 1	1.19 $\pm$ 0.87 m	0.99 $\pm$ 0.77 m
CCW Fast 2	0.89 $\pm$ 0.65 m	
CCW Fast 3	0.90 $\pm$ 0.68 m	

4.2.4. Estimator velocity tracking

In this subsection, I evaluate the steady-state accuracy of the estimator linear velocity measurement, as well as the time constant. In this experiment, I subjected the GNSS+IMU sensor package to a linear velocity step input. I then extracted the linear velocity state estimates and plotted them versus time. Because the linear velocity pseudo-measurements demonstrate a strong first order response behavior, I can fit a first order response curve to the data and calculate the time constant τ and steady state response v_{ss} [124]. I performed five trials over 32 m at three different speeds. The results are plotted in Figure 101, Figure 102, and Figure 103.

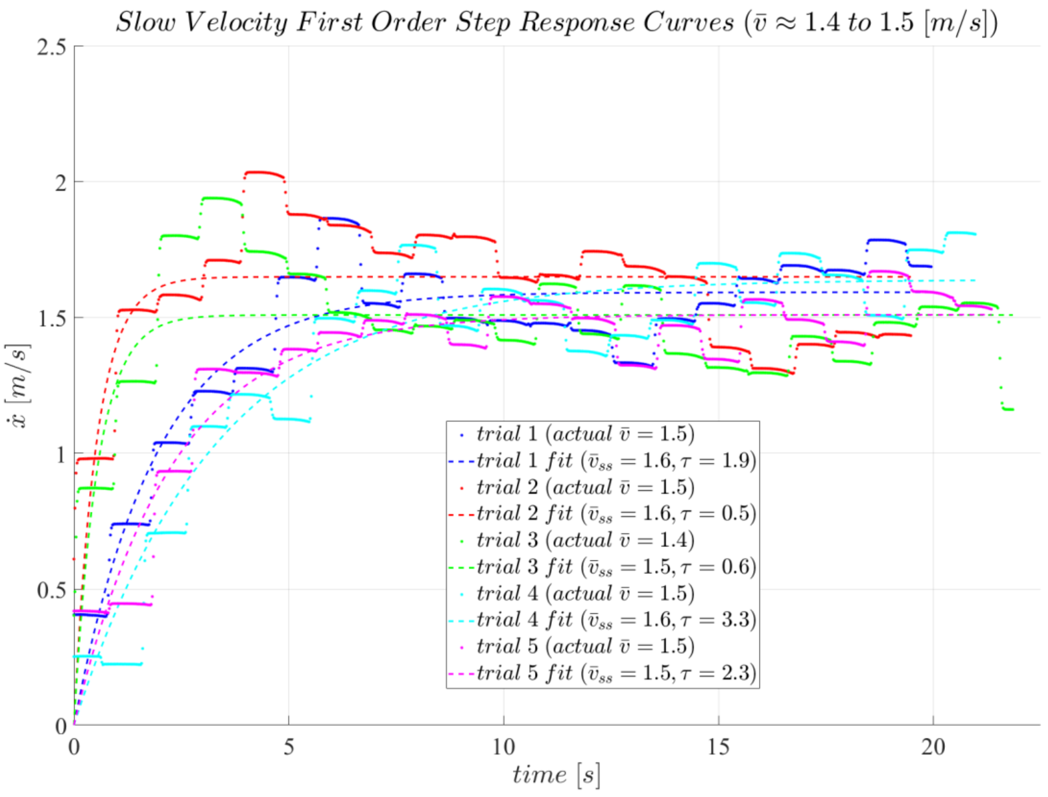

Figure 101: “Slow” (average velocity ≈ 1.4 to 1.5 [m/s]) linear velocity step response data and fitted first order step response curves for five trials. See legend for actual mean velocity $\bar{v}$, steady state velocity v_{ss}, and time constant τ.

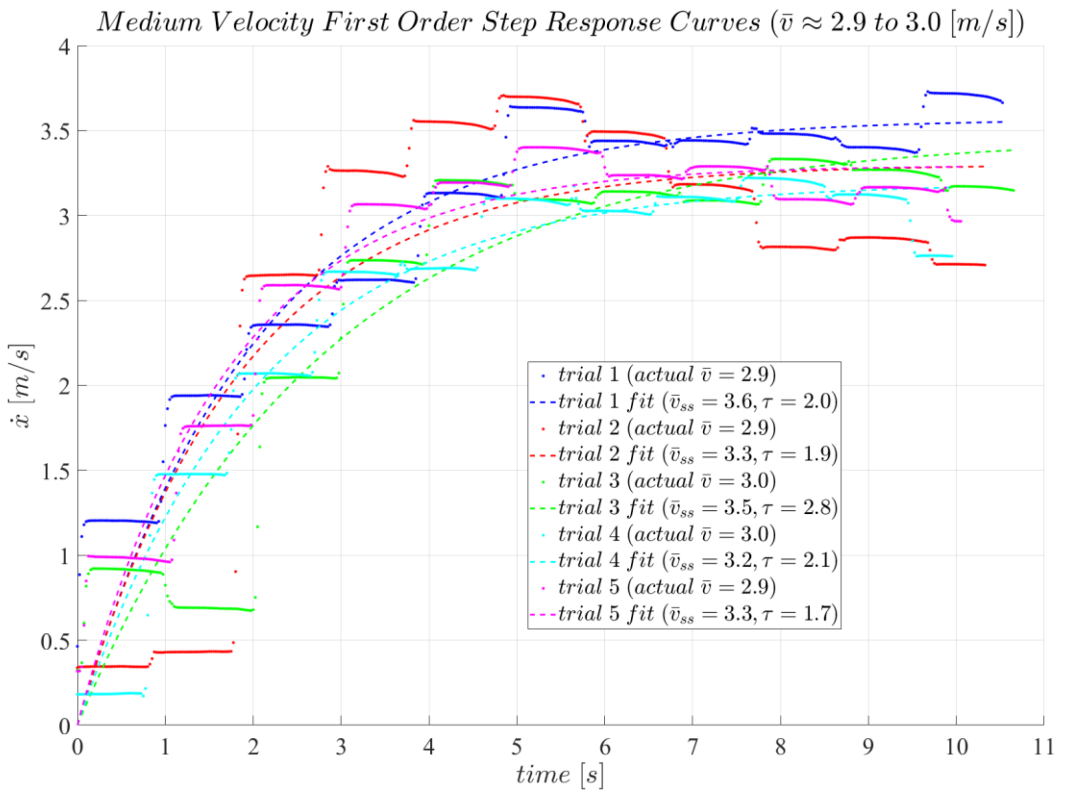

Figure 102: “Medium” (average velocity ≈ 2.9 to 3.0 [m/s]) linear velocity step response data and fitted first order step response curves for five trials. See legend for actual mean velocity $\bar{v}$, steady state velocity v_{ss}, and time constant τ.

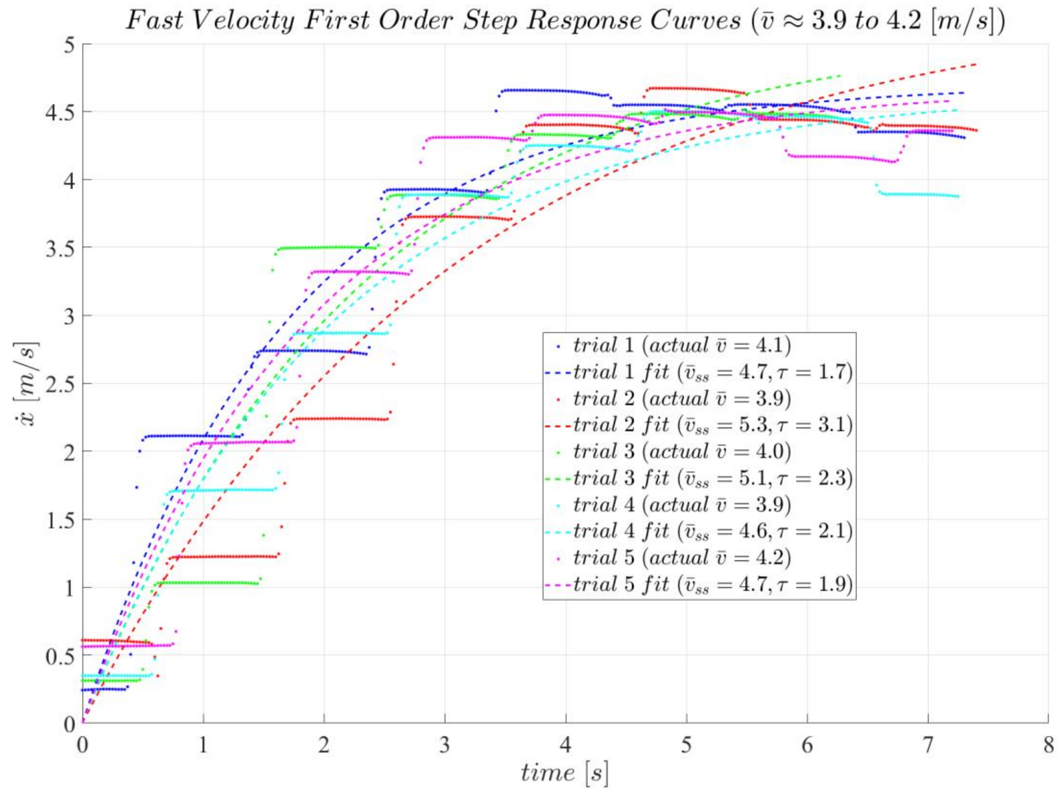

Figure 103: "Fast" (average velocity ≈ 3.9 to 4.2 [m/s]) linear velocity step response data and fitted first order step response curves for five trials. See legend for actual mean velocity $\bar{v}$, steady state velocity v_{ss}, and time constant τ.

For convenience, the results from the slow, medium, and fast linear velocity step response trials are compiled in Table 12.

Table 12: Actual Mean Velocity, Fitted Steady-State Velocity, and Fitted Time Constant for Linear Velocity Step Response Trials

Trial	Actual mean velocity $\bar{v}$	Fitted steady-state velocity v_{ss}	Estimator velocity error	Fitted time constant τ
Slow trial 1	1.5 m/s	1.6 m/s	+0.1 m/s (8.6 %)	1.9 s
Slow trial 2	1.5 m/s	1.6 m/s	+0.2 m/s (10.5 %)	0.5 s
Slow trial 3	1.4 m/s	1.5 m/s	+0.1 m/s (5.3 %)	0.6 s
Slow trial 4	1.5 m/s	1.6 m/s	+0.2 m/s (12.3 %)	3.3 s
Slow trial 5	1.5 m/s	1.5 m/s	+0.1 m/s (4,0 %)	2.3 s
Medium trial 1	2.9 m/s	3.6 m/s	+0.7 m/s (2.3 %)	2.0 s
Medium trial 2	2.9 m/s	3.3 m/s	+0.4 m/s (14.2 %)	1.9 s
Medium trial 3	3.0 m/s	3.5 m/s	+0.5 m/s (15.4 %)	2.8 s
Medium trial 4	3.0 m/s	3.2 m/s	+0.2 m/s (6.3%)	2.1 s
Medium trial 5	2.9 m/s	3.3 m/s	+0.4 m/s (14.0%)	1.7 s
Fast trial 1	4.1 m/s	4.7 m/s	+0.59 m/s (14.3%)	1.7 s

Fast trial 2	3.9 m/s	5.3 m/s	+1.4 m/s (35.9%)	3.1 s
Fast trial 3	4.0 m/s	5.1 m/s	+1.1 m/s (28.6%)	2.3 s
Fast trial 4	3.9 m/s	4.6 m/s	+0.8 m/s (20.2%)	2.1 s
Fast trial 5	4.2 m/s	4.7 m/s	+0.45 m/s (10.7%)	1.9 s

4.3. Video of estimator output on a moving vehicle

It is much easier to grasp the interactions between the time-series state variables in video form. In this section, I present a video demonstrating four different data collections of estimator operation on a moving vehicle. A screenshot of this video is shown in Figure 104.

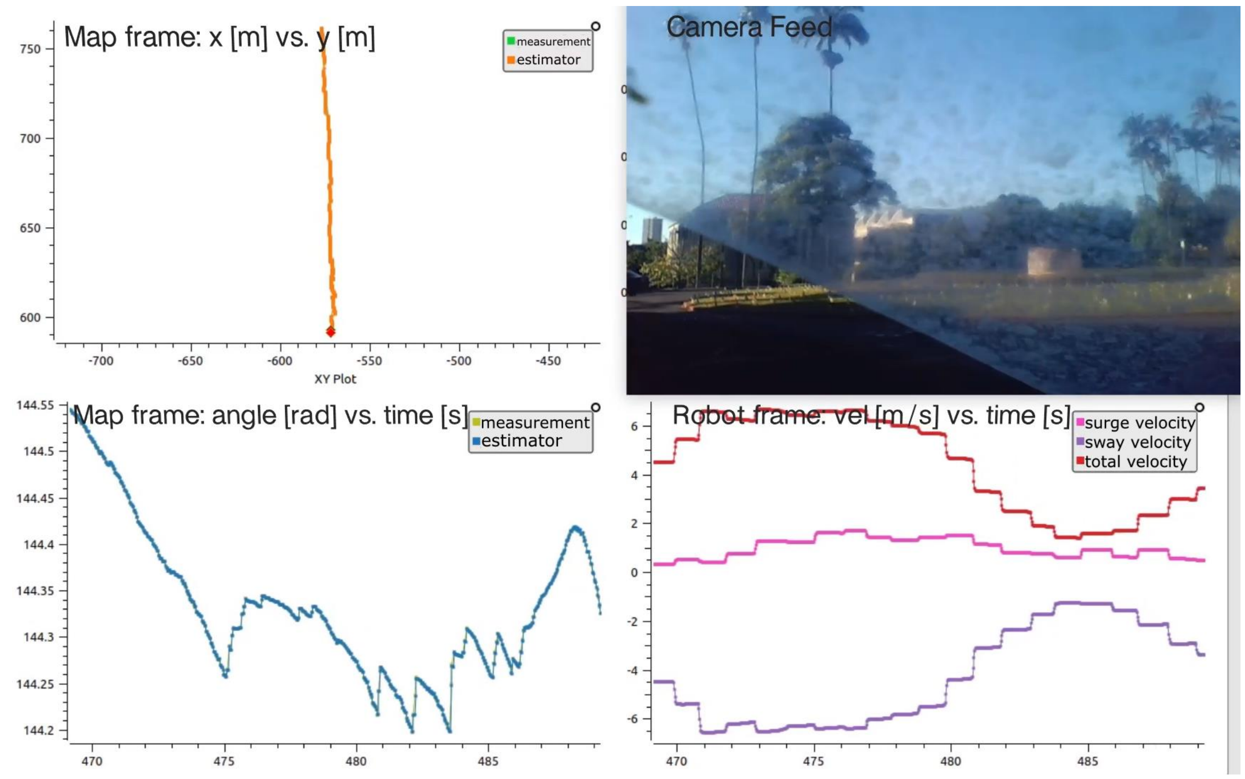

Figure 104: Screenshot of estimator operation video. This video plots the map frame x_m vs. y_m (top left), the map frame ψ_m vs. time (bottom left), a forward camera feed (top right), and the robot frame velocity $\left(\dot{x}_r, \dot{y}_r, \sqrt{\dot{x}_r^2 + \dot{y}_r^2}\right)$ vs. time.

This video is vieweable at reference [125]. Because the data collected in this video does not have an explicit ground truth, I cannot draw concrete conclusions from these particular datasets. This video content is included because viewing the time-series outputs from the estimator in video form alongside footage of robot motion exemplifies the usefulness of the estimator data. This should provide context to the well-validated estimator data presented in static plot form in the previous sections 4.1 and 4.2.

5. Conclusions and future work

5.1. Discussion of results

In section 4.1, I tested the state estimation algorithm in a robot simulation environment written in Matlab. I tested several scenarios: straight-line, forward-backward-forward, circular motion, constant surge-sway-yaw, constant surge-sway-negative-yaw, and alternating surge-sway-yaw. From these simulations I learned several key points:

- First, the estimator can generally predict the position of the robot to within the measurement accuracy of the GNSS sensor when in between GNSS updates; however, when outlier values are fed into the estimator and/or large changes in velocity occur time steps of <1s (the “refresh rate” of the forward-difference linear velocity), this prediction can vary significantly.
- Second, the robot frame linear velocity demonstrates significant steady-state error; however, this steady-state error is a function of the refresh rate of the GNSS and IMU sensors, and the intermediate processes used to extract linear velocity from these measurements. If the sensor refresh rate remains constant, the linear velocity steady-state error can be corrected using an open-loop multiplicative bias term. In our system, I found this to be 1.35. Once implemented, error in the linear velocity state is largely due to drift in the robot body frame linear position measurement conversions, and measurement variance.
- Third, increasing the number of GNSS linear position measurements used to calculate the forward-difference n decreases the variance in the linear velocity pseudo-measurement (and improves the steady-state error as discussed above), but comes at the cost of increased time constant in the response of the state estimator to changes in linear velocity. There is no “correct” answer for n, but for highly-transient mobile robots, decreasing the response time is generally more important than measurement variance, so I calculate the forward-difference using $n = 2$ (the minimum possible).

- Fourth, the sliding adaptive covariance profiling algorithm has two purposes: (1) to correctly handle sensor measurements that are taken in different discrete-time domains, and (2) to smooth the estimator outputs in real-time. The algorithm is clearly successful in (1) as the estimator properly fuses the GNSS and IMU sensor data to provide full state estimates in between sensor updates. In (2), the algorithm does smooth the outputs of the estimator in real time; however, this comes at the cost of increased response time. Again, there is no "correct" answer for the ideal degree of smoothing, but for highly transient mobile robots, decreasing response time is generally more important than measurement variance, thusly, I employ only a small amount of sliding adaptive covariance.

In section 4.2 I analyzed the performance of the estimator on real-world data. I first describe the GNSS+IMU sensor package and the experimental setup in subsection 4.2.1. In subsection 4.2.2 and 4.2.3 I test the state estimator on a characterized rectangular and circular motion path, respectively. Collecting sensor data and state estimator outputs on these characterized ground truths, I calculated the absolute error of the linear position measurements and state estimator outputs. The GNSS linear position measurements demonstrate significant geometric error, ranging from 1 to 6 m, as well as surprisingly high temporal error (latency), ranging from 1-3 s. While our results do not corroborate the advertised sensor specifications compiled on Table 1, they do corroborate the previous work by Alam et al. [52] in fusing low cost, single point prevision (SPP) GNSS sensors, where they encountered estimation error on the order of 6.4 m. In addition, the significant latency in the GNSS measurements, as previously noted in references [111]–[114], forced us to impose a delay in the IMU measurements as described in [113]. Employing a higher-accuracy GNSS sensors and/or extrinsic positing sensor systems like laser scans or computer vision to improve raw sensing performance will dramatically decrease the error of the estimator.

Because the low-cost GNSS sensor is the only linear position sensor in the system, one should not expect the state estimator to produce meaningful decreases in error relative to the ground truth. This is reinforced by the results compiled in Table 8 and Table 10, where

I found that the error of the state estimator relative to the GNSS measurements is small relative to the error of the GNSS measurements relative to the ground truth. This makes this analysis inconclusive with regards to characterizing the error of the state estimator. However, the analysis comparing the linear position estimates to the interpolated GNSS measurements in Table 9 and Table 11 show that the linear position state estimates have a mean error ranging from 0.4 to 1.2 m with a standard deviation ranging from 0.6 to 0.9 m across all trials.

In subsection 4.2.4 I compared the linear velocity state estimation data to a known linear velocity step input. I then fitted the estimation data to a first order response curve to extract the steady state velocity v_{ss} and the time constant τ. Excluding outlier values due to poor fitment quality, the linear velocity estimation error ranged from $+2\%$ to $+20\%$ (or 0.1 to 0.8 m/s). This range demonstrates a slight systematic overestimation bias. This demonstrates that the method of applying an open-loop linear velocity bias discussed in section 4.1.2 is effective for correcting the steady-state linear velocity error in the state estimator, provided that the state estimator is calibrated for the exact sensor system employed. The open-loop bias of 1.35 predicted by our simulation results is slightly too high for our real-world measurements. The time constant of the linear velocity pseudo-measurements ranged from 0.5 to 3.3 s; the time constant has a large variance partially due to it sensitivity to the timing of the step input relative to the GNSS measurements. If the step input occurs immediately proceeding a GNSS sensor update, the time constant can be less than one second; however, it if occurs immediately following a GNSS measurement, the linear velocity will not measure any change until the next GNSS measurement occurs a full second later. Therefore, this time constant can be significantly improved, if the refresh rate of the linear position sensor is increased.

In section 4.3, I applied our state estimation data to real data collected from a moving vehicle. This data is presented in video form. Because the data recorded in this video does not have an explicit ground truth, I cannot draw concrete conclusions from the video alone. This video content is included because viewing the time-series outputs from the estimator in video form alongside footage of robot motion exemplifies the usefulness of the estimator data validated in static plot form in the previous sections.

In conclusion, I have shown that our state estimation algorithm is capable of fusing GNSS and IMU sensor data into a single, cohesive state estimate without the need for a generalized linear velocity sensor. The real-world error in the low-cost GNSS sensor measurements was too high to effectively evaluate the contribution of ground truth error due to the state estimator, so I evaluated the mean error of the estimator relative to the interpolated measurements. **From this I found that the linear position state estimates have a mean error ranging from 0.4 to 1.2 m with a standard deviation ranging from 0.6 to 0.9 m across all trials.** In addition, I scrutinized the linear velocity estimation accuracy of the state estimator. **With a linear velocity open-loop bias of 1.35 determined from the simulated measurements, the linear velocity estimation error range from 0.1 to 0.8 m/s;** this systematic error demonstrates that the method of applying an open-loop linear velocity bias is effective for correcting the steady-state linear velocity error in the state estimator, provided that the state estimator is calibrated for the exact sensor system employed. The time constant of the linear velocity pseudo-measurements ranged from 0.5 to 3.3 s.

5.2. Future work

5.2.1. Redundant sensors

Using multiple GNSS sensors will not decrease the variance in the measurements, as the measurements are not fundamentally independent (they are subject to the same systematic error); however, more sensors should increase the number of readings (temporal resolution) of the sensor system. From our discussion in section 4.1.1 these addional measurements can be used to improve our response time, and reduce steady-state error.

In collecting real data, I found that the IMU AHRS/MARG orientation estimator has a strong tendency to drift in urban environment. With only a single source of orientation data, these “phantom” rotations ruin the estimator output. In this work, I have already described a solution for elegantly incorporating redundant sensor measurements, this allows us to incorporate information from other sensor systems to detect and compensate for these orientation errors. These sensors could include multiple GNSS sensors with a known separation for orientation measurement and using extrinsic sensors like robot vision and laser scans.

6. Appendices

6.1. Conversion between fixed-xyz Euler angles, rotation matrices, and angle-axis representation

Assuming a mobile robot with attached frame {B}, operating with respect to a nearby inertial map frame {A}. I can describe the position and orientation of the robot as a 6×1 state vector $(x, y, z, \phi, \theta, \psi)$. Because linear positions (x, y, z) are commutative, they are easily described by a 3×1 vector; however, angular positions are non-commutative, and therefore, it is necessary to specify the ordering of Euler angle rotations (ϕ, θ, ψ). In our work, I utilize the fixed-xyz Euler angle rotation order. In this section, I describe the conversion of Euler angles to rotation matrices, and angle-axis representation.

6.1.1. Fixed-xyz Euler angles (ϕ, θ, ψ) to rotation matrix ${}^A_BR$

Assuming fixed-xyz Euler angles (ϕ, θ, ψ) that describe the orientation of a robot body frame {B} with respect to the map frame {A}, a 3×3 rotation matrix is a common method of communicating the rotation of a coordinate system. The symbolic form of the rotation matrix will change based on the Euler angle rotation order; the rotation matrix given in equation (52) reflects the fixed-xyz rotation order (first, a rotation ϕ about x_A, then a rotation θ about y_A, and finally a rotation ψ about z_A).

$$
\begin{aligned}
&{}^{A}_{B}R_{xyz}(\phi,\theta,\psi) = {}^{A}_{B}R_{z}(\psi)\,{}^{A}_{B}R_{y}(\theta)\,{}^{A}_{B}R_{x}(\phi) \\
&= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \\
&= \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}
\end{aligned}
$$

Where (52)

${}^{A}_{B}R_{xyz}(\phi,\theta,\psi)$ is the rotation of frame {B} with respect to frame {A} for fixed-xyz Euler angles

ϕ is the fixed-xyz Euler angle of x-axis of frame {B} with respect to x-axis of frame {A}

θ is the fixed-xyz Euler angle of y-axis of frame {B} with respect to y-axis of frame {A}

ψ is the fixed-xyz Euler angle of z-axis of frame {B} with respect to z-axis of frame {A}

The inverse problem, i.e. extracting the fixed-xyz Euler angles (ϕ,θ,ψ) from a rotation matrix ${}^{A}_{B}R$, is also often of interest. The solution to this problem is given in equation (53).

$$
\begin{cases}
\phi = \operatorname{atan2}\left(\dfrac{r_{32}}{\cos\theta}, \dfrac{r_{33}}{\cos\theta}\right) = \operatorname{atan2}\left(\dfrac{\cos\theta\sin\phi}{\cos\theta}, \dfrac{\cos\theta\cos\phi}{\cos\theta}\right) = \operatorname{atan2}(\sin\phi, \cos\phi) \\
\theta = \operatorname{atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right) = \operatorname{atan2}\left(\sin\theta, \sqrt{(\cos\psi\cos\theta)^2 + (\sin\psi\cos\theta)^2}\right) \\
\psi = \operatorname{atan2}\left(\dfrac{r_{21}}{\cos\theta}, \dfrac{r_{11}}{\cos\theta}\right) = \operatorname{atan2}\left(\dfrac{\sin\psi\cos\theta}{\cos\theta}, \dfrac{\cos\psi\cos\theta}{\cos\theta}\right) = \operatorname{atan2}(\sin\psi, \cos\psi)
\end{cases}
$$

Where (53)

${}^{A}_{B}R_{\hat{K}}(\theta) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ is the rotation matrix describing the rotation of frame {B} with respect to frame {A}

ϕ is the fixed-xyz Euler angle of x-axis of frame {B} with respect to x-axis of frame {A}

θ is the fixed-xyz Euler angle of y-axis of frame {B} with respect to y-axis of frame {A}

ψ is the fixed-xyz Euler angle of z-axis of frame {B} with respect to z-axis of frame {A}

6.1.2. Rotation matrix ${}^{A}_{B}R$ to angle-axis representation $({}^{A}_{B}\hat{K}, {}^{A}_{B}\theta)$

In the equivalent angle-axis representation, an orientation between two coordinate systems can be represented as a single rotation, θ, between 0° and 180° about a general direction unit vector, $\widehat{K}$. One can convert a rotation matrix, R, to angle-axis representation using equation (54).

$$\begin{cases} {}^{A}_{B}\widehat{K} = \begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix} = \dfrac{1}{2\sin\theta} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \\ {}^{A}_{B}\theta = \operatorname{acos}\left(\dfrac{r_{11} + r_{22} + r_{33} - 1}{2}\right) \end{cases} \tag{54}$$

Where

${}^{A}_{B}\widehat{K}$ is the unit vector defining the axis of rotation of frame {B} with respect to frame {A}

${}^{A}_{B}\theta$ is the angle of rotation about ${}^{A}_{B}\widehat{K}$ of frame {B} with respect to frame {A}

${}^{A}_{B}R_{\widehat{K}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ is the rotation matrix describing the rotation of frame {B} with respect to frame {A}

The inverse problem, i.e. extracting the rotation matrix ${}^{A}_{B}R$ from the angle-axis representation $(\widehat{K}, \theta)$, is also often of interest. The solution to this problem is given in equation (55) [100].

$$\begin{aligned} &{}^{A}_{B}R_{\widehat{K}}(\theta) \\ &= \begin{bmatrix} k_x k_x(1-\cos\theta) + \cos\theta & k_x k_y(1-\cos\theta) - k_z\sin\theta & k_x k_z(1-\cos\theta) + k_y\sin\theta \\ k_x k_y(1-\cos\theta) + k_z\sin\theta & k_y k_y(1-\cos\theta) + \cos\theta & k_y k_z(1-\cos\theta) - k_x\sin\theta \\ k_x k_z(1-\cos\theta) - k_y\sin\theta & k_y k_z(1-\cos\theta) + k_x\sin\theta & k_z k_z(1-\cos\theta) + \cos\theta \end{bmatrix} \end{aligned} \tag{55}$$

Where

${}^{A}_{B}R_{\widehat{K}}$ is the rotation matrix describing the rotation of frame {B} with respect to frame {A}

${}^{A}_{B}\widehat{K} = \begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix}$ is the unit vector defining the axis of rotation of frame {B} with respect to frame {A}

${}^{A}_{B}\theta$ is the angle of rotation about $\widehat{K}$ of frame {B} with respect to frame {A}

6.2. Conversion between angular velocity ${}^{A}_{B}\vec{\Omega}$, velocity rotation matrix ${}^{A}_{B}\dot{R}$, and fixed-xyz Euler angle time derivatives $(\dot{\phi}, \dot{\theta}, \dot{\psi})$

In addition to the six position states $(x, y, z, \phi, \theta, \psi)$ that describe robot pose, I also aim to define the six velocity states that describe robot twist (linear and angular velocity). Because linear translations (and by extension, linear velocities) are commutative, they are easily expressed using a 3×1 vector $(\dot{x}, \dot{y}, \dot{z})$; however, because rotations (and by extension, angular velocities) are non-commutative, communicating the time rate of change of rotation using ordered Euler angles, i.e. $(\dot{\phi}, \dot{\theta}, \dot{\psi})$, is very unconventional. In this section, I will describe alternative methods of expressive the time rate of change of rotation; these include angular velocity ${}^{A}_{B}\vec{\Omega}$, velocity rotation matrix ${}^{A}_{B}\dot{R}$, fixed-xyz Euler angle time derivatives $(\dot{\phi}, \dot{\theta}, \dot{\psi})$, and how to convert between these representation methods.

6.2.1. Rotation matrix ${}^{A}_{B}R$ and velocity rotation matrix ${}^{A}_{B}\dot{R}$ to angular velocity ${}^{A}_{B}\vec{\Omega}$

The angular velocity of a body, ${}^{A}_{B}\vec{\Omega}$, is best described as the instantaneous axis of rotation of frame ${}^{A}_{B}\hat{K}$, times the speed of rotation ${}^{A}_{B}\dot{\theta}$. I can show that angular velocity vector ${}^{A}_{B}\vec{\Omega}$ is related to the rotation matrix ${}^{A}_{B}R$ and the derivative of the rotation matrix ${}^{A}_{B}\dot{R}$ by a skew-symmetric matrix S. This derivation is performed in equation (56).

First, note that all orthonormal matrices follow the property:

$$R\,R^T = I$$

Where

R is a 3×3 orthonormal rotation matrix

I is a 3×3 identity matrix

Taking the derivative,

$$\begin{gathered} \dot{R}\,R^T + R\,\dot{R}^T = 0 \\ \dot{R}\,R^T + \left(\dot{R}\,R^T\right)^T = 0 \end{gathered} \tag{56}$$

Substituting in a general matrix S,

$$\Rightarrow \begin{cases} S + (S)^T = 0 \\ S = \dot{R}\,R^T \end{cases}$$
$$\Rightarrow \begin{cases} S + S^T = 0 \\ \dot{R} = \dfrac{S}{R^T} \end{cases}$$
$$\Rightarrow \begin{cases} S + S^T = 0 \\ \dot{R} = S\,R \end{cases}$$

There are two important takeaways from equation (56). First, that the derivative of a general rotation matrix, $\dot{R}$, can be written as a product of the matrix S and the original rotation matrix R. Second, the matrix S follows the form, $S + S^T = 0$, which is a property exclusive to skew-symmetric matrices, which follow the form given in equation (57).

$$S = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{57}$$

Combining equations (56) and (57) allows us to relate the derivative of a rotation matrix, R, to the product of a skew symmetric matrix, S, and the rotation matrix, R. This skew symmetric matrix represents the angular velocity components of a rotating system. This is given in equation (58).

$$\dot{R} = \begin{bmatrix} 0 & -\Omega_z & \Omega_y \\ \Omega_z & 0 & -\Omega_x \\ -\Omega_y & \Omega_x & 0 \end{bmatrix} R$$

Where (58)

R is a 3×3 orthonormal rotation matrix

$\dot{R}$ is the time derivative of R

$\vec{\Omega}$ is the angular velocity vector

Because the skew symmetric matrix S, always follows the form given in equation (57), the angular velocity can be described using a 3×1 vector, $\vec{\Omega}$, which is defined in equation (59) to describe the angular velocity of frame {B} with respect to frame {A}.

$$^{A}_{B}\vec{\Omega} = \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} = \begin{bmatrix} k_x\dot{\theta} \\ k_y\dot{\theta} \\ k_z\dot{\theta} \end{bmatrix} = {}^{A}_{B}\widehat{K}\,{}^{A}_{B}\dot{\theta}$$

where (59)

$^{A}_{B}\vec{\Omega}$ is the 3×1 angular velocity vector of frame {B} with respect to frame {A}

$^{A}_{B}\widehat{K} = \begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix}$ is the unit vector defining the axis of rotation of frame {B} with respect to frame {A}

$\dot{\theta}$ is the rotation rate about $\widehat{K}$ of frame {B} with respect to frame {A}

Note that this process is only useful for finding the angular velocity vector $^{A}_{B}\vec{\Omega}$ when given the rotation matrix $^{A}_{B}R$ *and* the velocity rotation matrix $^{A}_{B}\dot{R}$, which is unlikely to happen in practice; however, the process in equations (56) through (59) is crucial for the derivation of the conversion between other time rate of change of rotation representations discussed in the next section.

6.2.2. Fixed-xyz Euler angle time derivatives $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ to angular velocity $^{A}_{B}\vec{\Omega}$

The fixed-xyz Euler angle time derivatives can be converted to angular velocity using the framework outlined in equations (56) through (59). By symbolically substituting the fixed-xyz Euler angle rotation matrices into $^{A}_{B}R$ and $^{A}_{B}\dot{R}$, I can derive the symbolic conversion between fixed-xyz Euler angle time derivatives $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ to angular velocity ${}^{A}_{B}\vec{\Omega}$. The symbolic form of the rotation matrix for fixed-xyz Euler angles is given in equation (52), and the symbolic form of the velocity rotation matrix for fixed-xyz Euler angles is given in equation (60).

$${}^{A}_{B}\dot{R}_{xyz}(\phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$\begin{cases} r_{11} = -\dot{\psi}\cos\theta\sin\psi - \dot{\theta}\cos\psi\sin\theta \\ r_{12} = \dot{\phi}(\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta) - \dot{\psi}(\cos\phi\cos\psi + \sin\phi\sin\psi\sin\theta) + \dot{\theta}\cos\psi\cos\theta\sin\phi \\ r_{13} = \dot{\phi}(\cos\phi\sin\psi - \cos\psi\sin\phi\sin\theta + \dot{\psi}(\cos\psi\sin\phi - \cos\phi\sin\psi\sin\theta) + \dot{\theta}\cos\phi\cos\psi\cos\theta \\ r_{21} = \dot{\psi}\cos\psi\cos\theta - \dot{\theta}\sin\psi\sin\theta \\ r_{22} = \dot{\theta}\cos\theta\sin\phi\sin\psi - \dot{\psi}(\cos\phi\sin\psi - \cos\psi\sin\phi\sin\theta) - \dot{\phi}(\cos\psi\sin\phi - \cos\phi\sin\psi\sin\theta) \\ r_{23} = \dot{\psi}(\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta) - \dot{\phi}(\cos\phi\cos\psi + \sin\phi\sin\psi\sin\theta) + \dot{\theta}\cos\phi\cos\theta\sin\psi \\ r_{31} = -\dot{\theta}\cos\theta \\ r_{32} = \dot{\phi}\cos\phi\cos\theta - \dot{\theta}\sin\phi\sin\theta \\ r_{33} = -\dot{\phi}\cos\theta\sin\phi - \dot{\theta}\cos\phi\sin\theta \end{cases}$$

Where (60)

${}^{A}_{B}\dot{R}_{xyz}(\phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi})$ is the velocity rotation matrix of frame {B} with respect to frame {A} for fixed-xyz Euler angles and fixed-xyz Euler angles time derivative

ϕ is the fixed-xyz Euler angle of x-axis of frame {B} with respect to x-axis of frame {A}

θ is the fixed-xyz Euler angle of y-axis of frame {B} with respect to y-axis of frame {A}

ψ is the fixed-xyz Euler angle of z-axis of frame {B} with respect to z-axis of frame {A}

$\dot{\phi} \approx \frac{\phi}{\Delta t}$ is the time derivative of ϕ

$\dot{\theta} \approx \frac{\theta}{\Delta t}$ is the time derivative of θ

$\dot{\psi} \approx \frac{\psi}{\Delta t}$ is the time derivative of $\dot{\psi}$

By combining equations (52), (58), and (60), I can solve for the symbolic form of the skew-symmetric matrix for fixed-xyz Euler angles. This is given in equation (61).

$$S = \begin{bmatrix} 0 & -(\dot{\psi} - \dot{\phi}\sin\theta) & \dot{\theta}\cos\psi + \dot{\phi}\cos\theta\sin\psi \\ \dot{\psi} - \dot{\phi}\sin\theta & 0 & -(\dot{\phi}\cos\psi\cos\theta - \dot{\theta}\sin\psi) \\ -(\dot{\theta}\cos\psi + \dot{\phi}\cos\theta\sin\psi) & \dot{\phi}\cos\psi\cos\theta - \dot{\theta}\sin\psi & 0 \end{bmatrix}$$

Where
- ϕ is the fixed-xyz Euler angle of x-axis of frame {B} with respect to x-axis of frame {A}
- θ is the fixed-xyz Euler angle of y-axis of frame {B} with respect to y-axis of frame {A} (61)
- ψ is the fixed-xyz Euler angle of z-axis of frame {B} with respect to z-axis of frame {A}
- $\dot{\phi} \approx \frac{\phi}{\Delta t}$ is the time derivative of ϕ
- $\dot{\theta} \approx \frac{\theta}{\Delta t}$ is the time derivative of θ
- $\dot{\psi} \approx \frac{\psi}{\Delta t}$ is the time derivative of $\dot{\psi}$

Combining equations (57), (58), (59), and (61), I can solve for the symbolic form of the angular velocity vector ${}^{A}_{B}\vec{\Omega}$ for fixed-xyz Euler angles. This is given in equation (62).

$${}^{A}_{B}\vec{\Omega} = \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} = \begin{bmatrix} \dot{\phi}\cos\psi\cos\theta - \dot{\theta}\sin\psi \\ \dot{\theta}\cos\psi + \dot{\phi}\cos\theta\sin\psi \\ \dot{\psi} - \dot{\phi}\sin\theta \end{bmatrix}$$

Where
- ${}^{A}_{B}\vec{\Omega}$ is the 3×1 angular velocity vector of frame {B} with respect to frame {A}
- ϕ is the fixed-xyz Euler angle of x-axis of frame {B} with respect to x-axis of frame {A}
- θ is the fixed-xyz Euler angle of y-axis of frame {B} with respect to y-axis of frame {A} (62)
- ψ is the fixed-xyz Euler angle of z-axis of frame {B} with respect to z-axis of frame {A}
- $\dot{\phi} \approx \frac{\phi}{\Delta t}$ is the time derivative of ϕ
- $\dot{\theta} \approx \frac{\theta}{\Delta t}$ is the time derivative of θ
- $\dot{\psi} \approx \frac{\psi}{\Delta t}$ is the time derivative of $\dot{\psi}$

Note that equations (60) through (62) are only valid for fixed-xyz Euler angles. For other Euler angle orderings, this process remains identical, but the starting rotation matrix back in equation (52) must be symbolically re-calculated. Because there are 24 different possible rotation matrix orders, listing them all here would be too cumbersome. ROS REP 103 [99] recommends using the fixed-xyz Euler angle ordering; however, if using a different Euler angle ordering in your system, I have written a Matlab script that symbolically calculates the three-dimensional rotation matrix ${}^{A}_{B}R$, the derivative of the three-dimensional rotation matrix (velocity rotation matrix) ${}^{A}_{B}\dot{R}$, the skew-symmetric matrix relating angular velocity vector to the rotation matrix, its derivative S, and angular velocity vector ${}^{A}_{B}\Omega$ for all 24 possible Euler angle orderings. It is available at [126].

6.3. Variance of the forward-difference method for discrete differentiation of GNSS data

Recall the equation for the forward-difference; this was given in equation (33) in the text. It is listed in equation (59) below.

$$f'(x) \approx \frac{x_{k+1} - x_k}{dt} \tag{63}$$

To calculate the variance of $f'(x)$, I need two rules, the variance sum law, which tells us how to add two variances, and the variance constant product law, which tells us how to multiply a constant by a variance. They are given in equation (64) and equation (65), respectively.

$$\begin{cases} var(x + y) = var(x) + var(y) \\ var(x - y) = var(x) + var(y) \end{cases} \tag{64}$$

$$var(c \cdot x) = c^2 \cdot var(x) \tag{65}$$

Rewriting equation (59) as sums and products gives equation (62).

$$\begin{aligned} f'(x) &\approx \frac{x_{k+1} - x_k}{dt} \\ &\approx \left(\frac{1}{dt}\right)(x_{k+1}) - \left(\frac{1}{dt}\right)(x_k) \end{aligned} \tag{66}$$

The variance of $f'(x)$ is calculated in equation (67).

$$\begin{aligned} var\big(f'(x)\big) &= \left(\frac{1}{dt}\right)^2 \cdot var(x_{k+1}) - \left(\frac{1}{dt}\right)^2 \cdot var\big(\sigma_{x_k}^2\big) \\ &= \frac{1}{dt^2} \cdot var\big(\sigma_{x_{k+1}}^2 - \sigma_{x_k}^2\big) \end{aligned}$$

Assuming $var\big(\sigma_{x_{k+1}}^2\big) \approx var(\sigma_{x_k}^2)$, (67)

$$\begin{aligned} var\big(f'(x)\big) &\approx \frac{1}{dt^2} \cdot var\big(2 \cdot \sigma_{x_k}^2\big) \\ &\approx \frac{\sqrt{2}}{dt^2} \cdot var\big(\sigma_{x_k}^2\big) \end{aligned}$$

Noting that the *subtraction* of two variances sums in the same fashion as the addition of two variances, equation (67) tells us that the variance of the finite difference equation is sensitive to the reciprocal of the time step, squared, dt^2. A typical, low-cost GNSS sensor has an update rate of 1 Hz, or $dt = 1$ time step (see Table 1); this results in a the forward-difference formula multiplying the variance of the linear position measurement $var\big(\sigma_{x_k}^2\big)$ by a constant of $\frac{\sqrt{2}}{1^2} \approx 1.414$. This tells us that the forward-difference operation increases the variance of the linear position measurement by a factor of 1.414, provided that the dimensional analysis is correctly handled. Increasing the number of measurements increases the time step, which stands to correct this increase in variance further. For example, if two measurements taken at 1s apart this same coefficient changes to $\frac{\sqrt{2}}{2^2} \approx 0.354$, which is a significant decrease in variance (a good thing).

6.4. Variance of the trapezoidal method for discrete integration for accelerometer data

Recall the equation for the trapezoidal method; this was given in equations (38) and (39) in the text. It is given as a more general case in equation (68) below.

$$\int_{x_k}^{x_{k+1}} f(\ddot{x})\,dx \approx \dot{x}_{k-1} + \frac{\ddot{x}_{i-1} + \ddot{x}_i}{2} dt \tag{68}$$

Similar to the variance calculation for the forward difference formula, calculating the variance of $f'(x)$ requires the, the variance sum law, and the variance constant product law, which are given in equations (64) and (65), respectively. Rewriting equation (68) as sums and products gives equation (69).

$$\begin{aligned} \int_{x_k}^{x_{k+1}} f(\ddot{x})\,dx &\approx \dot{x}_{k-1} + \frac{\ddot{x}_{i-1} + \ddot{x}_i}{2} dt \\ &\approx \dot{x}_{k-1} + \left(\frac{dt}{2}\right) x_k + \left(\frac{dt}{2}\right) x_{k+1} \end{aligned} \tag{69}$$

The variance of $\int_{x_k}^{x_{k+1}} f(x)\,dx$ is calculated in equation (70).

$$\begin{aligned} var\left(\int_{x_k}^{x_{k+1}} f(\ddot{x})\,dx\right) &\approx \left(\frac{dt}{2}\right)^2 \cdot var(x_k) + \left(\frac{dt}{2}\right)^2 \cdot var(x_{k+1}) \\ &\approx var(\dot{x}_{k-1}) + \frac{dt^2}{4} \cdot var\left(\sigma_{x_k}^2 + \sigma_{x_{k+1}}^2\right) \\ \text{Assuming } var\left(\sigma_{x_k}^2\right) &\approx var\left(\sigma_{x_{k+1}}^2\right), \\ var\left(\int_{x_k}^{x_{k+1}} f(\ddot{x})\,dx\right) &\approx var(\dot{x}_{k-1}) + \frac{dt^2}{4} \cdot var\left(2 \cdot \sigma_{x_k}^2\right) \\ &\approx var(\dot{x}_{k-1}) + \frac{\sqrt{2} \cdot dt^2}{4} \cdot var\left(\sigma_{x_k}^2\right) \end{aligned} \tag{70}$$

Equation (70) tells us that the variance is summed with the variance of the existing linear velocity measurement times a coefficient related to the sensitive to the time step squared, dt^2. For an IMU refresh rate of 200 Hz or $dt = \frac{1}{200} = 0.005\ s$. Therefore, the variance of the trapezoidal method operation is $var(\dot{x}_{k-1}) + \frac{\sqrt{2}\cdot(0.05)^2}{4} \approx var(\dot{x}_{k-1}) + 8.84E\ {-6}$, i.e. the trapezoidal method *sums* an insignificantly small amount of variance to the variance of the previous linear velocity variance. This is stated in equation (71).

$$var\left(\int_{x_k}^{x_{k+1}} f(\ddot{x})\,dx\right) \approx var(\dot{x}_{k-1}) \tag{71}$$

For $dt \ll 1$.

For this reason, taking the mean of sucessive acceleration measurements will not meaningfully improve the variance of the trapezoidal method.

6.5. Robot simulation based on the Euler numerical method

In this simulation, I model a robot as a point mass in 2D space. The forces that act on this point mass ae given in equation (71).

m is the mass of the robot $[kg]$
I is the rotational moment of inertia of robot $[kg \cdot m^2]$
b_x is the linear viscous damping along the robot frame x-axis (surge) $\left[N \cdot \frac{s}{m}\right]$
b_y is the linear viscous damping along the robot frame y-axis (sway) $\left[N \cdot \frac{s}{m}\right]$
b_ψ is the rotational viscous damping about the robot frame z-axis (yaw) $\left[N \cdot \frac{s}{m}\right]$ (72)
F_x is the force input along the robot frame x-axis (surge) $[N]$
F_y is the force input along the robot frame y-axis (sway) $[N]$
M_ψ is the moment input about the robot frame z-axis (yaw) $[N \cdot m]$

This two-dimensional simulation has three state variables (x, y, ψ) for position, velocity, and acceleration for a total of nine state variables. The state vectors in the map and robot body frames are given in equation (73).

$$\vec{x}_{map} = \begin{Bmatrix} x_m \\ y_m \\ \psi_m \\ \dot{x}_m \\ \dot{y}_m \\ \dot{\psi}_m \\ \ddot{x}_m \\ \ddot{y}_m \\ \ddot{\psi}_{z_m} \end{Bmatrix}, \quad \vec{x}_{robot} = \begin{Bmatrix} x_r \\ y_r \\ \psi_r \\ \dot{x}_r \\ \dot{y}_r \\ \dot{\psi}_r \\ \ddot{x}_r \\ \ddot{y}_r \\ \ddot{\psi}_r \end{Bmatrix} \tag{73}$$

The numerical integration can be performed in either of the two reference frames; however, because the linear viscous damping terms vary based the robot body frame coordinate system, it is more mathematically convenient to calculate the simulation in the robot body frame. Our force balance equations in the robot frame are given in equation (74).

$$\begin{gathered}
\begin{cases} \Sigma F_{x_r} = 0 \\ \Sigma_{F_{y_r}} = 0 \\ \Sigma M_{\psi_r} = 0 \end{cases} \\
\Rightarrow \begin{cases} -F_{mass} - F_{drag_x} + F_{x_r} = 0 \\ -F_{mass} - F_{drag_y} + F_{y_r} = 0 \\ -M_{RoI} - M_{drag_\psi} + M_{\psi_r} = 0 \end{cases} \\
\Rightarrow \begin{cases} -m\ddot{x}_r - b_{x_r}\dot{x}_r + F_{x_r} = 0 \\ -m\ddot{y}_r - b_{y_r}\dot{y}_r + F_{y_r} = 0 \\ -I\ddot{\psi}_r - b_{\psi_r}\dot{\psi}_r + M_{\psi_r} = 0 \end{cases} \\
\Rightarrow \begin{cases} m\ddot{x}_r = -b_{x_r}\dot{x}_r + F_{x_r} \\ m\ddot{y}_r = -b_{y_r}\dot{y}_r + F_{y_r} \\ I\ddot{\psi}_r = -b_{\psi_r}\dot{\psi}_r + M_{\psi_r} \end{cases} \\
\Rightarrow \begin{cases} \ddot{x}_r = -\dfrac{b_{x_r}\dot{x}_r}{m} + \dfrac{F_{x_r}}{m} \\ \ddot{y}_r = -\dfrac{b_{y_r}\dot{y}_r}{m} + \dfrac{F_{y_r}}{m} \\ \ddot{\psi}_r = -\dfrac{b_{\psi_r}\dot{\psi}_r}{I} + \dfrac{M_{\psi_r}}{I} \end{cases}
\end{gathered} \tag{74}$$

The system of equations in equation (74) can be rewritten as a set of coupled, first order ordinary differential equations. This is give in equation (75).

$$\begin{cases} x_1 = \dot{x}_r \\ x_2 = \dot{y}_r \\ x_3 = \dot{\psi}_r \\ x_4 = \dot{x}_1 = \ddot{x}_r = -\dfrac{b_{x_r}\dot{x}_r}{m} + \dfrac{F_{x_r}}{m} \\ x_5 = \dot{x}_2 = \ddot{y}_r = -\dfrac{b_{y_r}\dot{y}_r}{m} + \dfrac{F_{y_r}}{m} \\ x_6 = \dot{x}_3 = \ddot{\psi}_r = -\dfrac{b_{\psi_r}\dot{\psi}_r}{I} + \dfrac{M_{\psi_r}}{I} \end{cases} \tag{75}$$

Equation (75) is written in matrix form in equation (76).

$$\begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\psi}_r \\ \ddot{x}_r \\ \ddot{y}_r \\ \ddot{\psi}_r \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\dfrac{b_{x_r}}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\dfrac{b_{y_r}}{m} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\dfrac{b_{\psi_r}}{I} \end{bmatrix} \cdot \begin{bmatrix} x_r \\ y_r \\ \psi_r \\ \dot{x}_r \\ \dot{y}_r \\ \dot{\psi}_r \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \dfrac{1}{m} & 0 & 0 \\ 0 & \dfrac{1}{m} & 0 \\ 0 & 0 & \dfrac{1}{I} \end{bmatrix} \cdot \begin{bmatrix} F_{x_r} \\ F_{y_r} \\ M_{\psi_r} \end{bmatrix} \tag{76}$$

Equation (76) is written in canconical state-space form, which follows the form given in equation (77).

$$\begin{cases} \dot{\vec{x}}(t) = A(t) \cdot \vec{x}(t) + B(t) \cdot \vec{u}(t) \\ y(t) = C(t) \cdot \vec{x}(t) + D(t) \cdot \vec{u}(t) \end{cases}$$

Where
$\vec{x}(t)$ is the state vector
$\vec{u}(t)$ is the input vector
$\vec{y}(t)$ is the output vector
$A(t)$ is the state matrix
$B(t)$ is the input matrix
$C(t)$ is the output matrix
$D(t)$ is the feedthrough matrix

(77)

Equation (77) is written in discrete-time form in equation (78).

$$\begin{cases} \dot{\vec{x}}_{k+1} = A_k \cdot \vec{x}_k + B_k \cdot \vec{u}_k \\ \vec{y}_k = C_k \vec{x}_k + D_k \cdot \vec{u}_k \end{cases}$$

Where
$\vec{x}_k$ is the state vector
$\vec{u}_{\mathrm{k}}$ is the input vector
$\vec{y}_{\mathrm{k}}$ is the output vector
A_k is the state matrix
B_k is the input matrix
C_k is the output matrix
D_k is the feedthrough matrix

(78)

There are a variety of methods for solving a set of ordinary differential equations, in this case I use the forward Euler method. The forward Euler method is defined by equation (79).

$$\vec{x}_{k+1} = \vec{x}_k + \dot{\vec{x}}_k \cdot dt \tag{79}$$

Substituting in the discrete-time canonical state-space form from equation (78) into equation (79) allows us to conveniently write the discrete Euler method in matrix form. This is given in equation (80).

$$\vec{x}_{k+1} = \vec{x}_k + \dot{\vec{x}}_k \cdot dt$$

$$\begin{aligned} &\Rightarrow \vec{x}_{k+1} = \vec{x}_k + [A_k \cdot x_k + B_k \cdot \vec{u}_k] \cdot dt \\ &\Rightarrow \vec{x}_{k+1} = [I + A_k \cdot dt] \cdot \vec{x}_k + B_k \cdot \vec{u}_k \cdot dt \\ &\Rightarrow \vec{x}_{k+1} = F \cdot \vec{x}_k + G \cdot \vec{u}_k \end{aligned} \tag{80}$$

Thus, our new canonical state-space representation with the forward Euler method of numerical integration is given in equation (81).

$$\begin{cases} \dot{\vec{x}}_{k+1} = A_k \cdot \vec{x}_k + B_k \cdot \vec{u}_k \\ \vec{y}_k = C_k \vec{x}_k + D_k \cdot \vec{u}_k \end{cases}$$

Where
$\vec{x}_k$ is the state vector
$\vec{u}_k$ is the input vector
$\vec{y}_k$ is the output vector (81)
F_k is the state matrix, $F_k = I + A_k \cdot dt$
G_k is the input matrix, $G_k = B \cdot dt$
H_k is the output matrix, $H_k = C_k$
J_k is the feedthrough matrix, $J_k = D_k$
I is the identity matrix
dt is the discrete time step

Equation (81) will numerically evaluate the integral to equation (76), giving us the three position state and three velocity states. Because this simulation is intended to fuse sensor data from the position, velocity, and acceleration states, I also want to simulate the acceleration states of the robot. This information is actually provided in the bottom three rows of equation (76) before the Euler integration is performed. One convenient method of extracting this information is through the $\vec{y}_k$ component of equation (81). Substituting equation (76) into equation (81) yields the full nine state Euler integration for a single discrete-time step. This is given in equation (82).

$$\begin{cases} \vec{x}_{k+1} = F_k \cdot \vec{x}_k + G_k \cdot \vec{u}_k \\ \vec{y}_k = H_k \cdot \vec{x}_k + J_k \cdot \vec{u}_k \end{cases}$$

$$\Rightarrow \begin{cases} \begin{bmatrix} x_r \\ y_r \\ \psi_r \\ \dot{x}_r \\ \dot{y}_r \\ \dot{\psi}_r \end{bmatrix}_{k+1} = \left[\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{b_{x_r}}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{b_{y_r}}{m} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{b_{\psi_r}}{I} \end{bmatrix} \cdot dt \right] \cdot \begin{bmatrix} x_r \\ y_r \\ \psi_r \\ \dot{x}_r \\ \dot{y}_r \\ \dot{\psi}_r \end{bmatrix}_k + \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & \frac{1}{I} \end{bmatrix} \cdot dt \right] \cdot \begin{bmatrix} F_{x_r} \\ F_{y_r} \\ M_{\psi_r} \end{bmatrix} \\ \begin{bmatrix} x_r \\ y_r \\ \psi_r \\ \dot{x}_r \\ \dot{y}_r \\ \dot{\psi}_r \\ \ddot{x}_r \\ \ddot{y}_r \\ \ddot{\psi}_r \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{b_{x_r}}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{b_{y_r}}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{b_{\psi_r}}{I} & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_r \\ y_r \\ \psi_r \\ \dot{x}_r \\ \dot{y}_r \\ \dot{\psi}_r \\ \ddot{x}_r \\ \ddot{y}_r \\ \ddot{\psi}_r \end{bmatrix}_k + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & \frac{1}{I} \end{bmatrix} \cdot \begin{bmatrix} F_{x_r} \\ F_{y_r} \\ M_{\psi_r} \end{bmatrix} \end{cases} \tag{82}$$

To convert the pose from the robot body frame to the map frame, use the process outlined in section 3.1.5. This gives position, velocity, and acceleration state information in the robot body frame, and position state information in the map frame. Noise can be added to these states to simulate sensor data.

7. References

[1] F. T. Bruss, “250 years of ‘An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F.R.S. communicated by Mr. Price, in a letter to John Canton, A.M.F.R.S.,’” *Jahresbericht der Dtsch. Math.*, vol. 115, no. 3–4, pp. 129–133, 2014.

[2] A. O’Hagan and B. Luce, “A primer on Bayesian Statistics in Helath Economics and Outcomes Research,” *Scholarpedia*, vol. 4, no. 8, p. 5230, 2003.

[3] H. B. Mitchell, *Multi-sensor data fusion: an introduction*. Springer Science & Business Media, 2007.

[4] R. Boudjemaa and A. B. Forbes, “Parameter estimation methods for data fusion,” *Gt. Britain, Cent. Math. Sci. Comput.*, pp. 1–45, 2004.

[5] H. F. Durrant-Whyte, “Sensor Models and Multisensor Integration,” in *The International Journal of Robotics Research*, vol. 7, no. 6, Springer, 1988, pp. 97–113.

[6] S. Thrun, *Probabilistic robotics*, vol. 45, no. 3. MIT press, 2002.

[7] S. Roweis and Z. Ghahramani, “A unifying review of linear gaussian models,” *Neural Comput.*, vol. 11, no. 2, pp. 305–345, 1999.

[8] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *J. Basic Eng.*, vol. 82, no. 1, p. 35, 2011.

[9] S. I. Roumeliotis and G. A. Bekey, “Bayesian estimation and Kalman filtering: a unified framework for mobile robot localization,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2000, vol. 3, pp. 2985–2992.

[10] L. Ljung, “Asymptotic behaviour of extended Kalman filter as a parameter estimator for linear systems,” *IEEE Trans. Auto. Control*, vol. AC-24, no. 1, pp. 36–50, 1979.

[11] M. Hoshiya and E. Saito, “Structural identification by extended Kalman filter,” *J. Eng. Mech.*, vol. 110, no. 12, pp. 1757–1770, 1984.

[12] E. A. Wan and R. Van Der Merwe, “The unscented Kalman filter for nonlinear estimation,” in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, 2002, no. 3, pp. 153–158.

[13] S. J. Julier and J. K. Uhlmann, “New extension of the Kalman filter to nonlinear systems,”

in *Signal Processing, Sensor Fusion, and Target Recognition VI*, 2005, vol. 3068, p. 182.

[14] A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien, "Acting under uncertainty: discrete Bayesian models for mobile-robot navigation," in *IEEE International Conference on Intelligent Robots and Systems*, 1996, vol. 2, pp. 963–972.

[15] W. Burgard, A. Den Dieter, and F. Armin, "Integrating Global Position Estimation and Position Tracking for Mobile Robots : The Dynamic Markov Localization Approach," no. October, pp. 730–735, 1998.

[16] R. Simmons and S. Koenig, "Probabilistic Robot Navigation in Partially Observable Environments," in *IJCAI*, 1995, vol. 95, pp. 1080–1087.

[17] D. Fox, W. Burgard, S. Thrun, and A. B. Cremers, "Position estimation for mobile robots in dynamic environments," *Proc. Natl. Conf. Artif. Intell.*, pp. 983–988, 1998.

[18] D. Fox, "Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation," Citeseer, 2008.

[19] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEE proceedings F (radar and signal processing)*, 1993, vol. 140, no. 2, pp. 107–113.

[20] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A monte carlo algorithm for multi-robot localization," 1999.

[21] L. Chen, P. Sun, G. Zhang, J. Niu, and X. Zhang, "Fast monte carlo localization for mobile robot," in *Communications in Computer and Information Science*, 2011, vol. 144 CCIS, no. PART 2, pp. 207–211.

[22] M. Reini, "Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models," *J. Comput. Graph. Stat.*, vol. 5, no. 1, pp. 1–25, 1996.

[23] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *ICRA*, 1999, vol. 2, pp. 1322–1328.

[24] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: efficient position estimation for mobile robots," *Proc. Natl. Conf. Artif. Intell.*, vol. 1999, no. 343–349, pp. 343–349, 1999.

[25] ArduPilotDevTeam, "ArduPilot Open Source Autopilot," *Ardupilot.org*, 2018. [Online]. Available: http://ardupilot.org/. [Accessed: 25-Mar-2019].

[26] ArduPilotDevTeam, “Ardupilot Copter,” *ardupilot.org*, 2019. [Online]. Available: http://ardupilot.org/copter/. [Accessed: 26-Mar-2019].

[27] ArduPilotDevTeam, “ArduPilot Plane,” *ardupilot.org*, 2019. [Online]. Available: http://ardupilot.org/plane/. [Accessed: 26-Mar-2019].

[28] ArduPilotDevTeam, “Ardupilot Rover,” *ardupilot.org*, 2019. [Online]. Available: http://ardupilot.org/rover/. [Accessed: 26-Mar-2019].

[29] A. Tridgell *et al.*, “ArduPilot,” *GitHub repository*, 2019. [Online]. Available: https://github.com/ArduPilot/ardupilot. [Accessed: 28-Mar-2019].

[30] ArduPilotDevTeam, “ArduPilot Development Site,” *Ardupilot.org*, 2019. [Online]. Available: http://ardupilot.org/dev/index.html. [Accessed: 27-Mar-2019].

[31] ArduPilotDevTeam, “ArduPilot Extended Kalman Filter Navigation Overview and Tuning,” *Ardupilot.org*, 2019. [Online]. Available: http://ardupilot.org/dev/docs/extended-kalman-filter.html. [Accessed: 28-Mar-2019].

[32] ArduPilotDevTeam, “ArduPilot EKF2 Estimation System,” *Ardupilot.org*, 2019. [Online]. Available: http://ardupilot.org/dev/docs/ekf2-estimation-system.html. [Accessed: 28-Mar-2019].

[33] A. Tridgell *et al.*, “ArduPilot Inertial Navigation State Derivations,” *GitHub Repos.*, 2015.

[34] “NavLab - a Generic Simulation and Post-processing Tool for Navigation,” *Navlab.net*, 2019. [Online]. Available: https://www.navlab.net/. [Accessed: 27-Mar-2019].

[35] K. Gade, “NavLab, a generic simulation and post-processing tool for navigation,” *Model. Identif. Control*, vol. 26, no. 3, pp. 135–150, 2005.

[36] M. Stanley, “Open-Source-Sensor-Fusion,” *GitHub repository*, 2014. [Online]. Available: https://github.com/memsindustrygroup/Open-Source-Sensor-Fusion/wiki. [Accessed: 26-Mar-2019].

[37] Memsindustrygroup, “Open-Source-Sensor-Fusion Kalman Filter,” *GitHub repository*, 2014. [Online]. Available: https://github.com/memsindustrygroup/Open-Source-Sensor-Fusion/blob/master/docs/Sensor Fusion Kalman Filter.docx.

[38] “OpenIMU - The Open-Source GPS/INS Platform,” *Aceinna.com*, 2019. [Online].

Available: http://www.aceinna.com/openimu. [Accessed: 14-Mar-2019].

[39] OpenImuDevTeam, “python-openimu,” *GitHub repository*, 2019. [Online]. Available: https://github.com/Aceinna/python-openimu.

[40] OpenImuDevTeam, “Overview — Aceinna OpenIMU Developer Manual documentation,” *OpenIMU Documentation*, 2019. [Online]. Available: https://openimu.readthedocs.io/en/latest/intro.html. [Accessed: 14-Mar-2019].

[41] OpenImuDevTeam, “Kalman Filter — Aceinna OpenIMU Developer Manual documentation,” *OpenIMU Documentation*, 2019. [Online]. Available: https://openimu.readthedocs.io/en/latest/algorithms/KalmanFilter.html. [Accessed: 02-Apr-2019].

[42] PX4DevTeam, “PX4 Open Source Autopilot,” *PX4.io*, 2019. [Online]. Available: https://px4.io/. [Accessed: 27-Mar-2019].

[43] PX4, “PX4 Autopilot Firmware,” *GitHub repository*, 2019. [Online]. Available: https://github.com/PX4/Firmware. [Accessed: 27-Mar-2019].

[44] PX4DevTeam, “PX4 Autopilot Development Guide,” *Dronecode*, 2019. [Online]. Available: https://dev.px4.io/v1.8.2/en/.

[45] PX4DevTeam, “PX4 Autopilot Development Guide - Using the ECL EKF,” *Dronecode*, 2019. [Online]. Available: https://dev.px4.io/v1.8.2/en/tutorials/tuning_the_ecl_ekf.html. [Accessed: 29-Mar-2019].

[46] P. Riseborough, “PX4 Autopilot ECL EKF Documentation - Process and Observation Models,” *GitHub repository*, 2017. [Online]. Available: https://github.com/PX4/ecl/blob/master/EKF/documentation/Process and Observation Models.pdf. [Accessed: 03-Apr-2019].

[47] PX4DevTeam, “PX4 Autopilot User Guide,” *Dronecode*, 2019. [Online]. Available: https://docs.px4.io/en/. [Accessed: 27-Mar-2019].

[48] T. Moore, “robot_localization - ROS Wiki,” *ROS Wiki*, 2016. [Online]. Available: http://wiki.ros.org/robot_localization. [Accessed: 14-Mar-2019].

[49] Charles River Analytics, “robot_localization,” *GitHub repository*, 2019. [Online]. Available: https://github.com/cra-ros-pkg/robot_localization. [Accessed: 03-Apr-

2019].

[50] T. Moore and D. Stouch, “A Generalized Extended Kalman Filter Implementation for the Robot Operating System,” in *Intelligent Autonomous Systems 13*, Springer, 2016, pp. 335–348.

[51] T. Moore, “robot_localization wiki — robot_localization 2.6.4 documentation,” *Docs.ROS.org*. [Online]. Available: http://docs.ros.org/lunar/api/robot_localization/html/index.html. [Accessed: 14-Mar-2019].

[52] M. Alam, G. Moreno, M. Sipos, and J. Rohac, “INS / GNSS Localization Using 15 State Extended Kalman Filter,” in *International Conference in Aerospace for Young Scientists*, 2016, pp. 425–435.

[53] A. Dhariwal and G. S. Sukhatme, “Experiments in robotic boat localization,” in *IEEE International Conference on Intelligent Robots and Systems*, 2007, pp. 1702–1708.

[54] G. Papadopoulos, M. F. Fallon, J. J. Leonard, and N. M. Patrikalakis, “Cooperative localization of marine vehicles using nonlinear state estimation,” in *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, 2010, pp. 4874–4879.

[55] A. Subramanian, X. Gong, J. N. Riggins, D. J. Stilwell, and C. L. Wyatt, “Shoreline mapping using an omni-directional camera for autonomous surface vehicle applications,” in *Oceans 2006*, 2006, pp. 1–6.

[56] M. Karimi, M. Bozorg, and A. R. Khayatian, “A comparison of DVL/INS fusion by UKF and EKF to localize an autonomous underwater vehicle,” in *International Conference on Robotics and Mechatronics, ICRoM 2013*, 2013, pp. 62–67.

[57] B. Ferreira, A. Matos, and N. Cruz, “Single beacon navigation: Localization and control of the MARES AUV,” in *MTS/IEEE Seattle, OCEANS 2010*, 2010, pp. 1–9.

[58] A. S. Gadre and D. J. Stilwell, “Toward underwater navigation based on range measurements from a single location,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2004, vol. 2004, no. 5, pp. 4472–4477.

[59] G. C. Karras and K. J. Kyriakopoulos, “Localization of an underwater vehicle using an IMU and a laser-based vision system,” in *2007 Mediterranean Conference on Control*

& Automation, 2007, pp. 1–6.

[60] D. Loebis, R. Sutton, J. Chudley, and W. Naeem, “Adaptive tuning of a Kalman filter via fuzzy logic for an intelligent AUV navigation system,” *Control Eng. Pract.*, vol. 12, no. 12 SPEC. ISS., pp. 1531–1539, 2004.

[61] Y. Petillot, I. T. Ruiz, and D. M. Lane, “Underwater vehicle obstacle avoidance and path planning using a multi-beam forward looking sonar,” *IEEE J. Ocean. Eng.*, vol. 26, no. 2, pp. 240–251, 2001.

[62] M. T. Sabet, P. Sarhadi, and M. Zarini, “Extended and unscented Kalman filters for parameter estimation of a hydrodynamic model of vessel,” in *Ocean Engineering*, 2014, vol. 91 (2014), pp. 329–339.

[63] E. Bayramoglu, N. A. Andersen, N. K. Poulsen, J. C. Andersen, and O. Ravn, “Mobile robot navigation in a corridor using visual odometry,” in *2009 International Conference on Advanced Robotics*, 2009, pp. 1–6.

[64] L. Jetto, S. Longhi, and G. Venturini, “Development and experimental validation of an adaptive extended Kalman filter for the localization of mobile robots,” *IEEE Trans. Robot. Autom.*, vol. 15, no. 2, pp. 219–229, 1999.

[65] E. Kiriy and M. Buehler, “Three-state extended kalman filter for mobile robot localization,” *McGill Univ. Montr. Canada, Tech. Rep. TR-CIM*, vol. 5, p. 23, 2002.

[66] S. J. Kwon, K. W. Yang, S. Park, and Y. Ryuh, “Robust mobile robot localization with Combined Kalman Filter-perturbation estimator,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005, pp. 190–195.

[67] S. J. Kwon, K. W. Yang, and S. Park, “An effective kalman filter localization method for mobile robots,” in *IEEE International Conference on Intelligent Robots and Systems*, 2006, pp. 1524–1529.

[68] S. Lee and J. B. Song, “Robust mobile robot localization using optical flow sensors and encoders,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2004, vol. 2004, no. 1, pp. 1039–1044.

[69] C. H. Messom, G. Sen Gupta, S. Demidenko, and L. Y. Siong, “Improving predictive control of a mobile robot: Application of image processing and kalman filtering,” in *Proceedings of the 20th IEEE Instrumentation Technology Conference (Cat. No.*

03CH37412), 2003, vol. 2, pp. 1492–1496.

[70] K. Gade and F. Forskningsinstitutt, "Inertial Navigation - Theory and Applications," Norwegian University of Science and Technology, 2018.

[71] K. Gade, "The Seven Ways to Find Heading," *J. Navig.*, vol. 69, no. 05, pp. 955–970, 2016.

[72] S. Madgwick, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays," *Rep. x-io Univ. Bristol*, vol. 25, pp. 113–118, 2010.

[73] E. Macias, D. Torres, and S. Ravindran, "Nine-axis sensor fusion using the direction cosine matrix algorithm on the msp430f5xx family," *Texas Instruments Appl. Rep.*, pp. 1–15, 2012.

[74] M. Petovello, "How does a GNSS receiver estimate velocity?," *Insid. GNSS*, pp. 38–41, 2015.

[75] S. Williams, G. Dissanayake, and H. Durrant-Whyte, "Towards terrain-aided navigation for underwater robotics," *Adv. Robot.*, vol. 15, no. 5, pp. 533–549, 2001.

[76] M. Moore and J. Wang, "An extended dynamic model for kinematic positioning," *J. Navig.*, vol. 56, no. 1, pp. 79–88, 2003.

[77] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. 2011.

[78] R. R. Labbe, "Kalman and Bayesian Filters in Python," *GitHub repository*. Github. com, p. 497, 2017.

[79] C. Mongredien, "NEO-M8P u-blox M8 High Precision GNSS Modules," 2017.

[80] "SparkFun GPS-RTK Board - NEO-M8P-2 (Qwiic) - GPS-15005 - SparkFun Electronics." [Online]. Available: https://www.sparkfun.com/products/15005. [Accessed: 06-Mar-2019].

[81] "Copernicus II GPS Receiver," 2009.

[82] "SparkFun GPS Module - Copernicus II DIP (12 Channel) - GPS-11858 - SparkFun Electronics." [Online]. Available: https://www.sparkfun.com/products/11858. [Accessed: 06-Mar-2019].

[83] "GLOBALSAT GPS Module," 2013.

[84] "GPS Receiver - EM-506 (48 Channel) - GPS-12751 - SparkFun Electronics." [Online]. Available: https://www.sparkfun.com/products/12751. [Accessed: 06-Mar-2019].

[85] "Datasheet of GPS smart antenna module, LS20030~3," 2006.

[86] "GPS Receiver - LS20031 5Hz (66 Channel) - GPS-08975 - SparkFun Electronics." [Online]. Available: https://www.sparkfun.com/products/8975. [Accessed: 06-Mar-2019].

[87] "FGPMMOPA6H GPS Standalone Module Data Sheet," 2011.

[88] "Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates [Version 3] ID: 746 - $39.95 : Adafruit Industries, Unique &amp; fun DIY electronics and kits." [Online]. Available: https://www.adafruit.com/product/746?gclid=CjwKCAiA_P3jBRAqEiwAZyWWaNcjc17qhNCsX3lhl93N2lAQGpCRyRozq7JLT_eFK7Fr6rJ1LIvZHxoChJIQAvD_BwE. [Accessed: 06-Mar-2019].

[89] Y. Zhao, "Applying time-differenced carrier phase in nondifferential GPS/IMU tightly coupled navigation systems to improve the positioning performance," *IEEE Trans. Veh. Technol.*, vol. 66, no. 2, pp. 992–1003, 2016.

[90] "MPU-9250 Product Specification," 2014.

[91] "SparkFun 9DoF Razor IMU M0 - SEN-14001 - SparkFun Electronics." [Online]. Available: https://www.sparkfun.com/products/14001. [Accessed: 06-Mar-2019].

[92] "BNO080 Data Sheet," 2017.

[93] "SparkFun VR IMU Breakout - BNO080 (Qwiic) - SEN-14686 - SparkFun Electronics." [Online]. Available: https://www.sparkfun.com/products/14686. [Accessed: 06-Mar-2019].

[94] "UM7 Datasheet."

[95] "UM7-LT Orientation Sensor." [Online]. Available: https://www.pololu.com/product/2763. [Accessed: 06-Mar-2019].

[96] "VMU931 Technical Data Sheet," 2016.

[97] "Variense Compact IMU 9-Axis w/ Rugged Case - RobotShop." [Online]. Available: https://www.robotshop.com/en/variense-compact-imu-9-axis-w--rugged-case-vmu931b.html. [Accessed: 06-Mar-2019].

[98] T. SNAME, "Nomenclature for treating the motion of a submerged body through a fluid," *Soc. Nav. Archit. Mar. Eng. Tech. Res. Bull. No.*, pp. 1–5, 1950.

[99] "REP 103 -- Standard Units of Measure and Coordinate Conventions (ROS.org)."

[Online]. Available: http://www.ros.org/reps/rep-0103.html. [Accessed: 10-Jan-2019].

[100] J. J. Craig, *Introduction to Robotics: Mechanics and Control, Third Edition*, Third. Pearson Education, 2005.

[101] M. Nadri and H. Hammouri, "Design of a continuous-discrete observer for state affine systems," *Appl. Math. Lett.*, vol. 16, no. 6, pp. 967–974, 2003.

[102] T. Ahmed-Ali, R. Postoyan, and F. Lamnabhi-Lagarrigue, "Continuous--discrete adaptive observers for state affine systems," *Automatica*, vol. 45, no. 12, pp. 2986–2990, 2009.

[103] H. Hammouri, M. Nadri, and R. Mota, "Constant gain observer for continuous-discrete time uniformly observable systems," in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006, pp. 5406–5411.

[104] F. Deza, E. Busvelle, J. P. Gauthier, and D. Rakotopara, "High gain estimation for nonlinear systems," *Syst. Control Lett.*, vol. 18, no. 4, pp. 295–299, 1992.

[105] M. Nadri, H. Hammouri, and C. Astorga, "Observer design for continuous-discrete time state affine systems up to output injection," *Eur. J. Control*, vol. 10, no. 3, pp. 252–263, 2004.

[106] M. Arcak and D. NešIć, "A framework for nonlinear sampled-data observer design via approximate discrete-time models and emulation," *Automatica*, vol. 40, no. 11, pp. 1931–1938, 2004.

[107] I. Karafyllis and C. Kravaris, "From continuous-time design to sampled-data design of observers," *IEEE Trans. Automat. Contr.*, vol. 54, no. 9, pp. 2169–2174, 2009.

[108] D. Luong-Van, M. J. Tordon, and J. Katupitiya, "Covariance profiling for an adaptive Kalman filter to suppress sensor quantization effects," in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, 2004, vol. 3, pp. 2680-2685 Vol.3.

[109] J. Z. Sasiadek and Q. Wang, "Sensor fusion based on fuzzy Kalman filtering for autonomous robot vehicle," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, 1999, vol. 4, pp. 2970–2975.

[110] J. Z. Sasiadek, Q. Wang, and M. B. Zeremba, "Fuzzy adaptive Kalman filtering for

INS/GPS data fusion," in *Proceedings of the 2000 IEEE International Symposium on Intelligent Control. Held jointly with the 8th IEEE Mediterranean Conference on Control and Automation (Cat. No.00CH37147)*, 2000, pp. 181–186.

[111] M. Mallick, S. Coraluppi, and C. Carthel, "Advances in asynchronous and decentralized estimation," in *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*, 2001, vol. 4, pp. 4–1873.

[112] M. Mallick, S. Coraluppi, and Y. Bar-shalom, "Comparison of Out-of-sequence Measurement Algorithms in Multi-platform Target Tracking," *Proc. 4th Annu. Conf. Inf. Fusion*, vol. 2, 2012.

[113] A. Khosravian, J. Trumpf, R. Mahony, and T. Hamel, "Recursive attitude estimation in the presence of multi-rate and multi-delay vector measurements," *Proc. Am. Control Conf.*, vol. 2015-July, pp. 3199–3205, 2015.

[114] Y. Bar-Shalom, "Update with out-of-sequence measurements in tracking: Exact solution," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 38, no. 3, pp. 769–778, 2002.

[115] D. M. Lane, "Variance Sum Law I," *Online Stat Book*, 2006. [Online]. Available: http://onlinestatbook.com/2/summarizing_distributions/variance_sum_law.html. [Accessed: 08-Apr-2019].

[116] P. Tagare, "Signal averaging," *Biomed. Digit. signal Process. Willis Tompkins Ed.*, 1993.

[117] P. a Bromiley, "Products and Convolutions of Gaussian Probability Density Functions Density Functions," *Tina Memo*, vol. 3, no. 2003, p. No. 2003-003, 2003.

[118] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[119] E. Venator and RosDevelopers, "ROS nmea_navsat_driver," *ROS Wiki*, 2019. [Online]. Available: http://wiki.ros.org/nmea_navsat_driver. [Accessed: 20-Jun-2019].

[120] M. Purvis and A. Brown, "ROS um7," *ROS Wiki*, 2017. [Online]. Available: http://wiki.ros.org/um7. [Accessed: 12-Nov-2019].

[121] ROS, "ROS Bags," *Ros Wiki*, 2019. [Online]. Available: http://wiki.ros.org/Bags. [Accessed: 20-Jun-2019].

[122] D. Faconti, "ROS plotjuggler," *ROS Wiki*, 2019. [Online]. Available: http://wiki.ros.org/plotjuggler. [Accessed: 12-Nov-2019].

[123] Google, “Google Maps & Google Earth GeoGuidelines,” *google.com*, 2018. [Online]. Available: https://www.google.com/permissions/geoguidelines/. [Accessed: 20-Jun-2019].

[124] D. Rowell, “Review of first-and second-order system response,” *Report, Massachusetts Inst. Technol.*, 2004.

[125] RIP Laboratory, “State estimator for robot localization using only a single low cost GNSS/GPS and IMU sensor,” *Youtube*, 2019. [Online]. Available: https://youtu.be/waGiURYGILw. [Accessed: 18-Dec-2019].

[126] Riplaboratory, “3D Orientation Representation,” *Github repository*, 2019. [Online]. Available: https://github.com/riplaboratory/3D_orientation_representation. [Accessed: 27-Oct-2019].

8. Supplementary Materials

8.1. Robot simulation and state estimator Matlab source code

8.1.1. Simulation library script

```
classdef usvSim

    methods (Static)

        function [t,sim,sen] = simUserInputs()

            % USER INPUT: time properties
            t.dt = 0.001;   % simulation time step [s]
            t.start = 0;    % simulation start time [s]
            t.end = 50;     % simulation end time [s]

            % USER INPUT: surface vehicles lumped parameters
            sim.m = 225;    % mass/inertia [kg]
            sim.I = 100;    % rotational moment of inertia [kg*m^2]
            sim.bxr = 40;   % drag in the surge (robot frame along x) direction [N*s/m]
            sim.byr = 400;  % drag in the sway (robot frame along y) direction [N*s/m]
            sim.bpr = 300;  % rotational drag in the yaw (robot frame about z) direction [N*s/m]

            % USER INPUT: sensor setup
            sen.gnss.rr = 1;            % refresh rate of GNSS sensor [Hz]
            sen.gnss.linPosVar = 1;     % variance of the GNSS sensor data [m]
            sen.imu.rr = 20;            % refresh rate of IMU sensor [Hz]
            sen.imu.angPosVar = 0.05;   % variance of the IMU angular position sensor data [rad]
            sen.imu.angVelVar = 0.05;   % variance of the IMU angular velocity sensor data
[rad/s]
            sen.imu.linAccVar = 0.1;    % variance of the IMU linear acceleration sensor data
[m/s^2]

        end

        function t = timeSetup(t)

            % Time property calculations (not user input)
            t.N = round(t.end/t.dt - t.start/t.dt);     % total number of time steps [ ]
            t.now = t.start;                            % current time [s]

        end

        function sim = simSetup(sim,t)

            % State, output, and input vectors
            sim.xr = zeros(6,t.N);  % simulation state vector in robot reference frame
            sim.yr = zeros(9,t.N);  % simulation output vector in robot reference frame
            sim.xm = zeros(3,t.N);  % simulation state vector in map reference frame
            sim.u = zeros(3,t.N);   % input vector (control inputs act on robot frame)

            % Create simulation state matrices
            sim.A = [...
                0 0 0 1 0 0;...
                0 0 0 0 1 0;...
                0 0 0 0 0 1;...
                0 0 0 -sim.bxr/sim.m 0 0;...
                0 0 0 0 -sim.byr/sim.m 0;...
                0 0 0 0 0 -sim.bpr/sim.I];      % state matrix [6x6]
            sim.B = [...
                0 0 0;...
                0 0 0;...
                0 0 0;...
                1/sim.m 0 0;...
                0 1/sim.m 0;...
                0 0 1/sim.I];                   % input matrix [6x3]
            sim.C = [...
                eye(6,9);...
                [sim.A(4:6,:),zeros(3,3)]];     % output matrix [9x9]
            sim.D = [...
                zeros(6,3);...
                sim.B(4:6,:)];                  % feedthrough matrix [9x3]

        end

        function sen = senSetup(sen,t)

            % Sensor data vectors
            sen.gnss.linPos = zeros(2,t.end/(1/sen.gnss.rr));       % linear position data from
GNSS
            sen.gnss.linPosRbt = zeros(2,t.end/(1/sen.gnss.rr));    % linear position data from
GNSS, converted to the robot frame
            sen.gnss.angPos = zeros(2,t.end/(1/sen.gnss.rr));       % angular position data from
IMU, references to GNSS timestamp
            sen.imu.angPos = zeros(1,t.end/(1/sen.imu.rr));         % angular position data from
IMU
            sen.imu.angVel = zeros(1,t.end/(1/sen.imu.rr));         % angular velocity data from
IMU
            sen.imu.linAcc = zeros(2,t.end/(1/sen.imu.rr));         % linear acceleration data
from IMU

        end

        function sen = resetSensorTracker(sen)

            % Sensor update tracker
            sen.gnss.update = 0;            % gnss update tracker (boolean)
            sen.imu.update = 0;             % imu update tracker (boolean)
            sen.gnss.k = 1;                 % gnss iteration tracker
            sen.imu.k = 1;                  % imu iteration tracker
            sen.gnss.lastUpdateTime = 0;    % gnss last update time tracker [s]
            sen.imu.lastUpdateTime = 0;     % imu last update time tracker [s]

        end

        function sim = controller(sim,k)

%             % Straight-line controller
%             if (k == 0)
%                 sim.u(1,k) = 0;     % force input in surge direction
%                 sim.u(2,k) = 0;     % force input in sway direction
%                 sim.u(3,k) = 0;     % torque input in yaw direction
%             else
%                 sim.u(1,k) = 500;   % force input in surge direction
%                 sim.u(2,k) = 0;     % force input in sway direction
%                 sim.u(3,k) = 0;     % torque input in yaw direction
%             end

%             % Forward-backward-forward controller
%             if (k == 0)
%                 sim.u(1,k) = 0;     % force input in surge direction
%                 sim.u(2,k) = 0;     % force input in sway direction
%                 sim.u(3,k) = 0;     % torque input in yaw direction
%             elseif (k >= 1 && k < 15000)
%                 sim.u(1,k) = 500;   % force input in surge direction
%                 sim.u(2,k) = 0;     % force input in sway direction
%                 sim.u(3,k) = 0;     % torque input in yaw direction
%             elseif (k >= 15000 && k < 30000)
%                 sim.u(1,k) = -500;  % force input in surge direction
%                 sim.u(2,k) = 0;     % force input in sway direction
%                 sim.u(3,k) = 0;     % torque input in yaw direction
%              else
%                  sim.u(1,k) = 500;   % force input in surge direction
%                  sim.u(2,k) = 0;     % force input in sway direction
%                  sim.u(3,k) = 0;     % torque input in yaw direction
%              end

%              % Forward and rotate controller
%              if (k == 0)
%                  sim.u(1,k) = 0;     % force input in surge direction
%                  sim.u(2,k) = 0;     % force input in sway direction
%                  sim.u(3,k) = 0;     % torque input in yaw direction
%              else
%                  sim.u(1,k) = 350;   % force input in surge direction
%                  sim.u(2,k) = 0;     % force input in sway direction
%                  sim.u(3,k) = 100;   % torque input in yaw direction
%              end

%              % Constant surge, sway, and yaw controller
%              if (k == 0)
%                  sim.u(1,k) = 0;     % force input in surge direction
%                  sim.u(2,k) = 0;     % force input in sway direction
%                  sim.u(3,k) = 0;     % torque input in yaw direction
%              else
%                  sim.u(1,k) = 200;   % force input in surge direction
%                  sim.u(2,k) = 1000;  % force input in sway direction
%                  sim.u(3,k) = -100;   % torque input in yaw direction
%              end

            % Funky trajectory controller
            if (k == 0)
                sim.u(1,k) = 0;     % force input in surge direction
                sim.u(2,k) = 0;     % force input in sway direction
                sim.u(3,k) = 0;     % torque input in yaw direction
            elseif (k > 1 && k < 15000)
                sim.u(1,k) = 300;   % force input in surge direction
                sim.u(2,k) = 150;     % force input in sway direction
                sim.u(3,k) = 50;   % torque input in yaw direction
            elseif (k > 1 && k < 30000)
                sim.u(1,k) = -300;   % force input in surge direction
                sim.u(2,k) = -150;     % force input in sway direction
                sim.u(3,k) = -50;   % torque input in yaw direction
            elseif (k > 1 && k < 45000)
                sim.u(1,k) = 300;   % force input in surge direction
                sim.u(2,k) = 150;     % force input in sway direction
                sim.u(3,k) = 50;   % torque input in yaw direction
            else
                sim.u(1,k) = -300;     % force input in surge direction
                sim.u(2,k) = 150;     % force input in sway direction
                sim.u(3,k) = -50;     % torque input in yaw direction
            end

        end

        function sim = runSim(obj,t,sim,k)

            % Create discrete state matrices for Euler integration
            F = eye(6)+sim.A*t.dt;      % discrete state matrix
            G = sim.B*t.dt;             % discrete input matrix
            H = sim.C;                  % discrete output matrix
            J = sim.D;                  % discrete feedthrough matrix

            if k ~= t.N

                % Simulate plant using discrete Euler integration
                sim.xr(:,k+1) = F*sim.xr(:,k)+...
                    G*[sim.u(1,k);sim.u(2,k);sim.u(3,k)];     % state matrix integration solution
                sim.yr(:,k+1) = H*[sim.xr(:,k+1);zeros(3,1)]+...
                    J*[sim.u(1,k);sim.u(2,k);sim.u(3,k)];     % state observer matrix integration
solution
```

```
            % Convert pose in robot frame to pose in map frame
            [sim.xm(1,k+1),sim.xm(2,k+1)] =...
                obj.rbt2map(sim.xr(1,k+1),sim.xr(2,k+1),...
                sim.xr(1,k),sim.xr(2,k),sim.xr(3,k),...
                sim.xm(1,k),sim.xm(2,k));
            sim.xm(3,k+1) = sim.xr(3,k+1);

        end
    end

    function [xmf,ymf] = rbt2map(xrf,yrf,xr0,yr0,psi0,xm0,ym0)
        % Converts pose in the robot frame to pose in the map frame

        % Calculate translations and rotations in robot frame
        Txr = xrf-xr0;
        Tyr = yrf-yr0;

        % Calculate intermediate length and angle
        li = sqrt(Txr^2+Tyr^2);
        psii = atan2(yrf-yr0,xrf-xr0);

        % Calculate translations and rotations in map frame
        Txm = cos(psii+psi0)*li;
        Tym = sin(psii+psi0)*li;

        % Calculate individual components in the map frame
        xmf = xm0+Txm;
        ymf = ym0+Tym;

    end

    function [xrf,yrf] = map2rbt(xmf,ymf,xm0,ym0,psi0,xr0,yr0)
        % Converts pose in the map frame to pose in the robot frame

        % Calculate translations and rotations in robot frame
        Txm = xmf-xm0;
        Tym = ymf-ym0;

        % Calculate intermediate length and angle
        li = sqrt(Txm^2+Tym^2);
        psii = atan2(ymf-ym0,xmf-xm0);

        % Calculate translations and rotations in robot frame
        Txr = cos(psii-psi0)*li;
        Tyr = sin(psii-psi0)*li;

        % Calculate individual components in the robot frame
        xrf = xr0+Txr;
        yrf = yr0+Tyr;

    end

    function sen = checkSensorUpdate(t,sen,k)

        if k ~= t.N

            % Increment GNSS and set update flag to true if GNSS update
            % should occur
            if mod(k*t.dt,1/sen.gnss.rr) == 0       % update sensor reading at rr
                sen.gnss.k = sen.gnss.k+1;          % increment sensor
                sen.gnss.update = 1;                % set update flag to true
                sen.gnss.lastUpdateTime = t.now;    % set the last time the sensor updated
            else
                sen.gnss.update = 0;                % set update flag to false
            end

            % Increment IMU and set update flag to true if IMU update
            % should occur
            if mod(k*t.dt,1/sen.imu.rr) == 0        % update sensor reading at rr
                sen.imu.k = sen.imu.k+1;            % increment sensor
                sen.imu.update = 1;                  % set update flag to true
                sen.imu.lastUpdateTime = t.now;      % set the last time the sensor updated
            else
                sen.imu.update = 0;                  % set update flag to false
            end
        end
    end

    function sen = updateSensors(obj,t,sim,sen,k)

        if k ~= t.N

            % Add noise to GNSS states to simulate sensor data if
            % GNSS update should occur
            if sen.gnss.update == 1

                sen.gnss.linPos(1,sen.gnss.k) =...
                    sim.xm(1,k)+randn(1)*sqrt(sen.gnss.linPosVar);
                sen.gnss.linPos(2,sen.gnss.k) =...
                    sim.xm(2,k)+randn(1)*sqrt(sen.gnss.linPosVar);
                sen.gnss.angPos(1,sen.gnss.k) =...
                    sen.imu.angPos(1,sen.imu.k-1);

            end

            % Add noise to IMU states to simulate sensor data if
            % IMU update should occur
            if sen.imu.update == 1

                sen.imu.angPos(1,sen.imu.k) =...
                    sim.yr(3,k)+randn(1)*sqrt(sen.imu.angPosVar);
                sen.imu.angVel(1,sen.imu.k) =...
                    sim.yr(6,k)+randn(1)*sqrt(sen.imu.angVelVar);
                sen.imu.linAcc(1,sen.imu.k) =...
                    sim.yr(7,k)+randn(1)*sqrt(sen.imu.linAccVar);
                sen.imu.linAcc(2,sen.imu.k) =...
                    sim.yr(8,k)+randn(1)*sqrt(sen.imu.linAccVar);

            end
        end

        % Convert GNSS to robot frame (this MUST occur after previous steps)
        if sen.gnss.update == 1

            if sen.gnss.k ~= 1

                % Convert GNSS pose in map frame to pose in true robot frame
                [xrf,yrf] = obj.map2rbt(...
                    sen.gnss.linPos(1,sen.gnss.k),...
                    sen.gnss.linPos(2,sen.gnss.k),...
                    sen.gnss.linPos(1,sen.gnss.k-1),...
                    sen.gnss.linPos(2,sen.gnss.k-1),...
                    sen.gnss.angPos(1,sen.gnss.k),...
                    sen.gnss.linPosRbt(1,sen.gnss.k-1),...
                    sen.gnss.linPosRbt(2,sen.gnss.k-1));

                % Save robot frame conversion
                sen.gnss.linPosRbt(1,sen.gnss.k) = xrf;
                sen.gnss.linPosRbt(2,sen.gnss.k) = yrf;

            else

                sen.gnss.linPosRbt(1,sen.gnss.k) = 0;
                sen.gnss.linPosRbt(2,sen.gnss.k) = 0;

            end

        end

    end
```

```
        function est = estUserInputs()
            % User inputs for the estimator

            % USER INPUT: time step properties
            est.dt = 0.025;     % estimator time step [s]

            % USER INPUT: Number of past sensor updates to store for
            % calculating the incremental moving mean of sensor data
            % (if supported).
            est.linPos.n = 2;   % linear position mean array size (must be 2 or more)
            est.angPos.n = 2;   % angular position mean array size (must be 2 or more)
            est.angVel.n = 2;   % angular velocity mean array size (must be 2 or more)
            est.linAcc.n = 2;   % linear acceleration mean array size (must be 2 or more)

            % USER INPUT: sliding adaptive covariance gain
            est.gnssSlidingCovariance = 1;  % 1 for GNSS sliding covariance gain, 0 for binary
covariance gain calculation
            est.imuSlidingCovariance = 0;   % 1 for IMU sliding covariance gain, 0 for binary
covariance gain calculation

            % USER INPUT: decide which sensors to use for pseudo linear
            % velocity calculation
            est.useGnssLinVel = 1;  % set to 1 to use GNSS forward difference, set to 0 to ignore
            est.useImuLinVel = 1;   % set to 1 to use IMU trapezoidal method, set to 0 to ignore
            est.linVelBias = 1.35;   % bias (multiplier) on linear velocity output

            % USER INPUT: robot frame estimator measurement noise covariance
            est.rbt.R.linPos = 2;       % linear position [m]
            est.rbt.R.angPos = 0.075;   % angular position [rad]
            est.rbt.R.linVel = 2;       % linear velocity [m/s]
            est.rbt.R.angVel = 0.075;   % angular velocity [rad/s]
            est.rbt.R.linAcc = 0.25;    % linear acceleration [m/s^s]
            est.rbt.R.angAcc = 1;       % angular acceleration [rad/s^2] (not measured, so this
is actually irrelevant)

            % USER INPUT: robot frame estimator process noise covariance
            est.rbt.Q.linPos = est.rbt.R.linPos*1E1;    % linear position [m]
            est.rbt.Q.angPos = est.rbt.R.angPos*1E1;    % angular position [rad]
            est.rbt.Q.linVel = est.rbt.R.linVel*1E1;    % linear velocity [m/s]
            est.rbt.Q.angVel = est.rbt.R.angVel*1E-1;   % angular velocity [rad/s]
            est.rbt.Q.linAcc = est.rbt.R.linAcc*1E-1;   % linear acceleration [m/s^s]
            est.rbt.Q.angAcc = est.rbt.R.angAcc*1E-1;   % angular acceleration [rad/s^2] (not
measured, so this is actually irrelevant)

            % USER INPUT: robot frame estimator observation/observability
            est.H.linPos = 1;       % linear position (measured)
            est.H.angPos = 1;       % angular position (measured)
            est.H.linVel = 1;       % linear velocity (pseudo-measured)
            est.H.angVel = 1;       % angular velocity (measured)
            est.H.linAcc = 1;       % linear acceleration (measured)
            est.H.angAcc = 0;       % angular acceleration (NOT measured)

            % USER INPUT: map frame estimator measurement noise covariance
            est.map.R.linPos = 2;       % linear position [m]
            est.map.R.angPos = 0.075;   % angular position [rad]
            est.map.R.linVel = 1;       % linear velocity [m/s]
            est.map.R.angVel = 0.075;   % angular velocity [rad/s]

            % USER INPUT: map frame estimator process noise covariance
            est.map.Q.linPos = est.map.R.linPos*1E1;    % linear position [m]
            est.map.Q.angPos = est.map.R.angPos*1E5;    % angular position [rad]
            est.map.Q.linVel = est.map.R.linVel*1E5;    % linear velocity [m/s]
            est.map.Q.angVel = est.map.R.angVel*1E5;    % angular velocity [rad/s]

            % USER INPUT: map frame estimator observation/observability
            est.H.linPos = 1;       % linear position (measured)
            est.H.angPos = 1;       % angular position (measured)
            est.H.linVel = 1;       % linear velocity (pseudo-measured)
            est.H.angVel = 1;       % angular velocity (measured)
        end

        function est = estSetup(t,est)

            % Instantiate measurement variables and arrays
            est.k = 1;                                      % iteration tracking variable
            est.gnssLastUpdateTime = 0;                     % last GNSS update time [s]
            est.imuLastUpdateTime = 0;                      % last IMU update time [s]
            est.gnssSlidingGain = 1;                        % sliding covariance gain for GNSS
            est.imuSlidingGain = 1;                         % sliding covariance gain for IMU
            est.gnssImuSlidingGain = 1;                     % sliding covariance gain for
GNSS+IMU
            est.slidingGain = 1;                            % final sliding covariance gain for
estimator iteration
            est.linPosMapArray = zeros(2,est.linPos.n);     % linear position directly from GNSS
in map frame measurement array
            est.linPosRbtArray = zeros(2,est.linPos.n);     % linear position directly from GNSS
in robot frame measurement array
            est.angPosArray = zeros(1,est.angPos.n);        % angular position directly from IMU
in map and robot frame measurement array
            est.linVelGnssAprxRbt = zeros(2,1);             % approximated linear velocity in
robot frame from forward-difference of GNSS
            est.linVelImuAprxRbt = zeros(2,1);              % approximated linear velocity in
robot frame from trapezoidal method of IMU
            est.angVelArray = zeros(1,est.angVel.n);        % angular velocity directly from IMU
in map and robot frame measurement array
            est.linAccRbtArray = zeros(2,est.linAcc.n);     % linear acceleration directly from
IMU in robot frame measurement array

            % Instantiate estimator matrices
            est.rbt.m = zeros(9,t.end/est.dt);              % robot frame estimator measurement
matrix
            est.rbt.x = zeros(9,t.end/est.dt);              % robot frame estimator state
estimate
            est.rbt.L = zeros(9,9,t.end/est.dt);            % robot frame estimator kalman gain
matrix
            est.rbt.P = zeros(9,9,t.end/est.dt);            % robot frame estimator covariance
matrix
            est.rbt.s = zeros(9,9,t.end/est.dt);            % robot frame sliding covariance gain
matrix
            est.map.m = zeros(6,t.end/est.dt);              % map frame estimator measurement
matrix
            est.map.x = zeros(6,t.end/est.dt);              % map frame estimator state estimate
            est.map.L = zeros(6,6,t.end/est.dt);            % map frame estimator kalman gain
matrix
            est.map.P = zeros(6,6,t.end/est.dt);            % map frame estimator covariance
matrix
            est.map.s = zeros(6,6,t.end/est.dt);            % map frame sliding covariance gain
matrix

            % Robot frame covariance, obvervation, state and input matrices
            est.rbt.Q = [...
                est.rbt.Q.linPos 0 0 0 0 0 0 0 0;...
                0 est.rbt.Q.linPos 0 0 0 0 0 0 0;...
                0 0 est.rbt.Q.angPos 0 0 0 0 0 0;...
                0 0 0 est.rbt.Q.linVel 0 0 0 0 0;...
                0 0 0 0 est.rbt.Q.linVel 0 0 0 0;...
                0 0 0 0 0 est.rbt.Q.angVel 0 0 0;...
                0 0 0 0 0 0 est.rbt.Q.linAcc 0 0;...
                0 0 0 0 0 0 0 est.rbt.Q.linAcc 0;...
                0 0 0 0 0 0 0 0 est.rbt.Q.angAcc];       % process noise covariance matrix
            est.rbt.R = [...
                est.rbt.R.linPos 0 0 0 0 0 0 0 0;...
                0 est.rbt.R.linPos 0 0 0 0 0 0 0;...
                0 0 est.rbt.R.angPos 0 0 0 0 0 0;...
                0 0 0 est.rbt.R.linVel 0 0 0 0 0;...
                0 0 0 0 est.rbt.R.linVel 0 0 0 0;...
                0 0 0 0 0 est.rbt.R.angVel 0 0 0;...
                0 0 0 0 0 0 est.rbt.R.linAcc 0 0;...
                0 0 0 0 0 0 0 est.rbt.R.linAcc 0;...
                0 0 0 0 0 0 0 0 est.rbt.R.angAcc];      % measurement noise covariance matrix
            est.rbt.H = [...
                est.H.linPos 0 0 0 0 0 0 0 0;...
                0 est.H.linPos 0 0 0 0 0 0 0;...
                0 0 est.H.angPos 0 0 0 0 0 0;...
                0 0 0 est.H.linVel 0 0 0 0 0;...
                0 0 0 0 est.H.linVel 0 0 0 0;...
                0 0 0 0 0 est.H.angVel 0 0 0;...
                0 0 0 0 0 0 est.H.linAcc 0 0;...
                0 0 0 0 0 0 0 est.H.linAcc 0;...
                0 0 0 0 0 0 0 0 est.H.angAcc];       % obervation matrix
            est.rbt.A = [...
                1 0 0 est.dt 0 0 0 0 0;...
                0 1 0 0 est.dt 0 0 0 0;...
                0 0 1 0 0 est.dt 0 0 0;...
                0 0 0 1 0 0 0 0 0;...
                0 0 0 0 1 0 0 0 0;...
                0 0 0 0 0 1 0 0 0;...
                0 0 0 0 0 0 1 0 0;...
                0 0 0 0 0 0 0 1 0;...
                0 0 0 0 0 0 0 0 1];                     % state matrix
            est.rbt.B = zeros(9,3);                     % input matrix

            % Map frame covariance, obvervation, state and input matrices
            est.map.Q = [...
                est.map.Q.linPos 0 0 0 0 0;...
                0 est.map.Q.linPos 0 0 0 0;...
                0 0 1E5 0 0 0;...
                0 0 0 1E5 0 0;...
                0 0 0 0 1E5 0;...
                0 0 0 0 0 1E5];                 % process noise covariance matrix
            est.map.R = [...
                est.map.R.linPos 0 0 0 0 0;...
                0 est.map.R.linPos 0 0 0 0;...
                0 0 est.map.R.angPos 0 0 0;...
                0 0 0 est.map.R.linVel 0 0;...
                0 0 0 0 est.map.R.linVel 0;...
                0 0 0 0 0 est.map.R.angVel];    % measurement noise covariance matrix
            est.map.H = [...
                est.H.linPos 0 0 0 0 0;...
                0 est.H.linPos 0 0 0 0;...
                0 0 est.H.angPos 0 0 0;...
                0 0 0 est.H.linVel 0 0;...
                0 0 0 0 est.H.linVel 0;...
                0 0 0 0 0 est.H.angVel];        % obervation matrix
            est.map.A = @(x) [...
                1 0 0 cos(x(3))*est.dt -sin(x(3))*est.dt 0;...
                0 1 0 sin(x(3))*est.dt cos(x(3))*est.dt 0;...
                0 0 1 0 0 0;...
                0 0 0 1 0 0;...
                0 0 0 0 1 0;...
                0 0 0 0 0 1];                   % state matrix
            est.map.B = zeros(6,3);             % input matrix

            % Warning message
            if (est.useGnssLinVel == 0 && est.useImuLinVel == 1)
                disp('Warning: you have elected to estimate linear velocity only using IMU
measurements, this simulation cannot replicate the real-world drift of IMU sensors, so this
simulation will return non-sensical linear velocity accuracy');
            elseif (est.useGnssLinVel == 0 && est.useImuLinVel == 0)
                disp('Error with linear velocity pseudo-measurement sensor selection. Simulation
will return non-sensical result.');
            end

        end

        function est = runEst(obj,t,est,sen,k)

            % Import sensor measurements to estimator measurement arrays
            est = obj.estMeasArrays(est,sen);

            % Calculate sliding covariance gain components for for GNSS and IMU sensor
            est = obj.estSlidingCovGain(t,est,sen);

            % Calculate measurement and sliding covariance gain for each respective robot frame
state
            est = obj.estRbtMeas(obj,est,sen);

            % Run robot frame Kalman filter
            est = obj.runRbtKal(obj,t,est,k);

            % Calculate measurement and sliding covariance gain for each respective map frame
state
            est = obj.estMapMeas(k,t,est);

            % Run map frame Kalman filter
            est = obj.runMapKal(obj,t,est,k);

        end

        function est = estMeasArrays(est,sen)

            % If GNSS updated, save data to measurement arrays
            if sen.gnss.update == 1

                % Set estimator GNSS last known sensor update to the sensor
                % last known update
                est.gnssLastUpdateTime = sen.gnss.lastUpdateTime;

                % Update the linear position in map frame measurement array
                est.linPosMapArray(:,2:end) = est.linPosMapArray(:,1:end-1);
                est.linPosMapArray(:,1) = sen.gnss.linPos(1:2,sen.gnss.k);

                % Update the linear positon in the robot frame measurement
                % array
                est.linPosRbtArray(:,2:end) = est.linPosRbtArray(:,1:end-1);
                est.linPosRbtArray(:,1) = sen.gnss.linPosRbt(1:2,sen.gnss.k);

            end

            % If IMU updated, save data to measurement arrays
            if sen.imu.update == 1

                % Set estimator last known sensor update to last known
                % sensor update
                est.imuLastUpdateTime = sen.imu.lastUpdateTime;

                % Update the angular position (in map and robot frame) measurement array
                est.angPosArray(1,2:end) = est.angPosArray(1,1:end-1);
                est.angPosArray(1,1) = sen.imu.angPos(1,sen.imu.k);

                % Update the angular velocity (in map and robot frame) measurement array
                est.angVelArray(1,2:end) = est.angVelArray(1,1:end-1);
                est.angVelArray(1,1) = sen.imu.angVel(1,sen.imu.k);

                % Update the linear acceleration in robot frame measurement array
                est.linAccRbtArray(:,2:end) = est.linAccRbtArray(:,1:end-1);
                est.linAccRbtArray(:,1) = sen.imu.linAcc(1:2,sen.imu.k);

            end

        end

        function est = estSlidingCovGain(t,est,sen)

            % Time calcuations
            dtGnss = 1/sen.gnss.rr;                     % time step of a single GNSS sensor
update
            dtImu = 1/sen.imu.rr;                       % time step of a single IMU sensor update
            gnssTSLU = t.now - est.gnssLastUpdateTime;  % time since last GNSS update
            imuTSLU = t.now - est.imuLastUpdateTime;    % time since last IMU update

            % GNSS sliding covariance matrix variables
            Gm = 10^5;                              % maximum covariance gain
            tscale = 0.15*dtGnss;                   % length of scaling region [s]
            tsmooth = 0.1*dtGnss;                   % length of scaling region [s]
            halfLifeSmooth = 0.1*tsmooth*log(2);    % half life of exponential decay of smoothing
curve
            halfLifeScale = 0.1*tscale*log(2);      % half life of exponential decay expressed as
a percentage of the scale time

            % Sliding adaptive covariance gain calculation
            if (est.gnssSlidingCovariance == 1)
                % GNSS smoothing region
                if (gnssTSLU >= 0 && gnssTSLU < tsmooth)
                    est.gnssSlidingGain = Gm*2.^...
                        (-gnssTSLU*(1/halfLifeSmooth))+1;
                % GNSS scaling region
                elseif (gnssTSLU >= tsmooth && gnssTSLU <= tsmooth+tscale)
                    est.gnssSlidingGain = Gm*2.^...
                        ((gnssTSLU-(tscale+tsmooth))*(1/halfLifeScale))+1;
                % GNSS rejected region
                else
                    est.gnssSlidingGain = Gm;
                end
            % GNSS binary adaptive covariance gain
            else
                if sen.gnss.update == 1
                    est.gnssSlidingGain = 1;
                else
                    est.gnssSlidingGain = Gm;
                end
            end

            % Note that if the refresh rate of the estimator should be at
            % least an order of magnitude faster than the IMU refresh rate,
            % otherwise you will run into discrete-time rounding issues. If
            % this occurs, use binary covariance profiling for IMU.

            % IMU sliding covariance matrix variables
            tscale = 0.15*dtImu;                    % length of scaling region [s]
            tsmooth = 0.05*dtImu;                   % length of scaling region [s]
            halfLifeSmooth = 0.1*tsmooth*log(2);    % half life of exponential decay of smoothing
curve
            halfLifeScale = 0.1*tscale*log(2);      % half life of exponential decay expressed as
a percentage of the scale time

            % IMU sliding covariance gain calculation
            if (est.imuSlidingCovariance == 1)
                % IMU smoothing region
                if (imuTSLU >= 0 && imuTSLU < imuSmoothTimeEnd)
                    est.imuSlidingGain = Gm*2.^...
                        (-imuTSLU*(1/halfLifeSmooth))+1;
                % IMU scaling region
                elseif (imuTSLU >= imuSmoothTimeEnd && imuTSLU <= imuScaleTimeEnd)
                    est.imuSlidingGain = Gm*2.^...
                        ((imuTSLU-(tscale+tsmooth))*(1/halfLifeScale))+1;
                % IMU rejected region
                else
                    est.imuSlidingGain = Gm;
                end
            % IMU binary adaptive covariance gain calculation
            else
                if sen.imu.update == 1
                    est.imuSlidingGain = 1;
                else
                    est.imuSlidingGain = Gm;
                end
            end
```

```
        end

        function est = estRbtMeas(obj,est,sen)

            % Set sliding covariance gain matrix to identity
            est.rbt.s(:,:,est.k) = eye(9,9);                    % robot frame sliding covariance
gain matrix

            % Linear position measurements and gains
            est.rbt.m(1:2,est.k) = est.linPosRbtArray(1:2,1);   % linear positions in robot frame
            est.rbt.s(1:2,:,est.k) =...
                est.rbt.R(1:2,:)*est.gnssSlidingGain;           % linear positions in revised
measurement noise covariance matrix based on sliding adaptive covariance gain

            % Angular position measurements and gains
            est.rbt.m(3,est.k) = est.angPosArray(1,1);          % angular position in robot frame
            est.rbt.s(3,:,est.k) =...
                est.rbt.R(3,:)*est.imuSlidingGain;              % angular position in revised
measurement noise covariance matrix based on sliding adaptive covariance gain

            % Linear velocity pseudo measurements and gains
            est = obj.estRbtLinVel(est,sen);                    % pseudo linear velocity
measurements

            % Angular velocity measurements
            est.rbt.m(6,est.k) = est.angVelArray(1,1);          % angular velocity in robot frame
            est.rbt.s(6,:,est.k) =...
                est.rbt.R(6,:)*est.imuSlidingGain;              % angular velocity in revised
measurement noise covariance matrix based on sliding adaptive covariance gain

            % Linear acceleration measurements
            est.rbt.m(7:8,est.k) = est.linAccRbtArray(1:2,1);   % linear accelerations in robot
frame
            est.rbt.s(7:8,:,est.k) =...
                est.rbt.R(7:8,:)*est.imuSlidingGain;            % linear accelerations in revised
measurement noise covariance matrix based on sliding adaptive covariance gain

        end

        function est = estRbtLinVel(est,sen)

            % Take only the forward-difference of linear positon data from
            % GNSS in the robot frame
            if (est.useGnssLinVel == 1 && est.useImuLinVel == 0)

                % First, calculate the forward difference of GNSS linear
                % position data
                est.linVelGnssAprxRbt(:,1) =...
                    (1/((1/sen.gnss.rr)*est.linPos.n))*...
                    (est.linPosRbtArray(1:2,1)-...
                    est.linPosRbtArray(1:2,end));               % linear velocity pseudo-
measurements due to forward difference of GNSS

                % Second, calculate variance of pseudo-measurements based
                % on forward-difference of GNSS linear position data
                % (going to ignore user values in the R matrix, since this
                % should be derived)
                vardxG =(sqrt(2)/((1/sen.gnss.rr)^2))*...
                    est.rbt.R(1,1)*est.gnssSlidingGain;         % variance of linear velocity
pseudo-measurements along x due to forward difference of GNSS
                vardyG = (sqrt(2)/((1/sen.gnss.rr)^2))*...
                    est.rbt.R(2,2)*est.gnssSlidingGain;         % variance of linear velocity
pseudo-measurements along y due to forward difference of GNSS

                % Third, save to measurement vector
                est.rbt.m(4:5,est.k) =...
                    est.linVelGnssAprxRbt(:,1)*...
                    est.linVelBias;                             % save robot frame pseudo-
measurement to measurement matrix
```

```
                % Fourth, save to revised measurement noise covariance
                % matrix based on sliding adaptive covariance gain
                est.rbt.s(4:5,:,est.k) = [...
                    0 0 0 vardxG 0 0 0 0 0;...
                    0 0 0 0 vardyG 0 0 0 0];                    % linear velocities in revised
measurement noise covariance matrix based on sliding adaptive covariance gain

            % Takes only the trapezoidal method of the linear acceleration
            % data from IMU in the map and robot frames
            elseif (est.useGnssLinVel == 0 && est.useImuLinVel == 1)

                % First, calculate the trapezoidal method of IMU linear
                % acceleration data
                if (est.k-1 == 0)
                    est.linVelImuAprxRbt(:,1) =...
                        (1/2)*(est.linAccRbtArray(1:2,2)+...
                        est.linAccRbtArray(1:2,1))*...
                        (1/sen.imu.rr);                         % trapezoidal method of linear
acceleration data in robot frame
                else
                     est.linVelImuAprxRbt(:,1) =...
                        est.rbt.x(4:5,est.k-1)+...
                        (1/2)*(est.linAccRbtArray(1:2,2)+...
                        est.linAccRbtArray(1:2,1))*...
                        (1/sen.imu.rr);                         % trapezoidal method of linear
acceleration data in robot frame based on last estimate
                end

                % Second, calculate variance of pseudo-measurements based
                % on trapezoidal method of IMU linear acceleration data
                % (going to ignore user values in the R matrix, since this
                % should be derived)
                vardxI = 1;                                     % nonsensical variance due to
integration summation
                vardyI = 1;                                     % nonsensical variance due to
integration summation

                % Third, save to measurement vector
                est.rbt.m(4:5,est.k) =...
                    est.linVelImuAprxRbt(:,1)*...
                    est.linVelBias;                             % save robot frame pseudo-
measurement to measurement matrix

                % Fourth, save to revised measurement noise covariance
                % matrix based on sliding adaptive covariance gain
                est.rbt.s(4:5,:,est.k) = [...
                    0 0 0 vardxI 0 0 0 0 0;...
                    0 0 0 0 vardyI 0 0 0 0];                    % linear velocities in revised
measurement noise covariance matrix based on sliding adaptive covariance gain

            % Takes both forward-difference of linear position data
            % from GNSS in map and robot frames AND trapezoidal method
            % from IMU in robot frame
            elseif (est.useGnssLinVel == 1 && est.useImuLinVel == 1)

                % First, calculate the forward difference of GNSS linear
                % position data and trapezoidal method of IMU linear
                % acceleration data
                est.linVelGnssAprxRbt(:,1) =...
                    (1/((1/sen.gnss.rr)*est.linPos.n))*...
                    (est.linPosRbtArray(1:2,1)-...
                    est.linPosRbtArray(1:2,end));               % linear velocity pseudo-
measurements due to forward difference of GNSS
                if (est.k-1 == 0)
                    est.linVelImuAprxRbt(:,1) =...
                        (1/2)*(est.linAccRbtArray(1:2,2)+...
                        est.linAccRbtArray(1:2,1))*...
                        (1/sen.imu.rr);                         % trapezoidal method of linear
acceleration data in robot frame
```

```
                else
                     est.linVelImuAprxRbt(:,1) =...
                        est.rbt.x(4:5,est.k-1)+...
                        (1/2)*(est.linAccRbtArray(1:2,2)+...
                        est.linAccRbtArray(1:2,1))*...
                        (1/sen.imu.rr);                         % trapezoidal method of linear
acceleration data in robot frame based on last estimate
                end

                % Second, calculate variance of pseudo-measurements based
                % on forward-difference of GNSS linear position data and on
                % trapezoidal method of IMU linear acceleration data
                % (going to ignore user values in the R matrix, since this
                % should be derived)
                vardxG =(sqrt(2)/((1/sen.gnss.rr)^2))*...
                    est.rbt.R(1,1)*est.gnssSlidingGain;         % variance of linear velocity
pseudo-measurements along x due to forward difference of GNSS
                vardyG = (sqrt(2)/((1/sen.gnss.rr)^2))*...
                    est.rbt.R(2,2)*est.gnssSlidingGain;         % variance of linear velocity
pseudo-measurements along y due to forward difference of GNSS
                vardx = (vardxG^2)/(2*vardxG);                  % variance of linear velocity
pseudo-measurements along x
                vardy = (vardyG^2)/(2*vardyG);                  % variance of linear velocity
pseudo-measurements along y

                % Third, save to measurement vector
                est.rbt.m(4,est.k) =...
                    (est.linVelGnssAprxRbt(1,1)+...
                    est.linVelImuAprxRbt(1,1))/2*...
                    est.linVelBias;
                est.rbt.m(5,est.k) =...
                    (est.linVelGnssAprxRbt(2,1)+...
                    est.linVelImuAprxRbt(2,1))/2*...
                    est.linVelBias;

                % Fourth, save to revised measurement noise covariance
                % matrix based on sliding adaptive covariance gain
                est.rbt.s(4:5,:,est.k) = [...
                    0 0 0 vardx 0 0 0 0 0;...
                    0 0 0 0 vardy 0 0 0 0];                     % linear velocities in revised
measurement noise covariance matrix based on sliding adaptive covariance gain

            % If this condition throws, there is an error
            else

                disp('Error in assigning mcaEst.useGnssLinVel and mcaEst.useImuLinVel
variables');
                est.rbt.m(4:5,est.k) = [0;0];
                est.rbt.s(4:5,:,est.k) = [...
                    0 0 0 1 0 0 0 0 0;...
                    0 0 0 0 1 0 0 0 0];

            end

        end

        function est = runRbtKal(obj,t,est,k)

            if k ~= t.N

                % Executes when estimator should update
                if mod(k*t.dt,est.dt) == 0

                    % State estimator setup
                    state.x = est.rbt.x(:,est.k);      % last state estimate vector from robot
frame
                    state.P = est.rbt.P(:,:,est.k);    % last covariance matrix
                    state.Q = est.rbt.Q;               % process noise covariance matrix
                    state.R = est.rbt.s(:,:,est.k);    % measurement noise covariance
                    state.H = est.rbt.H;               % observation matrix
                    state.z = est.rbt.m(:,est.k);     % measurement vector
                    state.u = [0;0;0];                % control input vector (don't give kalman
filter knowledge about thruster inputs)

                    % Discrete Kalman filter
                    state.A = est.rbt.A;
                    state.B = est.rbt.B;
                    state = obj.kalmanf(state);

                    % Store state estimate
                    est.rbt.x(:,est.k+1) = state.x;
                    est.rbt.L(:,:,est.k+1) = state.K;
                    est.rbt.P(:,:,est.k+1) = state.P;

                % Executes when estimator should not update (this happens
                % when the simulation executes more rapidly than the
                % estimator)

                else

                    % Do nothing

                end

            end

        end

        function est = estMapMeas(k,t,est)

            % Linear position measurements and gains
            est.map.m(1:2,est.k) =...
                est.linPosMapArray(1:2,1);                  % linear position in map frame
            est.map.s(1:2,:,est.k) =...
                est.map.R(1:2,:)*est.gnssSlidingGain;       % linear positions in sliding
covariance gain matrix

            % Angular position measurements and gains
            if k ~= t.N
                est.map.m(3,est.k) = est.rbt.x(3,est.k+1);      % angular position in map frame
            end
            est.map.s(3,:,est.k) =...
                est.map.R(3,:)*est.imuSlidingGain;          % angular position in sliding
covariance gain matrix

            % Robot frame linear velocity pseudo measurements and gains
            if k ~= t.N
                est.map.m(4:5,est.k) = est.rbt.x(4:5,est.k+1);
            end
            if (est.useGnssLinVel == 1 && est.useImuLinVel == 0)
                est.map.s(4:5,:,est.k) =...
                    est.map.R(4:5,:)*est.gnssSlidingGain;
            elseif (est.useGnssLinVel == 0 && est.useImuLinVel == 1)
                est.map.s(4:5,:,est.k) =...
                    est.map.R(4:5,:)*est.imuSlidingGain;
            elseif (est.useGnssLinVel == 1 && est.useImuLinVel == 1)
                est.map.s(4:5,:,est.k) =...
                    est.map.R(4:5,:)*est.gnssImuSlidingGain;
            else
                est.map.s(4:5,:,est.k) =...
                    est.map.R(4:5,:);
            end

            % Angular velocity measurements
            if k ~= t.N
                est.map.m(6,est.k) = est.rbt.x(6,est.k+1);      % angular velocity in map frame
            end
            est.map.s(6,:,est.k) =...
                est.map.R(6,:)*est.imuSlidingGain;          % angular velocity in sliding
covariance gain matrix
```

```
        end

        function est = runMapKal(obj,t,est,k)

            if k ~= t.N

                % Executes when estimator should update
                if mod(k*t.dt,est.dt) == 0

                    % State estimator setup
                    state.x = est.map.x(:,est.k);       % last estimate from robot frame
                    state.P = est.map.P(:,:,est.k);     % last covariance matrix
                    state.Q = est.map.Q;                % process noise covariance
                    state.R = est.map.R*...
                        est.map.s(:,:,est.k);           % measurement noise covariance
                    state.H = est.map.H;                % measurement matrix
                    state.z = est.map.m(:,est.k);       % meausurements
                    state.u = [0;0;0];                  % control input vector (don't give kalman
filter knowledge about thruster inputs)

                    % Discrete Kalman filter
                    state.A = est.map.A(est.map.m(:,est.k));
                    state.B = est.map.B;
                    state = obj.kalmanf(state);

                    % Store state estimate
                    est.map.x(:,est.k+1) = state.x;
                    est.map.L(:,:,est.k+1) = state.K;
                    est.map.P(:,:,est.k+1) = state.P;

                    % Increment tracking k variable
                    est.k = est.k+1;

                else

                    % Do nothing (might put code here in the future)

                end

            end

        end

        function s = kalmanf(s)

            % set defaults for absent fields:
            if ~isfield(s,'x'); s.x = nan*z; end
            if ~isfield(s,'P'); s.P = nan; end
            if ~isfield(s,'z'); error('Observation vector missing'); end
            if ~isfield(s,'u'); s.u = 0; end
            if ~isfield(s,'A'); s.A = eye(length(s.x)); end
            if ~isfield(s,'B'); s.B = 0; end
            if ~isfield(s,'Q'); s.Q = zeros(length(s.x)); end
            if ~isfield(s,'R'); error('Observation covariance missing'); end
            if ~isfield(s,'H'); s.H = eye(length(s.x)); end

            % Prediction for state vector and covariance:
            s.x = s.A*s.x+s.B*s.u;
            s.P = s.A*s.P*s.A'+s.Q;

            % Compute Kalman gain factor:
            s.K = s.P*s.H'*inv(s.H*s.P*s.H'+s.R);

            % Correction based on observation:
            s.x = s.x + s.K*(s.z-s.H*s.x);
            s.P = s.P - s.K*s.H*s.P;

        end
```

```
    end

end
```

8.1.2. Simulation data generation script

```
function genData()

    % Create new USV simulation
    usv = usvSim;

    % Setup USV simulation
    [t,sim,sen] = usv.simUserInputs();
    t = usv.timeSetup(t);
    sim = usv.simSetup(sim,t);
    sen = usv.senSetup(sen,t);
    sen = usv.resetSensorTracker(sen);

    % Run simulation
    for k = 1:1:t.N

        % Controller
        sim = usv.controller(sim,k);

        % Simulate robot
        sim = usv.runSim(usv,t,sim,k);

        % Update simulated sensor data
        sen = usv.checkSensorUpdate(t,sen,k);
        sen = usv.updateSensors(usv,t,sim,sen,k);

        % Increment time
        t.now = t.now+t.dt;

    end

    % Save output to file
    save('t.mat','t');
    save('sim.mat','sim');
    save('sen.mat','sen');

end
```

8.1.3. Run simulation and plot script

```
function runEstOnData()

    % Create new wamv simulation
    usv = usvSim;

    % Load existing sim and sen structures from file
    load('t.mat','t');
    load('sim.mat','sim');
    load('sen.mat','sen');

    % Reset the sensor tracking variables
    sen = usv.resetSensorTracker(sen);

    % Setup USV estimator
    est = usv.estUserInputs();
    est = usv.estSetup(t,est);
    % Run simulation
    for k = 1:1:t.N

        % Update sensor incrementors
        sen = usv.checkSensorUpdate(t,sen,k);

        % Executes when estimator should update
        if mod(k*t.dt,est.dt) == 0

            % Estimate state using constant-acceleration model
            est = usv.runEst(usv,t,est,sen,k);

        end

        % Increment time
        t.now = t.now+t.dt;

    end

    % Send stuff to workspace for debugging
    assignin('base','sim',sim);
    assignin('base','sen',sen);
    assignin('base','est',est);

    % Plot
    close all;
    plotXY(sim,sen,est,1);
    plotRbt(t,sim,sen,est,2);
    plotMap(t,sim,sen,est,3);

end

function plotXY(sim,sen,est,figN)

    % Set font size
    titleFontSize = 32;
    defaultFontSize = 28;
    markerSize = 10;
    trialTitleString =...
        '$Map\hspace{1mm}Frame\hspace{1mm}xy\hspace{1mm}Position$';

    % Robot x vs y
    figure(figN);
    hold on;
    plot(sim.xm(1,:),sim.xm(2,:),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(sen.gnss.linPos(1,:),sen.gnss.linPos(2,:),...
        'r','LineStyle','-','Marker','o','LineWidth',1,'MarkerSize',markerSize);
    plot(est.map.x(1,:),est.map.x(2,:),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    xlabel('$x_m\hspace{1mm}[m]$','Interpreter','latex');
    ylabel('$y_m\hspace{1mm}[m]$','Interpreter','latex');
    grid on;
    axis equal;
    legend('$simulated\hspace{1mm}ground\hspace{1mm}truth$',...
        '$simulated\hspace{1mm}measurement$',...
        '$estimator\hspace{1mm}output$',...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
    title(trialTitleString,...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    set(gcf,'Position',[0 0 1920 1280])

end

function plotRbt(t,sim,sen,est,figN)

    % Set font size
```

```
titleFontSize = 32;
defaultFontSize = 28;
markerSize = 10;

% Robot frame plot position states
figure(figN);
tt = 0:t.dt:t.end;
ttGnss = 0:1/sen.gnss.rr:t.end;
ttImu = 0:1/sen.imu.rr:t.end;
ttEst = 0:est.dt:t.end;
xr = subplot(3,3,1);
hold on;
plot(tt(1:end-1),sim.yr(1,:),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(ttGnss(1:end-1),sen.gnss.linPosRbt(1,:),...
    'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/1.5);
plot(ttEst(1:end-1),est.rbt.x(1,:),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
ylabel('$x_r\hspace{1mm}[m]$','Interpreter','latex');
title('$Robot\hspace{1mm}body\hspace{1mm}x_{r}\hspace{1mm}position$',...
    'FontSize',titleFontSize,...
    'Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
yr = subplot(3,3,4);
hold on;
plot(tt(1:end-1),sim.yr(2,:),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(ttGnss(1:end-1),sen.gnss.linPosRbt(2,:),...
    'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/1.5);
plot(ttEst(1:end-1),est.rbt.x(2,:),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
ylabel('$y_r\hspace{1mm}[m]$','Interpreter','latex');
title('$Robot\hspace{1mm}body\hspace{1mm}y_{r}\hspace{1mm}position$',...
    'FontSize',titleFontSize,...
    'Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
psir = subplot(3,3,7);
hold on;
plot(tt(1:end-1),rad2deg(sim.yr(3,:)),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(ttImu(1:end-1),rad2deg(sen.imu.angPos(1,:)),...
    'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
plot(ttEst(1:end-1),rad2deg(est.rbt.x(3,:)),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
ylabel('$\psi_r\hspace{1mm}[rad]$','Interpreter','latex');
title('$Robot\hspace{1mm}body\hspace{1mm}\psi_{r}\hspace{1mm}position$',...
    'FontSize',titleFontSize,...
    'Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);

% Robot frame plot velocity states
dxr = subplot(3,3,2);
hold on;
plot(tt(1:end-1),sim.yr(4,:),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(ttEst(1:end-1),est.rbt.m(4,:),...
    'm','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
plot(ttEst(1:end-1),est.rbt.x(4,:),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
```

```
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    ylabel('$\dot{x}_r\hspace{1mm}[m/s]$','Interpreter','latex');
    title('$Robot\hspace{1mm}body\hspace{1mm}\dot{x}_{r}\hspace{1mm}velocity$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
    dyr = subplot(3,3,5);
    hold on;
    plot(tt(1:end-1),sim.yr(5,:),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(ttEst(1:end-1),est.rbt.m(5,:),...
        'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
    plot(ttEst(1:end-1),est.rbt.m(5,:),...
        'm','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
    plot(ttEst(1:end-1),est.rbt.x(5,:),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    ylabel('$\dot{y}_r\hspace{1mm}[m/s]$','Interpreter','latex');
    title('$Robot\hspace{1mm}body\hspace{1mm}\dot{y}_{r}\hspace{1mm}velocity$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
    legend('$simulated\hspace{1mm}ground\hspace{1mm}truth$',...
        '$simulated\hspace{1mm}measurement$',...
        '$simulated\hspace{1mm}pseudo\hspace{1mm}measurement$',...
        '$estimator\hspace{1mm}output$',...
        'Interpreter','latex');                                             % plot legend here to
catch the pseudo-measurements
    dpsir = subplot(3,3,8);
    hold on;
    plot(tt(1:end-1),rad2deg(sim.yr(6,:)),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(ttImu(1:end-1),rad2deg(sen.imu.angVel(1,:)),...
        'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
    plot(ttEst(1:end-1),rad2deg(est.rbt.x(6,:)),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    ylabel('$\dot{\psi}_r\hspace{1mm}[rad/s]$','Interpreter','latex');
    title('$Robot\hspace{1mm}body\hspace{1mm}\dot{\psi}_{r}\hspace{1mm}velocity$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);

    % Robot frame plot acceleration states
    ddxr = subplot(3,3,3);
    hold on;
    plot(tt(1:end-1),sim.yr(7,:),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(ttImu(1:end-1),sen.imu.linAcc(1,:),...
        'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
    plot(ttEst(1:end-1),est.rbt.x(7,:),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    ylabel('$\ddot{x}_r\hspace{1mm}[m/s^2]$','Interpreter','latex');
    title('$Robot\hspace{1mm}body\hspace{1mm}\ddot{x}_{r}\hspace{1mm}acceleration$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
    ddyr = subplot(3,3,6);
    hold on;
    plot(tt(1:end-1),sim.yr(8,:),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(ttImu(1:end-1),sen.imu.linAcc(2,:),...
        'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
    plot(ttEst(1:end-1),est.rbt.x(8,:),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    ylabel('$\ddot{y}_r\hspace{1mm}[m/s^2]$','Interpreter','latex');
    title('$Robot\hspace{1mm}body\hspace{1mm}\ddot{y}_{r}\hspace{1mm}acceleration$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);

    % Figure properties
    linkaxes([xr,yr,psir,dxr,dyr,dpsir,ddxr,ddyr],'x');
    linkaxes([xr,yr],'y');
    linkaxes([dxr,dyr],'y');
    linkaxes([ddxr,ddyr],'y');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
    set(gcf,'Position',[0 0 1920 1280])

end

function plotMap(t,sim,sen,est,figN)

    % Set font size
    titleFontSize = 32;
    defaultFontSize = 28;
    markerSize = 10;

    % Map frame plot position states
    figure(figN);
    tt = 0:t.dt:t.end;
    ttGnss = 0:1/sen.gnss.rr:t.end;
    ttImu = 0:1/sen.imu.rr:t.end;
    ttEst = 0:est.dt:t.end;
    xm = subplot(3,3,1);
    hold on;
    plot(tt(1:end-1),sim.xm(1,:),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(ttGnss(1:end-1),sen.gnss.linPos(1,:),...
        'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/1.5);
    plot(ttEst(1:end-1),est.map.x(1,:),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    ylabel('$x_m\hspace{1mm}[m]$','Interpreter','latex');
    title('$Map\hspace{1mm}x_{m}\hspace{1mm}position$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
    ym = subplot(3,3,4);
    hold on;
    plot(tt(1:end-1),sim.xm(2,:),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(ttGnss(1:end-1),sen.gnss.linPos(2,:),...
        'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/1.5);
    plot(ttEst(1:end-1),est.map.x(2,:),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    ylabel('$y_m\hspace{1mm}[m]$','Interpreter','latex');
    title('$Map\hspace{1mm}y_{m}\hspace{1mm}position$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
    psim = subplot(3,3,7);
    hold on;
    plot(tt(1:end-1),rad2deg(sim.xm(3,:)),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
```

```
plot(ttImu(1:end-1),rad2deg(sen.imu.angPos(1,:)),...
    'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
plot(ttEst(1:end-1),rad2deg(est.map.x(3,:)),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
ylabel('$\psi_m\hspace{1mm}[rad]$','Interpreter','latex');
title('$Map\hspace{1mm}\psi_{m}\hspace{1mm}position$',...
    'FontSize',titleFontSize,...
    'Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);

% Robot frame plot velocity states
dxr = subplot(3,3,2);
hold on;
plot(tt(1:end-1),sim.yr(4,:),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(ttEst(1:end-1),est.map.x(4,:),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
ylabel('$\dot{x}_r\hspace{1mm}[m/s]$','Interpreter','latex');
title('$Robot\hspace{1mm}body\hspace{1mm}\dot{x}_{r}\hspace{1mm}velocity$',...
    'FontSize',titleFontSize,...
    'Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
dyr = subplot(3,3,5);
hold on;
plot(tt(1:end-1),sim.yr(5,:),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(ttEst(1:end-1),est.map.x(5,:),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
ylabel('$\dot{y}_r\hspace{1mm}[m/s]$','Interpreter','latex');
title('$Robot\hspace{1mm}body\hspace{1mm}\dot{y}_{r}\hspace{1mm}velocity$',...
    'FontSize',titleFontSize,...
    'Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
dpsir = subplot(3,3,8);
hold on;
plot(tt(1:end-1),rad2deg(sim.yr(6,:)),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(ttImu(1:end-1),rad2deg(sen.imu.angVel(1,:)),...
    'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
plot(ttEst(1:end-1),rad2deg(est.map.x(6,:)),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
ylabel('$\dot{\psi}_r\hspace{1mm}[m/s]$','Interpreter','latex');
title('$Robot\hspace{1mm}body\hspace{1mm}\dot{\psi}_{r}\hspace{1mm}velocity$',...
    'FontSize',titleFontSize,...
    'Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);

% Legend
legend('$simulated\hspace{1mm}ground\hspace{1mm}truth$',...
    '$simulated\hspace{1mm}measurement$',...
    '$estimator\hspace{1mm}output$',...
    'Interpreter','latex');

% Figure properties
linkaxes([xm,ym,psim,dxr,dyr,dpsir],'x');
linkaxes([xm,ym],'y');
linkaxes([dxr,dyr],'y');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
```

```
    set(gcf,'Position',[0 0 1920 1280])

end
```

8.2. Cascading, dual-frame, motion-tracking estimator Matlab ROS package source code

```
function gnssImuEstimatorRosNode()

    % Start ROS
    rosshutdown;
    rosinit;

    % Create ROS subscribers
    rosSubscribers();

    % Create ROS publishers
    [pub,msg] = rosPublishers();

    % Estimator setup
    est = estUserInputs();
    est = estSetup(est);

    % Sensor setup and initial collection
    sen = sensorSetup();

    % Recording loop
    disp('Listening for ros topics.  Use ctrl+C to exit.');
    while true

        % Take time at start of loop
        tLoop = tic;
        tLoopStart = toc(tLoop);

        % Collect measurements from each sensor
        sen = collectMeasurements(sen);

        % Run modified constant acceleration estimator
        est = runEst(est,sen);

        % Write to publishers
        write2pub(pub,msg,est,sen)

        % Take time at end of loop to calculate delay
        tLoopEnd = tLoopStart+toc(tLoop);
        tLoop = tLoopEnd-tLoopStart;
        loopDelay = est.dt-tLoop;
        if (loopDelay > 0)
            pause(loopDelay);
        end

        % Debug information
        verbose = false;
        if verbose == true
            disp(['Loop execution took ',num2str(tLoop*1000),...
                ' [ms], delaying ',num2str(loopDelay*1000),' [ms].']);
        end

    end

end

function sub = rosSubscribers()

    % Debug statement
    disp('Creating Matlab ROS subscribers...');
    % GNSS subscriber
    try
        sub.gnss = rossubscriber('/fix',@gnssCallback);
    catch
        disp(' ');
        disp('/fix (GNSS) topic is not publishing, code will throw error!');
    end

    % IMU data subscriber
    try
        sub.imu.data = rossubscriber('/imu/data',@imuDataCallback);
    catch
        disp(' ');
        disp('/imu/data (IMU) topic is not publishing, code will throw error!');
    end

    % Pause for first topic to publish
    pause(1);

    % Print debug statement
    disp('Matlab ROS subscribers created!');

end

function [pub,msg] = rosPublishers()

    % Debug statement
    disp('Creating Matlab ROS publishers...');

    % Create publishers
    pub.gnss.map.x = rospublisher('/est/gnss/map/x','std_msgs/Float32');
    pub.gnss.map.y = rospublisher('/est/gnss/map/y','std_msgs/Float32');
    pub.gnss.map.tz = rospublisher('/est/gnss/map/tz','std_msgs/Float32');
    pub.gnss.map.pose = rospublisher('/est/gnss/map/pose','geometry_msgs/Vector3');
    pub.gnss.map.tpp = rospublisher('/est/gnss/map/tpp','std_msgs/Float32');
    pub.gnss.rbt.x = rospublisher('/est/gnss/rbt/x','std_msgs/Float32');
    pub.gnss.rbt.y = rospublisher('/est/gnss/rbt/y','std_msgs/Float32');
    pub.imu.map.tz = rospublisher('/est/imu/map/tz','std_msgs/Float32');
    pub.imu.map.dtz = rospublisher('/est/imu/map/dtz','std_msgs/Float32');
    pub.imu.rbt.ddx = rospublisher('/est/imu/rbt/ddx','std_msgs/Float32');
    pub.imu.rbt.ddy = rospublisher('/est/imu/rbt/ddy','std_msgs/Float32');
    pub.est.map.x = rospublisher('/est/map/x','std_msgs/Float32');
    pub.est.map.y = rospublisher('/est/map/y','std_msgs/Float32');
    pub.est.map.tz = rospublisher('/est/map/tz','std_msgs/Float32');
    pub.est.map.pose = rospublisher('/est/map/pose','geometry_msgs/Vector3');
    pub.est.map.dtz = rospublisher('/est/map/dtz','std_msgs/Float32');
    pub.est.rbt.x = rospublisher('/est/rbt/x','std_msgs/Float32');
    pub.est.rbt.y = rospublisher('/est/rbt/y','std_msgs/Float32');
    pub.est.rbt.dx = rospublisher('/est/rbt/dx','std_msgs/Float32');
    pub.est.rbt.dy = rospublisher('/est/rbt/dy','std_msgs/Float32');
    pub.est.rbt.ddx = rospublisher('/est/rbt/ddx','std_msgs/Float32');
    pub.est.rbt.ddy = rospublisher('/est/rbt/ddy','std_msgs/Float32');
    pub.est.linVelAbs = rospublisher('/est/linVelAbs','std_msgs/Float32');
    pub.est.linVelSurge = rospublisher('/est/linVelSurge','std_msgs/Float32');
    pub.est.linVelSway = rospublisher('/est/linVelSway','std_msgs/Float32');

    % Create publisher messages
    msg.gnss.map.x = rosmessage(pub.gnss.map.x);
    msg.gnss.map.y = rosmessage(pub.gnss.map.y);
    msg.gnss.map.tz = rosmessage(pub.gnss.map.tz);
    msg.gnss.map.pose = rosmessage(pub.gnss.map.pose);
    msg.gnss.map.tpp = rosmessage(pub.gnss.map.tpp);
    msg.gnss.rbt.x = rosmessage(pub.gnss.rbt.x);
    msg.gnss.rbt.y = rosmessage(pub.gnss.rbt.y);
    msg.imu.map.tz = rosmessage(pub.imu.map.tz);
    msg.imu.map.dtz = rosmessage(pub.imu.map.dtz);
    msg.imu.rbt.ddx = rosmessage(pub.imu.rbt.ddx);
    msg.imu.rbt.ddy = rosmessage(pub.imu.rbt.ddy);
    msg.est.map.x = rosmessage(pub.est.map.x);
    msg.est.map.y = rosmessage(pub.est.map.y);
    msg.est.map.tz = rosmessage(pub.est.map.tz);
    msg.est.map.pose = rosmessage(pub.est.map.pose);
    msg.est.map.dtz = rosmessage(pub.est.map.dtz);
    msg.est.rbt.x = rosmessage(pub.est.rbt.x);
    msg.est.rbt.y = rosmessage(pub.est.rbt.y);
    msg.est.rbt.dx = rosmessage(pub.est.rbt.dx);
    msg.est.rbt.dy = rosmessage(pub.est.rbt.dy);
    msg.est.rbt.ddx = rosmessage(pub.est.rbt.ddx);
    msg.est.rbt.ddy = rosmessage(pub.est.rbt.ddy);
    msg.est.linVelAbs = rosmessage(pub.est.linVelAbs);
    msg.est.linVelSurge = rosmessage(pub.est.linVelSurge);
    msg.est.linVelSway = rosmessage(pub.est.linVelSway);

    % Instantiate blank message
    msg.gnss.map.x.Data = 0;
    msg.gnss.map.y.Data = 0;
    msg.gnss.map.tz.Data = 0;
    msg.gnss.map.pose.X = 0;
    msg.gnss.map.pose.Y = 0;
    msg.gnss.map.pose.Z = 0;
    msg.gnss.map.tpp.Data = 0;
    msg.gnss.rbt.x.Data = 0;
    msg.gnss.rbt.y.Data = 0;
    msg.imu.map.tz.Data = 0;
    msg.imu.map.dtz.Data = 0;
    msg.imu.rbt.ddx.Data = 0;
    msg.imu.rbt.ddy.Data = 0;
    msg.est.map.x.Data = 0;
    msg.est.map.y.Data = 0;
    msg.est.map.tz.Data = 0;
    msg.est.map.pose.X = 0;
    msg.est.map.pose.Y = 0;
    msg.est.map.pose.Z = 0;
    msg.est.map.dtz.Data = 0;
    msg.est.rbt.x.Data = 0;
    msg.est.rbt.y.Data = 0;
    msg.est.rbt.dx.Data = 0;
    msg.est.rbt.dy.Data = 0;
    msg.est.rbt.ddx.Data = 0;
    msg.est.rbt.ddy.Data = 0;
    msg.est.linVelAbs.Data = 0;
    msg.est.linVelSurge.Data = 0;
    msg.est.linVelSway.Data = 0;

    % Publish first message to topics
    send(pub.gnss.map.x,msg.gnss.map.x);
    send(pub.gnss.map.y,msg.gnss.map.y);
    send(pub.gnss.map.tz,msg.gnss.map.tz);
    send(pub.gnss.map.pose,msg.gnss.map.pose);
    send(pub.gnss.map.tpp,msg.gnss.map.tpp);
    send(pub.gnss.rbt.x,msg.gnss.rbt.x);
    send(pub.gnss.rbt.y,msg.gnss.rbt.y);
    send(pub.imu.map.tz,msg.imu.map.tz);
    send(pub.imu.map.dtz,msg.imu.map.dtz);
    send(pub.imu.rbt.ddx,msg.imu.rbt.ddx);
    send(pub.imu.rbt.ddy,msg.imu.rbt.ddy);
    send(pub.est.map.x,msg.est.map.x);
    send(pub.est.map.y,msg.est.map.y);
    send(pub.est.map.tz,msg.est.map.tz);
    send(pub.est.map.pose,msg.est.map.pose);
    send(pub.est.map.dtz,msg.est.map.dtz);
    send(pub.est.rbt.x,msg.est.rbt.x);
    send(pub.est.rbt.y,msg.est.rbt.y);
    send(pub.est.rbt.dx,msg.est.rbt.dx);
    send(pub.est.rbt.dy,msg.est.rbt.dy);
    send(pub.est.rbt.ddx,msg.est.rbt.ddx);
    send(pub.est.rbt.ddy,msg.est.rbt.ddy);
    send(pub.est.linVelAbs,msg.est.linVelAbs);
    send(pub.est.linVelSurge,msg.est.linVelSurge);
    send(pub.est.linVelSway,msg.est.linVelSway);
```

```
end

function est = estUserInputs()

    % USER INPUT: time step properties
    est.dt = 0.025;   % how quickly estimator should update [s]

    % USER INPUT: Number of past sensor updates to store for
    % calculating the incremental moving mean of sensor data
    % (if supported).
    est.linPos.n = 2;    % linear position mean array size (must be 2 or more)
    est.angPos.n = 2;    % angular position mean array size (must be 2 or more)
    est.angVel.n = 2;    % angular velocity mean array size (must be 2 or more)
    est.linAcc.n = 2;    % linear acceleration mean array size (must be 2 or more)

    % USER INPUT: sliding adaptive covariance gain
    est.gnssSlidingCovariance = 1;  % 1 for GNSS sliding covariance gain, 0 for binary covariance
gain calculation
    est.imuSlidingCovariance = 0;   % 1 for IMU sliding covariance gain, 0 for binary covariance
gain calculation

    % USER INPUT: decide which sensors to use for pseudo linear velocity
    % calculation
    est.useGnssLinVel = 1;  % set to 1 to use GPS forward difference, set to 0 to ignore
    est.useImuLinVel = 1;   % set to 1 to use IMU trapezoidal method, set to 0 to ignore
    est.linVelBias = 1.35;  % bias (multiplier) on linear velocity output

    % USER INPUT: robot body frame estimator measurement noise covariance
    est.rbt.R.linPos = 2;       % linear position [m]
    est.rbt.R.angPos = 0.075;   % angular position [rad]
    est.rbt.R.linVel = 2;       % linear velocity [m/s]
    est.rbt.R.angVel = 0.075;   % angular velocity [rad/s]
    est.rbt.R.linAcc = 0.25;    % linear acceleration [m/s^s]
    est.rbt.R.angAcc = 0.5;     % angular acceleration [rad/s^2] (not measured, so this is
actually irrelevant)

    % USER INPUT: robot body frame estimator process noise covariance
    est.rbt.Q.linPos = est.rbt.R.linPos*1E1;    % linear position [m]
    est.rbt.Q.angPos = est.rbt.R.angPos*1E1;    % angular position [rad]
    est.rbt.Q.linVel = est.rbt.R.linVel*1E1;    % linear velocity [m/s] (not measured, so this is
actually irrelevant)
    est.rbt.Q.angVel = est.rbt.R.angVel*1E-2;   % angular velocity [rad/s]
    est.rbt.Q.linAcc = est.rbt.R.linAcc*5E-2;   % linear acceleration [m/s^s]
    est.rbt.Q.angAcc = est.rbt.R.angAcc*5E-2;   % angular acceleration [rad/s^2] (not measured,
so this is actually irrelevant)

    % USER INPUT: robot body frame observation/observability
    est.rbt.H.linPos = 1;   % linear position (measured)
    est.rbt.H.angPos = 1;   % angular position (measured)
    est.rbt.H.linVel = 1;   % linear velocity (pseudo-measured)
    est.rbt.H.angVel = 1;   % angular velocity (measured)
    est.rbt.H.linAcc = 1;   % linear acceleration (measured)
    est.rbt.H.angAcc = 0;   % angular acceleration (NOT measured)

    % USER INPUT: hybrid map frame estimator measurement noise covariance
    est.map.R.linPos = 2;       % linear position [m]
    est.map.R.angPos = 0.075;   % angular position [rad]
    est.map.R.linVel = 1;       % linear velocity [m/s] (not measured, so this is actually
irrelevant)
    est.map.R.angVel = 0.075;   % angular velocity [rad/s]

    % USER INPUT: robot body frame estimator process noise covariance
    est.map.Q.linPos = est.rbt.R.linPos*1E1;    % linear position [m]
    est.map.Q.angPos = est.rbt.R.angPos*1E5;    % angular position [rad] (set to be arbitrarily
large)
    est.map.Q.linVel = est.rbt.R.linVel*1E5;    % linear velocity [m/s] (set to be arbitrarily
large)
    est.map.Q.angVel = est.rbt.R.angVel*1E5;    % angular velocity [rad/s] (set to be arbitrarily
large)
```

```
    % USER INPUT: robot body frame observation/observability
    est.map.H.linPos = 1;   % linear position (measured)
    est.map.H.angPos = 1;   % angular position (measured)
    est.map.H.linVel = 1;   % linear velocity (measured)
    est.map.H.angVel = 1;   % angular velocity (measured)

end

function est = estSetup(est)

    % Instantiate measurement variables and arrays
    est.gnssSlidingGain = 1;                        % sliding covariance gain for GPS
    est.imuSlidingGain = 1;                         % sliding covariance gain for IMU
    est.gnssImuSlidingGain = 1;                     % sliding covariance gain for GPS+IMU
    est.slidingGain = 1;                            % final sliding covariance gain for estimator
iteration
    est.linPosMapArray = zeros(2,2);                % linear position directly from GPS in map
frame measurement array
    est.linPosRbtArray = zeros(2,est.linPos.n);     % linear position directly from GPS in robot
frame measurement array
    est.angPosArray = zeros(1,est.angPos.n);        % angular position directly from IMU in map
and robot frame measurement array
    est.linVelGnssAprxRbt = zeros(2,1);             % approximated linear velocity in robot frame
from forward-difference of GPS
    est.linVelImuAprxRbt = zeros(2,1);              % approximated linear velocity in robot frame
from trapezoidal method of IMU
    est.angVelArray = zeros(1,est.angVel.n);        % angular velocity directly from IMU in map
and robot frame measurement array
    est.linAccRbtArray = zeros(2,est.linAcc.n);     % linear acceleration directly from IMU in
robot frame measurement array

    % Instantiate estimator matrices
    est.rbt.m = zeros(9,1);     % robot frame estimator measurement matrix
    est.rbt.x = zeros(9,1);     % robot frame estimator state estimate
    est.rbt.L = zeros(9,9);     % robot frame estimator kalman gain matrix
    est.rbt.P = zeros(9,9);     % robot frame estimator covariance matrix
    est.rbt.s = zeros(9,9);     % robot frame sliding covariance gain matrix
    est.map.m = zeros(6,1);     % map frame estimator measurement matrix
    est.map.x = zeros(6,1);     % map frame estimator state estimate
    est.map.L = zeros(6,6);     % map frame estimator kalman gain matrix
    est.map.P = zeros(6,6);     % map frame estimator covariance matrix
    est.map.s = zeros(6,6);     % map frame sliding covariance gain matrix

    % Robot frame covariance, obvervation, state and input matrices
    est.rbt.Q = [...
        est.rbt.Q.linPos 0 0 0 0 0 0 0 0;...
        0 est.rbt.Q.linPos 0 0 0 0 0 0 0;...
        0 0 est.rbt.Q.angPos 0 0 0 0 0 0;...
        0 0 0 est.rbt.Q.linVel 0 0 0 0 0;...
        0 0 0 0 est.rbt.Q.linVel 0 0 0 0;...
        0 0 0 0 0 est.rbt.Q.angVel 0 0 0;...
        0 0 0 0 0 0 est.rbt.Q.linAcc 0 0;...
        0 0 0 0 0 0 0 est.rbt.Q.linAcc 0;...
        0 0 0 0 0 0 0 0 est.rbt.Q.angAcc];      % process noise covariance matrix
    est.rbt.R = [...
        est.rbt.R.linPos 0 0 0 0 0 0 0 0;...
        0 est.rbt.R.linPos 0 0 0 0 0 0 0;...
        0 0 est.rbt.R.angPos 0 0 0 0 0 0;...
        0 0 0 est.rbt.R.linVel 0 0 0 0 0;...
        0 0 0 0 est.rbt.R.linVel 0 0 0 0;...
        0 0 0 0 0 est.rbt.R.angVel 0 0 0;...
        0 0 0 0 0 0 est.rbt.R.linAcc 0 0;...
        0 0 0 0 0 0 0 est.rbt.R.linAcc 0;...
        0 0 0 0 0 0 0 0 est.rbt.R.angAcc];      % measurement noise covariance matrix
    est.rbt.H = [...
        est.rbt.H.linPos 0 0 0 0 0 0 0 0;...
        0 est.rbt.H.linPos 0 0 0 0 0 0 0;...
        0 0 est.rbt.H.angPos 0 0 0 0 0 0;...
        0 0 0 est.rbt.H.linVel 0 0 0 0 0;...
        0 0 0 0 est.rbt.H.linVel 0 0 0 0;...
        0 0 0 0 0 est.rbt.H.angVel 0 0 0;...
        0 0 0 0 0 0 est.rbt.H.linAcc 0 0;...
        0 0 0 0 0 0 0 est.rbt.H.linAcc 0;...
        0 0 0 0 0 0 0 0 est.rbt.H.angAcc];      % obervation matrix
    est.rbt.A = [...
        1 0 0 est.dt 0 0 0 0 0;...
        0 1 0 0 est.dt 0 0 0 0;...
        0 0 1 0 0 est.dt 0 0 0;...
        0 0 0 1 0 0 0 0 0;...
        0 0 0 0 1 0 0 0 0;...
        0 0 0 0 0 1 0 0 0;...
        0 0 0 0 0 0 1 0 0;...
        0 0 0 0 0 0 0 1 0;...
        0 0 0 0 0 0 0 0 1];                 % state matrix
    est.rbt.B = zeros(9,3);                 % input matrix

    % Map frame covariance, obvervation, state and input matrices
    est.map.Q = [...
        est.map.Q.linPos 0 0 0 0 0;...
        0 est.map.Q.linPos 0 0 0 0;...
        0 0 1E5 0 0 0;...
        0 0 0 1E5 0 0;...
        0 0 0 0 1E5 0;...
        0 0 0 0 0 1E5];                     % process noise covariance matrix
    est.map.R = [...
        est.map.R.linPos 0 0 0 0 0;...
        0 est.map.R.linPos 0 0 0 0;...
        0 0 est.map.R.angPos 0 0 0;...
        0 0 0 est.map.R.linVel 0 0;...
        0 0 0 0 est.map.R.linVel 0;...
        0 0 0 0 0 est.map.R.angVel];        % measurement noise covariance matrix
    est.map.H = [...
        est.map.H.linPos 0 0 0 0 0;...
        0 est.map.H.linPos 0 0 0 0;...
        0 0 est.map.H.angPos 0 0 0;...
        0 0 0 est.map.H.linVel 0 0;...
        0 0 0 0 est.map.H.linVel 0;...
        0 0 0 0 0 est.map.H.angVel];        % obervation matrix
    est.map.A = @(x) [...
        1 0 0 cos(x(3))*est.dt -sin(x(3))*est.dt 0;...
        0 1 0 sin(x(3))*est.dt cos(x(3))*est.dt 0;...
        0 0 1 0 0 est.dt;...
        0 0 0 1 0 0;...
        0 0 0 0 1 0;...
        0 0 0 0 0 1];                   % state matrix
    est.map.B = zeros(6,3);             % input matrix

    % Warning message
    if (est.useGnssLinVel == 0 && est.useImuLinVel == 1)
        disp('Warning: you have elected to estimate linear velocity only using IMU measurements,
this simulation cannot replicate the real-world drift of IMU sensors, so this simulation will
return non-sensical linear velocity accuracy');
    elseif (est.useGnssLinVel == 0 && est.useImuLinVel == 0)
        disp('Error with linear velocity pseudo-measurement sensor selection. Simulation will
return non-sensical result.');
    end

end

function sen = sensorSetup()

    % Sensor variables
    sen.gnss.rr = 1;            % approximate refresh rate of GNSS
    sen.imu.rr = 20;            % approximate refresh rate of IMU

    % Instantiate GNSS sensor storage variables
    sen.gnss.time = 0;          % last GNSS update time from ROS
    sen.gnss.timeStore = 0;     % last known GNSS update time in Matlab (used to check sensor
update)
```

```
    sen.gnss.tslu = 0;           % time since last GNSS update
    sen.gnss.update = 0;         % GNSS sensor update flag
    sen.gnss.lat = 0;            % GNSS latitude
    sen.gnss.lon = 0;            % GNSS longitude
    sen.gnss.alt = 0;            % GNSS altitude
    sen.gnss.x = 0;              % GNSS linear position along x
    sen.gnss.y = 0;              % GNSS linear position along y
    sen.gnss.z = 0;              % GNSS linear position along z
    sen.gnss.xLast = 0;          % previous GNSS linear position along x (for frame conversion)
    sen.gnss.yLast = 0;          % previous GNSS linear position along y (for frame conversion)
    sen.gnss.tpp = 0;            % angle of line connecting GNSS linear position (for frame
conversion)
    sen.gnss.tzUnwrap = 0;       % IMU angular position about z during GNSS update (for frame
conversion)
    sen.gnss.tzUnwrapLast = 0;   % IMU angular position about z during previous GNSS update (for
frame conversion)
    sen.gnss.xr = 0;             % GNSS linear position converted to robot frame x (surge)
    sen.gnss.yr = 0;             % GNSS linear position converted to robot frame y (sway)

    % Instantiate IMU sensor storage variables
    sen.imu.time = 0;            % last IMU update time from ROS
    sen.imu.timeStore = 0;       % last known IMU update time in Matlab (used to check sensor
update
    sen.imu.tslu = 0;            % time since last IMU update
    sen.imu.update = 0;          % IMU sensor update flag
    sen.imu.tx = 0;              % IMU angular position about x
    sen.imu.ty = 0;              % IMU angular position about y
    sen.imu.tz = 0;              % IMU angular position about z
    sen.imu.dtx = 0;             % IMU angular velocity about x
    sen.imu.dty = 0;             % IMU angular velocity about y
    sen.imu.dtz = 0;             % IMU angular velocity about z
    sen.imu.ddx = 0;             % IMU linear acceleration along x
    sen.imu.ddy = 0;             % IMU linear acceleration along y
    sen.imu.ddz = 0;             % IMU linear acceleration along z
    sen.imu.txLast = 0;          % IMU last angle about x (for unwrapping)
    sen.imu.tyLast = 0;          % IMU last angle about y (for unwrapping)
    sen.imu.tzLast = 0;          % IMU last angle about z (for unwrapping)
    sen.imu.txAddAngle = 0;      % IMU angle about x tracking variable (for unwrapping)
    sen.imu.tyAddAngle = 0;      % IMU angle about y tracking variable (for unwrapping)
    sen.imu.tzAddAngle = 0;      % IMU angle about z tracking variable (for unwrapping)
    sen.imu.txUnwrap = 0;        % IMU unwrapped angle about x (for unwrapping)
    sen.imu.tyUnwrap = 0;        % IMU unwrapped angle about y (for unwrapping)
    sen.imu.tzUnwrap = 0;        % IMU unwrapped angle about z (for unwrapping)

    % Global variables to grab from subscriber callbacks
    global gnssSec gnssNsec
    global gnssLat gnssLon gnssAlt
    global imutz imuTic

    % Get a starting time so measurements start reasonably close to zero
    sen.t0 = toc(imuTic);

    % Get a starting GNSS coordinate so measurements start reasonably close
    % to zero
    sen.gnss.lat0 = gnssLat;
    sen.gnss.lon0 = gnssLon;
    sen.gnss.alt0 = gnssAlt;
    sen.gnss.time = gnssSec+gnssNsec*1E-9-sen.t0;
    sen.gnss.timeStore = sen.gnss.time;

    % Get starting IMU orientation for frame conversion
    sen.imu.tz = imutz;
    sen.gnss.tz = sen.imu.tz;

end

function sen = collectMeasurements(sen)

    % Very brief pause to allow callbacks to execute
    pause(0.001);   % 1 ms pause
    % Global variables to grab from subscriber callbacks
    global gnssSec gnssNsec
    global gnssLat gnssLon gnssAlt
    global imuSec imuNsec
    global imutx imuty imutz
    global imudtx imudty imudtz
    global imuddx imuddy imuddz

    % Get current GNSS and IMU time
    sen.gnss.time = gnssSec+gnssNsec*1E-9-sen.t0;   % last GNSS update time from ROS
    sen.imu.time = imuSec+imuNsec*1E-9-sen.t0;      % last IMU update time from ROS

    % Check if a GNSS update occured based on last known sensor update time
    if sen.gnss.timeStore ~= sen.gnss.time

        % A change occured, set timeStore to last known update time, and
        % set update flag to 1
        sen.gnss.timeStore = sen.gnss.time;
        sen.gnss.update = 1;

        % Save the previouly GNSS x and y datapoints in the map frame
        sen.gnss.xLast = sen.gnss.x;
        sen.gnss.yLast = sen.gnss.y;

        % Save measurements from subscriber callbacks to local variables
        sen.gnss.lat = gnssLat;     % GNSS latitude
        sen.gnss.lon = gnssLon;     % GNSS longitude
        sen.gnss.alt = gnssAlt;     % GNSS altitude

        % Convert lat-lon-alt to flat-earth coodrinates
        lla = [sen.gnss.lat,sen.gnss.lon,sen.gnss.alt];
        llo = [sen.gnss.lat0,sen.gnss.lon0];
        flat = lla2flat(lla,llo,0,-sen.gnss.alt0);

        % Save measurements from subscriber callbacks to local variables
        sen.gnss.tzUnwrapLast = sen.gnss.tzUnwrap;
        sen.gnss.x = flat(1,2);     % MATLAB GIVES THE FLAT EARTH VECTOR IN [Y,X,Z] FOR SOME
***** REASON
        sen.gnss.y = flat(1,1);
        sen.gnss.z = flat(1,3);
        sen.gnss.tzUnwrap = sen.imu.tzUnwrap;

    else

        sen.gnss.update = 0;

    end

    % Check if an IMU update occured based on last known sensor update time
    if sen.imu.timeStore ~= sen.imu.time

        % A change occured, set timeStore to last known update time, and
        % set update flag to 1
        sen.imu.timeStore = sen.imu.time;
        sen.imu.update = 1;

        % Save measurements from subscriber callbacks to local variables
        sen.imu.txLast = sen.imu.tx;
        sen.imu.tyLast = sen.imu.ty;
        sen.imu.tzLast = sen.imu.tz;
        sen.imu.tx = imutx;
        sen.imu.ty = imuty;
        sen.imu.tz = imutz;
        sen.imu.dtx = imudtx;
        sen.imu.dty = imudty;
        sen.imu.dtz = imudtz;
        sen.imu.ddx = imuddx;
        sen.imu.ddy = imuddy;
        sen.imu.ddz = imuddz;
        % Unwrap angles
        [sen.imu.txUnwrap,sen.imu.txAddAngle] =...
            unwrapAngle(sen.imu.tx,sen.imu.txLast,sen.imu.txAddAngle);
        [sen.imu.tyUnwrap,sen.imu.tyAddAngle] =...
            unwrapAngle(sen.imu.ty,sen.imu.tyLast,sen.imu.tyAddAngle);
        [sen.imu.tzUnwrap,sen.imu.tzAddAngle] =...
            unwrapAngle(sen.imu.tz,sen.imu.tzLast,sen.imu.tzAddAngle);

    else

        sen.imu.update = 0;

    end

    % Convert GNSS to robot frame (this MUST occur after previous steps)
    if sen.gnss.update == 1

        [xrf,yrf] = map2rbt(...
            sen.gnss.x,...
            sen.gnss.y,...
            sen.gnss.xLast,...
            sen.gnss.yLast,...
            sen.gnss.tzUnwrap,...
            sen.gnss.xr,...
            sen.gnss.yr);

        sen.gnss.xr = xrf;
        sen.gnss.yr = yrf;

    end

end

function [xrf,yrf] = map2rbt(xmf,ymf,xm0,ym0,psi0,xr0,yr0)
    % Converts pose in the map frame to pose in the robot frame

    % Calculate translations and rotations in robot frame
    Txm = xmf-xm0;
    Tym = ymf-ym0;

    % Calculate intermediate length and angle
    li = sqrt(Txm^2+Tym^2);
    psii = atan2(ymf-ym0,xmf-xm0);

    % Calculate translations and rotations in robot frame
    Txr = cos(psii-psi0)*li;
    Tyr = sin(psii-psi0)*li;

    % Calculate individual components in the robot frame
    xrf = xr0+Txr;
    yrf = yr0+Tyr;

end

function [tUnwrap,tAdd] = unwrapAngle(t,tLast,tAdd)

    if (abs(t-tLast)) > (3/4)*pi
        if (t > tLast)
            tAdd = tAdd-2*pi;
        else
            tAdd = tAdd+2*pi;
        end
    end
    tUnwrap = t+tAdd;

end

function est = runEst(est,sen)
```

```
    % Fill estimator measurement arrays with sensor data
    est = estMeasArrays(est,sen);

    % Calculate estimator sliding covariances based on sensor time data
    [est,sen] = estSlidingCovGain(est,sen);

    % Fill robot-frame estimator measurement and sliding covariance matrix
    est = estRbtMeas(est,sen);

    % Run robot-frame estimator
    est = runRbtKal(est);

    % Fill map-frame estimator measurement and sliding covariance matrix
    est = estMapMeas(est);

    % Run map-frame estimator
    est = runMapKal(est);

end

function est = estMeasArrays(est,sen)

    % Update GNSS sensor array if an sensor update occured
    if sen.gnss.update == 1

        % Set estimator last known sensor update to last known
        % sensor update
        est.gpsLastUpdateTime = sen.gnss.timeStore;

        % Update the linear position in map frame measurement array
        est.linPosMapArray(:,2:end) = est.linPosMapArray(:,1:end-1);
        est.linPosMapArray(:,1) = [sen.gnss.x;sen.gnss.y];

        % Update the linear position in robot frame measurement array
        est.linPosRbtArray(:,2:end) = est.linPosRbtArray(:,1:end-1);
        est.linPosRbtArray(:,1) = [sen.gnss.xr;sen.gnss.yr];

    end

    % Update IMU sensor array if an sensor update occured
    if sen.imu.update == 1

        % Set estimator last known sensor update to last known
        % sensor update
        est.gpsLastUpdateTime = sen.imu.timeStore;

        % Update the angular position (in map and robot frame) measurement array
        est.angPosArray(1,2:end) = est.angPosArray(1,1:end-1);
        est.angPosArray(1,1) = sen.imu.tzUnwrap;

        % Update the angular velocity (in map and robot frame) measurement array
        est.angVelArray(1,2:end) = est.angVelArray(1,1:end-1);
        est.angVelArray(1,1) = sen.imu.dtz;

        % Update the linear acceleration in robot frame measurement array
        est.linAccRbtArray(:,2:end) = est.linAccRbtArray(:,1:end-1);
        est.linAccRbtArray(:,1) = [sen.imu.ddx;sen.imu.ddy];

    end

end

function [est,sen] = estSlidingCovGain(est,sen)

%     % Sliding covariance profiling gain.
%     gainDegree = 5;                             % orders of magnitude increase in maximized
covariance gain
%     gnssSmoothTimeEnd = (1/sen.gnss.rr)*0.15;   % when the smoothing time should end (squared
decrease of variance to 1)
```

```
%     gnssScaleTimeEnd = (1/sen.gnss.rr)*0.2;     % when the scaling time should end (exponential
growth of variance from 1)
%     imuSmoothTimeEnd = (1/sen.imu.rr)*0.15;     % when the smoothing time should end (squared
decrease of variance to 1)
%     imuScaleTimeEnd = (1/sen.imu.rr)*0.2;       % when the scaling time should end (exponential
growth of variance from 1)

    % Time calculations
    dtGnss = 1/sen.gnss.rr;         % time step of a single GNSS sensor update
    dtImu = 1/sen.imu.rr;           % time step of a single IMU sensor update
    global gnssTic imuTic
    sen.gnss.tslu = toc(gnssTic);
    sen.imu.tslu = toc(imuTic);

    % GNSS sliding covariance matrix variables
    Gm = 10^5;                              % maximum covariance gain
    tscale = 0.15*dtGnss;                   % length of scaling region [s]
    tsmooth = 0.1*dtGnss;                   % length of scaling region [s]
    halfLifeSmooth = 0.1*tsmooth*log(2);    % half life of exponential decay of smoothing curve
    halfLifeScale = 0.1*tscale*log(2);      % half life of exponential decay expressed as a
percentage of the scale time

    % GNSS sliding covariance gain calculation
    if (est.gnssSlidingCovariance == 1)
        % GNSS smoothing region
        if (sen.gnss.tslu >= 0 & sen.gnss.tslu < tsmooth)
            est.gnssSlidingGain = Gm*2.^...
                (-sen.gnss.tslu*(1/halfLifeSmooth))+1;
        % GNSS scaling region
        elseif (sen.gnss.tslu >= tsmooth & sen.gnss.tslu <= tsmooth+tscale)
            est.gnssSlidingGain = Gm*2.^...
                ((sen.gnss.tslu-(tscale+tsmooth))*(1/halfLifeScale))+1;
        % GNSS rejected region
        else
            est.gnssSlidingGain = Gm;
        end
    % GNSS binary adaptive covariance gain
    else
        if sen.gnss.update == 1
            est.gnssSlidingGain = 1;
        else
            est.gnssSlidingGain = Gm;
        end
    end

    % Note that if the refresh rate of the estimator should be at least an
    % order of magnitude faster than the IMU refresh rate, otherwise you
    % will run into discrete-time rounding issues. If this occurs, use
    % binary covariance profiling for IMU.

    % IMU sliding covariance matrix variables
    tscale = 0.15*dtImu;                    % length of scaling region [s]
    tsmooth = 0.05*dtImu;                   % length of scaling region [s]
    halfLifeSmooth = 0.1*tsmooth*log(2);    % half life of exponential decay of smoothing curve
    halfLifeScale = 0.1*tscale*log(2);      % half life of exponential decay expressed as a
percentage of the scale time

    % IMU sliding covariance gain calculation
    if (est.imuSlidingCovariance == 1)
        % IMU smoothing region
        if (est.imuSlidingGain >= 0 & est.imuSlidingGain < imuSmoothTimeEnd)
            est.imuSlidingGain = Gm*2.^...
                (-sen.imu.tslu*(1/halfLifeSmooth))+1;
        % IMU scaling region
        elseif (est.imuSlidingGain >= imuSmoothTimeEnd & est.imuSlidingGain <= imuScaleTimeEnd)
            est.imuSlidingGain = Gm*2.^...
                ((sen.imu.tslu-(tscale+tsmooth))*(1/halfLifeScale))+1;
        % IMU rejected region
        else
            est.imuSlidingGain = Gm;
        end
    % IMU binary adaptive covariance gain calculation
    else
        if sen.imu.update == 1
            est.imuSlidingGain = 1;
        else
            est.imuSlidingGain = Gm;
        end
    end

end

function est = estRbtMeas(est,sen)

    % Set sliding covariance gain matrix to identity
    est.rbt.s = eye(9,9);                              % robot frame sliding covariance gain
matrix

    % Linear position measurements and gains
    est.rbt.m(1:2,1) = est.linPosRbtArray(1:2,1);      % linear positions in robot frame
    est.rbt.s(1:2,:) =...
        est.rbt.R(1:2,:)*est.gnssSlidingGain;          % linear positions in sliding covariance
gain matrix

    % Angular position measurements and gains
    est.rbt.m(3,1) = est.angPosArray(1,1);             % angular position in robot frame
    est.rbt.s(3,:) =...
        est.rbt.R(3,:)*est.imuSlidingGain;             % angular position in sliding covariance
gain matrix

    % Linear velocity pseudo measurements and gains
    est = estRbtLinVel(est,sen);                       % pseudo linear velocity measurements

    % Angular velocity measurements
    est.rbt.m(6,1) = est.angVelArray(1,1);             % angular velocity in robot frame
    est.slidingGain = est.imuSlidingGain;
    est.rbt.s(6,:) = ...
        est.rbt.R(6,:)*est.imuSlidingGain;             % angular velocity in sliding covariance
gain matrix

    % Linear acceleration measurements
    est.rbt.m(7:8,1) = est.linAccRbtArray(1:2,1);      % linear accelerations in robot frame
    est.slidingGain = est.imuSlidingGain;
    est.rbt.s(7:8,:) =...
        est.rbt.R(7:8,:)*est.imuSlidingGain;           % linear accelerations in sliding
covariance gain matrix

end

function est = estRbtLinVel(est,sen)

    % Take only the forward-difference of linear positon data from
    % GNSS in the robot frame
    if (est.useGnssLinVel == 1 && est.useImuLinVel == 0)

        % First, calculate the forward-difference of GNSS linear positon
        % data
        est = fdGnss(est,sen);                         % linear velocity pseudo-measurement

        % Second, calculate the variance of forward-difference of GNSS
        % linear positon data
        vardxG = (sqrt(2)/((1/sen.gnss.rr)^2))*...
            est.rbt.R(1,1)*est.gnssSlidingGain;        % variance of linear velocity pseudo-
measurements along x due to forward difference of GNSS
        vardyG = (sqrt(2)/((1/sen.gnss.rr)^2))*...
            est.rbt.R(2,2)*est.gnssSlidingGain;        % variance of linear velocity pseudo-
measurements along y due to forward difference of GNSS

        % Third, save to measurement vector
        est.rbt.m(4:5,1) =...
```

```
            est.linVelGnssAprxRbt(:,1)*est.linVelBias;  % save robot frame pseudo-measurement to
robot frame measurement matrix

        % Fourth, save to revised measurement noise covariance matrix
        % based on sliding adaptive covariance gain
        est.slidingGain = est.gnssSlidingGain;
        est.rbt.s(4:5,:) = [...
            0 0 0 vardxG 0 0 0 0 0;...
            0 0 0 0 vardyG 0 0 0 0];                      % linear velocity velocity terms in
revised measurement noise covariance matrix

    % Takes only the trapezoidal method of the linear acceleration
    % data from IMU in the map and robot frames
    elseif (est.useGnssLinVel == 0 && est.useImuLinVel == 1)

        % First, calculate the trapezoidal method of IMU linear
        % acceleration data
        est = tmImu(est,sen);

        % Second, calculate the variance of trapezoidal method of IMU
        % linear acceleration data
        vardxI = 1;     % nonsensical variance due to integration summation
        vardyI = 1;     % nonsensical variance due to integration summation

        % Third, save to measurement vector
        est.rbt.m(4:5,1) =...
            est.linVelImuAprxRbt(:,1)*est.linVelBias;   % save robot frame pseudo-measurement to
measurement matrix

        % Fourth, save to revised measurement noise covariance
        % matrix based on sliding adaptive covariance gain
        est.rbt.s(4:5,:) = [...
            0 0 0 vardxI 0 0 0 0 0;...
            0 0 0 0 vardyI 0 0 0 0];                      % linear velocities in revised
measurement noise covariance matrix based on sliding adaptive covariance gain

    % Takes both forward-difference of linear position data
    % from GNSS in robot frame AND trapezoidal method from IMU in robot
    % frame
    elseif (est.useGnssLinVel == 1 && est.useImuLinVel == 1)

        % First, calculate the forward-difference of GNSS linear positon
        % data and trapezoidal method of IMU linear acceleration data
        est = fdGnss(est,sen);      % linear velocity pseudo-measurement
        est = tmImu(est,sen);       % linear velocity pseudo-measurement

        % Second, calculate variance of pseudo-measurements based
        % on forward-difference of GNSS linear position data and on
        % trapezoidal method of IMU linear acceleration data
        % (going to ignore user values in the R matrix, since this
        % should be derived)
        vardxG =(sqrt(2)/((1/sen.gnss.rr)^2))*...
            est.rbt.R(1,1)*est.gnssSlidingGain;            % variance of linear velocity pseudo-
measurements along x due to forward difference of GNSS
        vardyG = (sqrt(2)/((1/sen.gnss.rr)^2))*...
            est.rbt.R(2,2)*est.gnssSlidingGain;            % variance of linear velocity pseudo-
measurements along y due to forward difference of GNSS
        vardx = (vardxG^2)/(2*vardxG);                     % variance of linear velocity pseudo-
measurements along x
        vardy = (vardyG^2)/(2*vardyG);                     % variance of linear velocity pseudo-
measurements along y

        % Third, save to measurement vector
        est.rbt.m(4,1) =...
            (est.linVelGnssAprxRbt(1,1)+...
            est.linVelImuAprxRbt(1,1))/2*est.linVelBias;
        est.rbt.m(5,1) =...
            (est.linVelGnssAprxRbt(2,1)+...
            est.linVelImuAprxRbt(2,1))/2*est.linVelBias;
        % Fourth, save to revised measurement noise covariance
        % matrix based on sliding adaptive covariance gain
        est.rbt.s(4:5,:) = [...
            0 0 0 vardx 0 0 0 0 0;...
            0 0 0 0 vardy 0 0 0 0];                     % linear velocities in revised measurement
noise covariance matrix based on sliding adaptive covariance gain

    % If this condition throws, there is an error
    else

        disp('Error in assigning mcaEst.useGnssLinVel and mcaEst.useImuLinVel variables');
        est.rbt.m(4:5,est.k) = [0;0];
        est.rbt.s(4:5,:) = [...
            0 0 0 1 0 0 0 0 0;...
            0 0 0 0 1 0 0 0 0];

    end

end

function est = fdGnss(est,sen)

    % Calculate forward-difference of GNSS linear position data
    % converted to the robot frame
    est.linVelGnssAprxRbt(:,1) =...
        (1/((1/sen.gnss.rr)*est.linPos.n))*...
        (est.linPosRbtArray(1:2,1)-...
        est.linPosRbtArray(1:2,end));               % forward-difference of linear position
measurements in robot frame

end

function est = tmImu(est,sen)

    % Calculate the trapezoidal method of the IMU linear
    % acceleration data in the robot frame
    est.linVelImuAprxRbt(:,1) =...
        est.rbt.x(4:5,1)+...
        (1/2)*(est.linAccRbtArray(:,2)+...
        est.linAccRbtArray(:,1))*...
        (1/sen.imu.rr);                             % trapezoidal method of linear acceleration
data in robot frame based on last estimate

end

function est = runRbtKal(est)
% This function assumes that if this function is run, you want the
% estimator to update.

    % State estimator setup
    state.x = est.rbt.x;        % last estimate
    state.P = est.rbt.P;        % last covariance matrix
    state.Q = est.rbt.Q;        % process noise covariance
    state.R = est.rbt.s;        % measurement noise covariance
    state.H = est.rbt.H;        % measurement matrix
    state.z = est.rbt.m;        % meausrements
    state.u = [0;0;0];          % control input vector (don't give kalman filter knowledge about
thruster inputs)

    % Discrete Kalman filter
    state.A = est.rbt.A;
    state.B = est.rbt.B;
    state = kalmanf(state);

    % Store state estimate
    est.rbt.x = state.x;
    est.rbt.L = state.K;
    est.rbt.P = state.P;

end
```

```
function est = estMapMeas(est)

    % Set sliding covariance gain matrix to identity
    est.map.s = eye(6,6);                                % map frame sliding covariance gain
matrix

    % Linear position measurements and gains
    est.map.m(1:2,1) = est.linPosMapArray(1:2,1);        % linear position in map frame
    est.map.s(1:2,:) =...
        est.map.R(1:2,:)*est.gnssSlidingGain;            % linear positions in sliding covariance
gain matrix

    % Angular position measurements and gains
    est.map.m(3,1) = est.rbt.x(3,1);                     % angular position in map frame
    est.map.s(3,:) =...
        est.map.R(3,:)*est.imuSlidingGain;               % angular position in sliding covariance
gain matrix

    % Robot frame linear velocity pseudo measurements and gains comes from
    % the linear velocity estimate output from the robot frame estimator
    est.map.m(4:5,1) = est.rbt.x(4:5,1);
    if (est.useGnssLinVel == 1 && est.useImuLinVel == 0)
        est.map.s(4:5,:) =...
            est.map.R(4:5,:)*est.gnssSlidingGain;
    elseif (est.useGnssLinVel == 0 && est.useImuLinVel == 1)
        est.map.s(4:5,:) =...
            est.map.R(4:5,:)*est.imuSlidingGain;
    elseif (est.useGnssLinVel == 1 && est.useImuLinVel == 1)
        est.map.s(4:5,:) =...
            est.map.R(4:5,:)*est.gnssImuSlidingGain;
    else
        est.map.s(4:5,:) =...
            est.map.R(4:5,:);
    end

    % Angular velocity measurements
    est.map.m(6,1) = est.rbt.x(6,1);                     % angular velocity in map frame
    est.map.s(6,:) =...
        est.map.R(6,:)*est.imuSlidingGain;               % angular velocity in sliding covariance
gain matrix

end

function est = runMapKal(est)
% This function assumes that if this function is run, you want the
% estimator to update.

    % State estimator setup
    state.x = est.map.x;        % last estimate
    state.P = est.map.P;        % last covariance matrix
    state.Q = est.map.Q;        % process noise covariance
    state.R = est.map.s;        % measurement noise covariance
    state.H = est.map.H;        % measurement matrix
    state.z = est.map.m;        % meausrements
    state.u = [0;0;0];          % control input vector (don't give kalman filter knowledge about
thruster inputs)

    % Discrete Kalman filter
    state.A = est.map.A(est.map.m);
    state.B = est.map.B;
    state = kalmanf(state);

    % Store state estimate
    est.map.x = state.x;
    est.map.L = state.K;
    est.map.P = state.P;

end
```

```
function s = kalmanf(s)

    % set defaults for absent fields:
    if ~isfield(s,'x'); s.x = nan*z; end
    if ~isfield(s,'P'); s.P = nan; end
    if ~isfield(s,'z'); error('Observation vector missing'); end
    if ~isfield(s,'u'); s.u = 0; end
    if ~isfield(s,'A'); s.A = eye(length(s.x)); end
    if ~isfield(s,'B'); s.B = 0; end
    if ~isfield(s,'Q'); s.Q = zeros(length(s.x)); end
    if ~isfield(s,'R'); error('Observation covariance missing'); end
    if ~isfield(s,'H'); s.H = eye(length(s.x)); end

    % Prediction for state vector and covariance:
    s.x = s.A*s.x+s.B*s.u;
    s.P = s.A*s.P*s.A'+s.Q;

    % Compute Kalman gain factor:
    s.K = s.P*s.H'*inv(s.H*s.P*s.H'+s.R);

    % Correction based on observation:
    s.x = s.x + s.K*(s.z-s.H*s.x);
    s.P = s.P - s.K*s.H*s.P;

end

function write2pub(pub,msg,est,sen)

    % Update GNSS sensor publishers if GNSS sensor updated
    if sen.gnss.update == 1

        % Update message data
        msg.gnss.map.x.Data = sen.gnss.x;
        msg.gnss.map.y.Data = sen.gnss.y;
        msg.gnss.map.tz.Data = sen.gnss.tzUnwrap;
        msg.gnss.map.pose.X = sen.gnss.x;
        msg.gnss.map.pose.Y = sen.gnss.y;
        msg.gnss.map.pose.Z = 0;
        msg.gnss.map.tpp.Data = sen.gnss.tpp;
        msg.gnss.rbt.x.Data = sen.gnss.xr;
        msg.gnss.rbt.y.Data = sen.gnss.yr;

        % Send/publish to topic
        send(pub.gnss.map.x,msg.gnss.map.x);
        send(pub.gnss.map.y,msg.gnss.map.y);
        send(pub.gnss.map.tz,msg.gnss.map.tz);
        send(pub.gnss.map.pose,msg.gnss.map.pose);
        send(pub.gnss.map.tpp,msg.gnss.map.tpp);
        send(pub.gnss.rbt.x,msg.gnss.rbt.x);
        send(pub.gnss.rbt.y,msg.gnss.rbt.y);

    end

    % Update IMU sensor publishers if IMU sensor updated
    if sen.imu.update == 1

        % Update message data
        msg.imu.map.tz.Data = sen.imu.tzUnwrap;
        msg.imu.map.dtz.Data = sen.imu.dtz;
        msg.imu.rbt.ddx.Data = sen.imu.ddx;
        msg.imu.rbt.ddy.Data = sen.imu.ddy;

        % Send/publish to topic
        send(pub.imu.map.tz,msg.imu.map.tz);
        send(pub.imu.map.dtz,msg.imu.map.dtz);
        send(pub.imu.rbt.ddx,msg.imu.rbt.ddx);
        send(pub.imu.rbt.ddy,msg.imu.rbt.ddy);

    end
    % Update estimator publishers
    msg.est.map.x.Data = est.map.x(1,1);
    msg.est.map.y.Data = est.map.x(2,1);
    msg.est.map.tz.Data = est.map.x(3,1);
    msg.est.map.pose.X = est.map.x(1,1);
    msg.est.map.pose.Y = est.map.x(2,1);
    msg.est.map.pose.Z = 0;
    msg.est.map.dtz.Data = est.map.x(6,1);
    msg.est.rbt.x.Data = est.rbt.x(1,1);
    msg.est.rbt.y.Data = est.rbt.x(2,1);
    msg.est.rbt.dx.Data = est.rbt.x(4,1);
    msg.est.rbt.dy.Data = est.rbt.x(5,1);
    msg.est.rbt.ddx.Data = est.rbt.x(7,1);
    msg.est.rbt.ddy.Data = est.rbt.x(8,1);
    msg.est.linVelAbs.Data = sqrt(est.rbt.x(4,1)^2+est.rbt.x(5,1)^2);
    msg.est.linVelSurge.Data = abs(est.rbt.x(4,1));
    msg.est.linVelSway.Data = abs(est.rbt.x(5,1));

    % Send/publish to topic
    send(pub.est.map.x,msg.est.map.x);
    send(pub.est.map.y,msg.est.map.y);
    send(pub.est.map.tz,msg.est.map.tz);
    send(pub.est.map.pose,msg.est.map.pose);
    send(pub.est.map.dtz,msg.est.map.dtz);
    send(pub.est.rbt.x,msg.est.rbt.x);
    send(pub.est.rbt.y,msg.est.rbt.y);
    send(pub.est.rbt.dx,msg.est.rbt.dx);
    send(pub.est.rbt.dy,msg.est.rbt.dy);
    send(pub.est.rbt.ddx,msg.est.rbt.ddx);
    send(pub.est.rbt.ddy,msg.est.rbt.ddy);
    send(pub.est.linVelAbs,msg.est.linVelAbs);
    send(pub.est.linVelSurge,msg.est.linVelSurge);
    send(pub.est.linVelSway,msg.est.linVelSway);

end

function gnssCallback(~,message)

    % Instantiate global placeholder variables for callback
    global gnssSec gnssNsec
    global gnssLat gnssLon gnssAlt
    global gnssTic

    % GNSS tic-toc
    gnssTic = tic;

    % Save message data to global variables
    gnssSec = message.Header.Stamp.Sec;
    gnssNsec = message.Header.Stamp.Nsec;
    gnssLat = message.Latitude;
    gnssLon = message.Longitude;
    gnssAlt = message.Altitude;

end

function imuDataCallback(~,message)

    % IMPORTANT: For information about frame correction, read the section
    % "Coordinate Systems" here:
    % https://github.com/riplaboratory/Kanaloa/blob/master/Documentation/Sensors/RSX-
UM7_intertialMeasurementUnit/README.md

    % Instantiate global placeholder variables for callback
    global imuSec imuNsec
    global imutx imuty imutz
    global imudtx imudty imudtz
    global imuddx imuddy imuddz
    global imuTic

    % IMU tic-toc
    imuTic = tic;

    % Convert quaternion to euler angles; this will output x-left,
    % y-forward, z-down
    eul = quat2eul([...
        message.Orientation.X,...
        message.Orientation.Y,...
        message.Orientation.Z,...
        message.Orientation.W],...
        'ZYX');

    % Correct quaternion from (x-left,y-forward,z-down) to (x-forward,y-left,z-up)
    zDn = eul(1,1);
    yFw = eul(1,2);
    xLf = eul(1,3);
    tx = yFw;
    ty = xLf;
    tz = -zDn+3*pi/2;

    % Manual mag vector correction
    tz = tz - 0.4;

    % Save message data to global variables
    imuSec = message.Header.Stamp.Sec;
    imuNsec = message.Header.Stamp.Nsec;
    imutx = tx;
    imuty = ty;
    imutz = tz;
    imudtx = message.AngularVelocity.X;
    imudty = message.AngularVelocity.Y;
    imudtz = message.AngularVelocity.Z;
    imuddx = message.LinearAcceleration.X;
    imuddy = message.LinearAcceleration.Y;
    imuddz = message.LinearAcceleration.Z;

end
```

8.3. Post-processing Matlab code for rectangle ground truth .csv data

```
function processSquareBags()

    % Import data runs, set argument to 0 to import from .mat (fast), set
    % to 1 to import from actual .csv (slow)
    square = importSquareData(0);

    % Manual data repairs
    square = manualDataRepairs(square);

    % Zero map frame data
    square = zeroMapData(square);

    % Create ground truth rectangle
    square = createGroundTruthRectangle(square);

    % Calculate errors
    square = calculateAbsoluteError(square);
    square = calculateInterpolationError(square);

    % Plot stuff
    plotAllData(square);

    % Output to workspace (optional)
    assignin('base','square',square);

end
```

```
function square = importSquareData(csvImport)

    % Import from actual .csv files
    if (csvImport == 1)

        % CCW Fast 1
        square.ccwFast1.gnssMapPose = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-est-
gnss-map-pose.csv';
        square.ccwFast1.gnssRbtPose = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-est-
gnss-rbt-pose.csv';
        square.ccwFast1.imuPose = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-est-imu-
rbt-pose.csv';
        square.ccwFast1.imuTwist = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-est-imu-
rbt-twist.csv';
        square.ccwFast1.imuAcc = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-est-imu-
rbt-acc.csv';
        square.ccwFast1.estMapPose = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-est-
map-pose.csv';
        square.ccwFast1.estRbtPose = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-est-
rbt-pose.csv';
        square.ccwFast1.estRbtTwist = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-est-
rbt-twist.csv';
        square.ccwFast1.estRbtAcc = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-est-rbt-
acc.csv';
        square.ccwFast1.estRbtLinVelAbs = 'zone20SquareCcwFast1/zone20SquareCcwFast1_estimated-
est-rbt-linVelAbs.csv';
        square.ccwFast1 = importEstimatorBag(square.ccwFast1);

        % CCW Fast 2
        square.ccwFast2.gnssMapPose = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-est-
gnss-map-pose.csv';
        square.ccwFast2.gnssRbtPose = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-est-
gnss-rbt-pose.csv';
        square.ccwFast2.imuPose = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-est-imu-
rbt-pose.csv';
        square.ccwFast2.imuTwist = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-est-imu-
rbt-twist.csv';
        square.ccwFast2.imuAcc = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-est-imu-
rbt-acc.csv';
        square.ccwFast2.estMapPose = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-est-
map-pose.csv';
        square.ccwFast2.estRbtPose = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-est-
rbt-pose.csv';
        square.ccwFast2.estRbtTwist = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-est-
rbt-twist.csv';
        square.ccwFast2.estRbtAcc = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-est-rbt-
acc.csv';
        square.ccwFast2.estRbtLinVelAbs = 'zone20SquareCcwFast2/zone20SquareCcwFast2_estimated-
est-rbt-linVelAbs.csv';
        square.ccwFast2 = importEstimatorBag(square.ccwFast2);

        % CCW Fast 3
        square.ccwFast3.gnssMapPose = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-est-
gnss-map-pose.csv';
        square.ccwFast3.gnssRbtPose = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-est-
gnss-rbt-pose.csv';
        square.ccwFast3.imuPose = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-est-imu-
rbt-pose.csv';
        square.ccwFast3.imuTwist = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-est-imu-
rbt-twist.csv';
        square.ccwFast3.imuAcc = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-est-imu-
rbt-acc.csv';
        square.ccwFast3.estMapPose = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-est-
map-pose.csv';
        square.ccwFast3.estRbtPose = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-est-
rbt-pose.csv';
        square.ccwFast3.estRbtTwist = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-est-
rbt-twist.csv';
        square.ccwFast3.estRbtAcc = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-est-rbt-
acc.csv';
```

```
        square.ccwFast3.estRbtLinVelAbs = 'zone20SquareCcwFast3/zone20SquareCcwFast3_estimated-
est-rbt-linVelAbs.csv';
        square.ccwFast3 = importEstimatorBag(square.ccwFast3);

        % CCW Fast 4
        square.ccwFast4.gnssMapPose = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-est-
gnss-map-pose.csv';
        square.ccwFast4.gnssRbtPose = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-est-
gnss-rbt-pose.csv';
        square.ccwFast4.imuPose = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-est-imu-
rbt-pose.csv';
        square.ccwFast4.imuTwist = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-est-imu-
rbt-twist.csv';
        square.ccwFast4.imuAcc = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-est-imu-
rbt-acc.csv';
        square.ccwFast4.estMapPose = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-est-
map-pose.csv';
        square.ccwFast4.estRbtPose = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-est-
rbt-pose.csv';
        square.ccwFast4.estRbtTwist = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-est-
rbt-twist.csv';
        square.ccwFast4.estRbtAcc = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-est-rbt-
acc.csv';
        square.ccwFast4.estRbtLinVelAbs = 'zone20SquareCcwFast4/zone20SquareCcwFast4_estimated-
est-rbt-linVelAbs.csv';
        square.ccwFast4 = importEstimatorBag(square.ccwFast4);

        % CCW Slow 1
        square.ccwSlow1.gnssMapPose = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-est-
gnss-map-pose.csv';
        square.ccwSlow1.gnssRbtPose = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-est-
gnss-rbt-pose.csv';
        square.ccwSlow1.imuPose = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-est-imu-
rbt-pose.csv';
        square.ccwSlow1.imuTwist = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-est-imu-
rbt-twist.csv';
        square.ccwSlow1.imuAcc = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-est-imu-
rbt-acc.csv';
        square.ccwSlow1.estMapPose = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-est-
map-pose.csv';
        square.ccwSlow1.estRbtPose = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-est-
rbt-pose.csv';
        square.ccwSlow1.estRbtTwist = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-est-
rbt-twist.csv';
        square.ccwSlow1.estRbtAcc = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-est-rbt-
acc.csv';
        square.ccwSlow1.estRbtLinVelAbs = 'zone20SquareCcwSlow1/zone20SquareCcwSlow1_estimated-
est-rbt-linVelAbs.csv';
        square.ccwSlow1 = importEstimatorBag(square.ccwSlow1);

        % CCW Slow 2
        square.ccwSlow2.gnssMapPose = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-est-
gnss-map-pose.csv';
        square.ccwSlow2.gnssRbtPose = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-est-
gnss-rbt-pose.csv';
        square.ccwSlow2.imuPose = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-est-imu-
rbt-pose.csv';
        square.ccwSlow2.imuTwist = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-est-imu-
rbt-twist.csv';
        square.ccwSlow2.imuAcc = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-est-imu-
rbt-acc.csv';
        square.ccwSlow2.estMapPose = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-est-
map-pose.csv';
        square.ccwSlow2.estRbtPose = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-est-
rbt-pose.csv';
        square.ccwSlow2.estRbtTwist = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-est-
rbt-twist.csv';
        square.ccwSlow2.estRbtAcc = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-est-rbt-
acc.csv';
```

```
        square.ccwSlow2.estRbtLinVelAbs = 'zone20SquareCcwSlow2/zone20SquareCcwSlow2_estimated-
est-rbt-linVelAbs.csv';
        square.ccwSlow2 = importEstimatorBag(square.ccwSlow2);

        % CCW Slow 3
        square.ccwSlow3.gnssMapPose = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-est-
gnss-map-pose.csv';
        square.ccwSlow3.gnssRbtPose = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-est-
gnss-rbt-pose.csv';
        square.ccwSlow3.imuPose = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-est-imu-
rbt-pose.csv';
        square.ccwSlow3.imuTwist = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-est-imu-
rbt-twist.csv';
        square.ccwSlow3.imuAcc = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-est-imu-
rbt-acc.csv';
        square.ccwSlow3.estMapPose = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-est-
map-pose.csv';
        square.ccwSlow3.estRbtPose = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-est-
rbt-pose.csv';
        square.ccwSlow3.estRbtTwist = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-est-
rbt-twist.csv';
        square.ccwSlow3.estRbtAcc = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-est-rbt-
acc.csv';
        square.ccwSlow3.estRbtLinVelAbs = 'zone20SquareCcwSlow3/zone20SquareCcwSlow3_estimated-
est-rbt-linVelAbs.csv';
        square.ccwSlow3 = importEstimatorBag(square.ccwSlow3);

        % CCW Slow 4
        square.ccwSlow4.gnssMapPose = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-est-
gnss-map-pose.csv';
        square.ccwSlow4.gnssRbtPose = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-est-
gnss-rbt-pose.csv';
        square.ccwSlow4.imuPose = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-est-imu-
rbt-pose.csv';
        square.ccwSlow4.imuTwist = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-est-imu-
rbt-twist.csv';
        square.ccwSlow4.imuAcc = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-est-imu-
rbt-acc.csv';
        square.ccwSlow4.estMapPose = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-est-
map-pose.csv';
        square.ccwSlow4.estRbtPose = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-est-
rbt-pose.csv';
        square.ccwSlow4.estRbtTwist = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-est-
rbt-twist.csv';
        square.ccwSlow4.estRbtAcc = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-est-rbt-
acc.csv';
        square.ccwSlow4.estRbtLinVelAbs = 'zone20SquareCcwSlow4/zone20SquareCcwSlow4_estimated-
est-rbt-linVelAbs.csv';
        square.ccwSlow4 = importEstimatorBag(square.ccwSlow4);

        % CCW Slow 5
        square.ccwSlow5.gnssMapPose = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-est-
gnss-map-pose.csv';
        square.ccwSlow5.gnssRbtPose = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-est-
gnss-rbt-pose.csv';
        square.ccwSlow5.imuPose = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-est-imu-
rbt-pose.csv';
        square.ccwSlow5.imuTwist = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-est-imu-
rbt-twist.csv';
        square.ccwSlow5.imuAcc = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-est-imu-
rbt-acc.csv';
        square.ccwSlow5.estMapPose = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-est-
map-pose.csv';
        square.ccwSlow5.estRbtPose = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-est-
rbt-pose.csv';
        square.ccwSlow5.estRbtTwist = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-est-
rbt-twist.csv';
        square.ccwSlow5.estRbtAcc = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-est-rbt-
acc.csv';
```

```
        square.ccwSlow5.estRbtLinVelAbs = 'zone20SquareCcwSlow5/zone20SquareCcwSlow5_estimated-
est-rbt-linVelAbs.csv';
        square.ccwSlow5 = importEstimatorBag(square.ccwSlow5);

        % CCW Slow 6
        square.ccwSlow6.gnssMapPose = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-est-
gnss-map-pose.csv';
        square.ccwSlow6.gnssRbtPose = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-est-
gnss-rbt-pose.csv';
        square.ccwSlow6.imuPose = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-est-imu-
rbt-pose.csv';
        square.ccwSlow6.imuTwist = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-est-imu-
rbt-twist.csv';
        square.ccwSlow6.imuAcc = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-est-imu-
rbt-acc.csv';
        square.ccwSlow6.estMapPose = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-est-
map-pose.csv';
        square.ccwSlow6.estRbtPose = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-est-
rbt-pose.csv';
        square.ccwSlow6.estRbtTwist = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-est-
rbt-twist.csv';
        square.ccwSlow6.estRbtAcc = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-est-rbt-
acc.csv';
        square.ccwSlow6.estRbtLinVelAbs = 'zone20SquareCcwSlow6/zone20SquareCcwSlow6_estimated-
est-rbt-linVelAbs.csv';
        square.ccwSlow6 = importEstimatorBag(square.ccwSlow6);

        % CW Fast 1
        square.cwFast1.gnssMapPose = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-gnss-
map-pose.csv';
        square.cwFast1.gnssRbtPose = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-gnss-
rbt-pose.csv';
        square.cwFast1.imuPose = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-imu-rbt-
pose.csv';
        square.cwFast1.imuTwist = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-imu-rbt-
twist.csv';
        square.cwFast1.imuAcc = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-imu-rbt-
acc.csv';
        square.cwFast1.estMapPose = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-map-
pose.csv';
        square.cwFast1.estRbtPose = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-rbt-
pose.csv';
        square.cwFast1.estRbtTwist = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-rbt-
twist.csv';
        square.cwFast1.estRbtAcc = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-rbt-
acc.csv';
        square.cwFast1.estRbtLinVelAbs = 'zone20SquareCwFast1/zone20SquareCwFast1_estimated-est-
rbt-linVelAbs.csv';
        square.cwFast1 = importEstimatorBag(square.cwFast1);

        % CW Fast 2
        square.cwFast2.gnssMapPose = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-gnss-
map-pose.csv';
        square.cwFast2.gnssRbtPose = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-gnss-
rbt-pose.csv';
        square.cwFast2.imuPose = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-imu-rbt-
pose.csv';
        square.cwFast2.imuTwist = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-imu-rbt-
twist.csv';
        square.cwFast2.imuAcc = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-imu-rbt-
acc.csv';
        square.cwFast2.estMapPose = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-map-
pose.csv';
        square.cwFast2.estRbtPose = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-rbt-
pose.csv';
        square.cwFast2.estRbtTwist = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-rbt-
twist.csv';
        square.cwFast2.estRbtAcc = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-rbt-
acc.csv';
```

```
        square.cwFast2.estRbtLinVelAbs = 'zone20SquareCwFast2/zone20SquareCwFast2_estimated-est-
rbt-linVelAbs.csv';
        square.cwFast2 = importEstimatorBag(square.cwFast2);

        % CW Fast 3
        square.cwFast3.gnssMapPose = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-gnss-
map-pose.csv';
        square.cwFast3.gnssRbtPose = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-gnss-
rbt-pose.csv';
        square.cwFast3.imuPose = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-imu-rbt-
pose.csv';
        square.cwFast3.imuTwist = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-imu-rbt-
twist.csv';
        square.cwFast3.imuAcc = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-imu-rbt-
acc.csv';
        square.cwFast3.estMapPose = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-map-
pose.csv';
        square.cwFast3.estRbtPose = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-rbt-
pose.csv';
        square.cwFast3.estRbtTwist = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-rbt-
twist.csv';
        square.cwFast3.estRbtAcc = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-rbt-
acc.csv';
        square.cwFast3.estRbtLinVelAbs = 'zone20SquareCwFast3/zone20SquareCwFast3_estimated-est-
rbt-linVelAbs.csv';
        square.cwFast3 = importEstimatorBag(square.cwFast3);

        % CW Fast 4
        square.cwFast4.gnssMapPose = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-gnss-
map-pose.csv';
        square.cwFast4.gnssRbtPose = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-gnss-
rbt-pose.csv';
        square.cwFast4.imuPose = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-imu-rbt-
pose.csv';
        square.cwFast4.imuTwist = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-imu-rbt-
twist.csv';
        square.cwFast4.imuAcc = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-imu-rbt-
acc.csv';
        square.cwFast4.estMapPose = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-map-
pose.csv';
        square.cwFast4.estRbtPose = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-rbt-
pose.csv';
        square.cwFast4.estRbtTwist = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-rbt-
twist.csv';
        square.cwFast4.estRbtAcc = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-rbt-
acc.csv';
        square.cwFast4.estRbtLinVelAbs = 'zone20SquareCwFast4/zone20SquareCwFast4_estimated-est-
rbt-linVelAbs.csv';
        square.cwFast4 = importEstimatorBag(square.cwFast4);

        % CW Slow 1
        square.cwSlow1.gnssMapPose = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-gnss-
map-pose.csv';
        square.cwSlow1.gnssRbtPose = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-gnss-
rbt-pose.csv';
        square.cwSlow1.imuPose = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-imu-rbt-
pose.csv';
        square.cwSlow1.imuTwist = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-imu-rbt-
twist.csv';
        square.cwSlow1.imuAcc = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-imu-rbt-
acc.csv';
        square.cwSlow1.estMapPose = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-map-
pose.csv';
        square.cwSlow1.estRbtPose = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-rbt-
pose.csv';
        square.cwSlow1.estRbtTwist = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-rbt-
twist.csv';
        square.cwSlow1.estRbtAcc = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-rbt-
acc.csv';
```

```
        square.cwSlow1.estRbtLinVelAbs = 'zone20SquareCwSlow1/zone20SquareCwSlow1_estimated-est-
rbt-linVelAbs.csv';
        square.cwSlow1 = importEstimatorBag(square.cwSlow1);

        % CW Slow 2
        square.cwSlow2.gnssMapPose = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-gnss-
map-pose.csv';
        square.cwSlow2.gnssRbtPose = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-gnss-
rbt-pose.csv';
        square.cwSlow2.imuPose = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-imu-rbt-
pose.csv';
        square.cwSlow2.imuTwist = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-imu-rbt-
twist.csv';
        square.cwSlow2.imuAcc = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-imu-rbt-
acc.csv';
        square.cwSlow2.estMapPose = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-map-
pose.csv';
        square.cwSlow2.estRbtPose = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-rbt-
pose.csv';
        square.cwSlow2.estRbtTwist = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-rbt-
twist.csv';
        square.cwSlow2.estRbtAcc = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-rbt-
acc.csv';
        square.cwSlow2.estRbtLinVelAbs = 'zone20SquareCwSlow2/zone20SquareCwSlow2_estimated-est-
rbt-linVelAbs.csv';
        square.cwSlow2 = importEstimatorBag(square.cwSlow2);

        % CW Slow 3
        square.cwSlow3.gnssMapPose = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-gnss-
map-pose.csv';
        square.cwSlow3.gnssRbtPose = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-gnss-
rbt-pose.csv';
        square.cwSlow3.imuPose = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-imu-rbt-
pose.csv';
        square.cwSlow3.imuTwist = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-imu-rbt-
twist.csv';
        square.cwSlow3.imuAcc = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-imu-rbt-
acc.csv';
        square.cwSlow3.estMapPose = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-map-
pose.csv';
        square.cwSlow3.estRbtPose = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-rbt-
pose.csv';
        square.cwSlow3.estRbtTwist = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-rbt-
twist.csv';
        square.cwSlow3.estRbtAcc = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-rbt-
acc.csv';
        square.cwSlow3.estRbtLinVelAbs = 'zone20SquareCwSlow3/zone20SquareCwSlow3_estimated-est-
rbt-linVelAbs.csv';
        square.cwSlow3 = importEstimatorBag(square.cwSlow3);

        % CW Slow 4
        square.cwSlow4.gnssMapPose = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-gnss-
map-pose.csv';
        square.cwSlow4.gnssRbtPose = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-gnss-
rbt-pose.csv';
        square.cwSlow4.imuPose = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-imu-rbt-
pose.csv';
        square.cwSlow4.imuTwist = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-imu-rbt-
twist.csv';
        square.cwSlow4.imuAcc = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-imu-rbt-
acc.csv';
        square.cwSlow4.estMapPose = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-map-
pose.csv';
        square.cwSlow4.estRbtPose = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-rbt-
pose.csv';
        square.cwSlow4.estRbtTwist = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-rbt-
twist.csv';
        square.cwSlow4.estRbtAcc = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-rbt-
acc.csv';
```

```
        square.cwSlow4.estRbtLinVelAbs = 'zone20SquareCwSlow4/zone20SquareCwSlow4_estimated-est-
rbt-linVelAbs.csv';
        square.cwSlow4 = importEstimatorBag(square.cwSlow4);

        % Save to file
        save('square.mat','square');

    % Import from .mat file
    else

        load('square.mat');

    end

end

function trial = importEstimatorBag(trial)

    % Sensor refresh rate for approximate time vectors
    gnssRR = 1;     % GNSS refresh rate in [Hz]
    imuRR = 20;     % IMU refresh rate in [Hz]
    estRR = 40;     % estimator refresh rate in [Hz]

    % GNSS map pose
    string = readmatrix(trial.gnssMapPose,...
        'Delimiter',',','OutputType','string');     % read in GNSS map pose as a string (for time
data)
    num = readmatrix(trial.gnssMapPose,...
        'Delimiter',',');                           % read in GNSS map pose as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/gnssRR:(row-2)*(1/gnssRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.gnssMapPoseDatetime = dt;
    trial.gnssMapPoseData = num(:,2:4);

    % GNSS robot pose
    string = readmatrix(trial.gnssRbtPose,...
        'Delimiter',',','OutputType','string');     % read in GNSS robot pose as a string (for
time data)
    num = readmatrix(trial.gnssRbtPose,...
        'Delimiter',',');                           % read in GNSS robot pose as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/gnssRR:(row-2)*(1/gnssRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.gnssRbtPoseDatetime = dt;
    trial.gnssRbtPoseData = num(:,2:4);

    % IMU pose
    string = readmatrix(trial.imuPose,...
        'Delimiter',',','OutputType','string');     % read in IMU pose as a string (for time
data)
    num = readmatrix(trial.imuPose,...
        'Delimiter',',');                           % read in IMU pose as doubles (for actual
data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/imuRR:(row-2)*(1/imuRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.imuPoseDatetime = dt;
    trial.imuPoseData = num(:,2:4);

    % IMU twist
    string = readmatrix(trial.imuTwist,...
        'Delimiter',',','OutputType','string');     % read in IMU twist as a string (for time
data)
    num = readmatrix(trial.imuTwist,...
        'Delimiter',',');                           % read in IMU twist as doubles (for actual
data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/imuRR:(row-2)*(1/imuRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.imuTwistDatetime = dt;
    trial.imuTwistData = num(:,2:4);

    % IMU acceleration
    string = readmatrix(trial.imuAcc,...
        'Delimiter',',','OutputType','string');     % read in IMU acceleration as a string (for
time data)
    num = readmatrix(trial.imuAcc,...
        'Delimiter',',');                           % read in IMU acceleration as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/imuRR:(row-2)*(1/imuRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.imuAccDatetime = dt;
    trial.imuAccData = num(:,2:4);

    % Estimator map pose
    string = readmatrix(trial.estMapPose,...
        'Delimiter',',','OutputType','string');     % read in estimator map pose as a string (for
time data)
    num = readmatrix(trial.estMapPose,...
        'Delimiter',',');                            % read in estimator map pose as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estMapPoseDatetime = dt;
    trial.estMapPoseData = num(:,2:4);

    % Estimator robot pose
    string = readmatrix(trial.estRbtPose,...
        'Delimiter',',','OutputType','string');     % read in estimator robot pose as a string
(for time data)
    num = readmatrix(trial.estRbtPose,...
        'Delimiter',',');                            % read in estimator robot pose as doubles
(for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estRbtPoseDatetime = dt;
    trial.estRbtPoseData = num(:,2:4);

    % Estimator robot twist
    string = readmatrix(trial.estRbtTwist,...
        'Delimiter',',','OutputType','string');     % read in estimator robot twist as a string
(for time data)
    num = readmatrix(trial.estRbtTwist,...
        'Delimiter',',');                           % read in estimator robot twist as doubles
(for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estRbtTwistDatetime = dt;
    trial.estRbtTwistData = num(:,2:4);

    % Estimator robot acceleration
    string = readmatrix(trial.estRbtAcc,...
        'Delimiter',',','OutputType','string');     % read in estimator robot acceleration as a
string (for time data)
    num = readmatrix(trial.estRbtAcc,...
        'Delimiter',',');                           % read in estimator robot acceleration as
doubles (for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estRbtAccDatetime = dt;
    trial.estRbtAccData = num(:,2:4);

    % Estimator robot absolute linear velocity vector
    string = readmatrix(trial.estRbtLinVelAbs,...
        'Delimiter',',','OutputType','string');     % read in estimator robot absolute linear
velocity vector as a string (for time data)
    num = readmatrix(trial.estRbtLinVelAbs,...
        'Delimiter',',');                           % read in estimator robot absolute linear
velocity vector as doubles (for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.linVelAbsDatetime = dt;
    trial.linVelAbsData = num(:,2:4);

end

function square = manualDataRepairs(square)

    % CCW Fast 1
    gnss0 = 1;
    imu0 = 20;
    est0 = 130;
    square.ccwFast1.gnssMapPoseDatetime(1:gnss0,:) = [];
    square.ccwFast1.gnssMapPoseData(1:gnss0,:) = [];
    square.ccwFast1.gnssRbtPoseDatetime(1:gnss0,:) = [];
    square.ccwFast1.gnssRbtPoseData(1:gnss0,:) = [];
    square.ccwFast1.imuPoseDatetime(1:imu0,:) = [];
    square.ccwFast1.imuPoseData(1:imu0,:) = [];
    square.ccwFast1.imuTwistDatetime(1:imu0,:) = [];
    square.ccwFast1.imuTwistData(1:imu0,:) = [];
    square.ccwFast1.imuAccDatetime(1:imu0,:) = [];
    square.ccwFast1.imuAccData(1:imu0,:) = [];
    square.ccwFast1.estMapPoseDatetime(1:est0,:) = [];
    square.ccwFast1.estMapPoseData(1:est0,:) = [];
    square.ccwFast1.estRbtPoseDatetime(1:est0,:) = [];
    square.ccwFast1.estRbtPoseData(1:est0,:) = [];
    square.ccwFast1.estRbtTwistDatetime(1:est0,:) = [];
    square.ccwFast1.estRbtTwistData(1:est0,:) = [];
    square.ccwFast1.estRbtAccDatetime(1:est0,:) = [];
    square.ccwFast1.estRbtAccData(1:est0,:) = [];
    square.ccwFast1.linVelAbsDatetime(1:est0,:) = [];
    square.ccwFast1.linVelAbsData(1:est0,:) = [];
    % remove a bunch of estimator entries at the end
    endRem = 120;
    square.ccwFast1.estMapPoseDatetime(end-endRem:end,:) = [];
    square.ccwFast1.estMapPoseData(end-endRem:end,:) = [];
    square.ccwFast1.estRbtPoseDatetime(end-endRem:end,:) = [];
    square.ccwFast1.estRbtPoseData(end-endRem:end,:) = [];
    square.ccwFast1.estRbtTwistDatetime(end-endRem:end,:) = [];
    square.ccwFast1.estRbtTwistData(end-endRem:end,:) = [];
    square.ccwFast1.estRbtAccDatetime(end-endRem:end,:) = [];
    square.ccwFast1.estRbtAccData(end-endRem:end,:) = [];
    square.ccwFast1.linVelAbsDatetime(end-endRem:end,:) = [];
    square.ccwFast1.linVelAbsData(end-endRem:end,:) = [];

    % CCW Fast 2
    gnss0 = 1;
    imu0 = 25;
    est0 = 90;
    square.ccwFast2.gnssMapPoseDatetime(1:gnss0,:) = [];
    square.ccwFast2.gnssMapPoseData(1:gnss0,:) = [];
    square.ccwFast2.gnssRbtPoseDatetime(1:gnss0,:) = [];
    square.ccwFast2.gnssRbtPoseData(1:gnss0,:) = [];
    square.ccwFast2.imuPoseDatetime(1:imu0,:) = [];
    square.ccwFast2.imuPoseData(1:imu0,:) = [];
    square.ccwFast2.imuTwistDatetime(1:imu0,:) = [];
    square.ccwFast2.imuTwistData(1:imu0,:) = [];
    square.ccwFast2.imuAccDatetime(1:imu0,:) = [];
    square.ccwFast2.imuAccData(1:imu0,:) = [];
    square.ccwFast2.estMapPoseDatetime(1:est0,:) = [];
    square.ccwFast2.estMapPoseData(1:est0,:) = [];
    square.ccwFast2.estRbtPoseDatetime(1:est0,:) = [];
    square.ccwFast2.estRbtPoseData(1:est0,:) = [];
    square.ccwFast2.estRbtTwistDatetime(1:est0,:) = [];
    square.ccwFast2.estRbtTwistData(1:est0,:) = [];
    square.ccwFast2.estRbtAccDatetime(1:est0,:) = [];
    square.ccwFast2.estRbtAccData(1:est0,:) = [];
    square.ccwFast2.linVelAbsDatetime(1:est0,:) = [];
    square.ccwFast2.linVelAbsData(1:est0,:) = [];
    % remove a bunch of estimator entries at the end
    endRem = 100;
    square.ccwFast2.estMapPoseDatetime(end-endRem:end,:) = [];
    square.ccwFast2.estMapPoseData(end-endRem:end,:) = [];
    square.ccwFast2.estRbtPoseDatetime(end-endRem:end,:) = [];
    square.ccwFast2.estRbtPoseData(end-endRem:end,:) = [];
    square.ccwFast2.estRbtTwistDatetime(end-endRem:end,:) = [];
    square.ccwFast2.estRbtTwistData(end-endRem:end,:) = [];
    square.ccwFast2.estRbtAccDatetime(end-endRem:end,:) = [];
    square.ccwFast2.estRbtAccData(end-endRem:end,:) = [];
    square.ccwFast2.linVelAbsDatetime(end-endRem:end,:) = [];
    square.ccwFast2.linVelAbsData(end-endRem:end,:) = [];

    % CCW Fast 3
    gnss0 = 0;
    imu0 = 22;
    est0 = 80;
    square.ccwFast3.gnssMapPoseDatetime(1:gnss0,:) = [];
    square.ccwFast3.gnssMapPoseData(1:gnss0,:) = [];
    square.ccwFast3.gnssRbtPoseDatetime(1:gnss0,:) = [];
    square.ccwFast3.gnssRbtPoseData(1:gnss0,:) = [];
    square.ccwFast3.imuPoseDatetime(1:imu0,:) = [];
    square.ccwFast3.imuPoseData(1:imu0,:) = [];
    square.ccwFast3.imuTwistDatetime(1:imu0,:) = [];
    square.ccwFast3.imuTwistData(1:imu0,:) = [];
    square.ccwFast3.imuAccDatetime(1:imu0,:) = [];
    square.ccwFast3.imuAccData(1:imu0,:) = [];
    square.ccwFast3.estMapPoseDatetime(1:est0,:) = [];
    square.ccwFast3.estMapPoseData(1:est0,:) = [];
    square.ccwFast3.estRbtPoseDatetime(1:est0,:) = [];
    square.ccwFast3.estRbtPoseData(1:est0,:) = [];
    square.ccwFast3.estRbtTwistDatetime(1:est0,:) = [];
    square.ccwFast3.estRbtTwistData(1:est0,:) = [];
    square.ccwFast3.estRbtAccDatetime(1:est0,:) = [];
    square.ccwFast3.estRbtAccData(1:est0,:) = [];
    square.ccwFast3.linVelAbsDatetime(1:est0,:) = [];
    square.ccwFast3.linVelAbsData(1:est0,:) = [];

    % CCW Fast 4
    gnss0 = 0;
    imu0 = 22;
    est0 = 45;
    square.ccwFast4.gnssMapPoseDatetime(1:gnss0,:) = [];
    square.ccwFast4.gnssMapPoseData(1:gnss0,:) = [];
    square.ccwFast4.gnssRbtPoseDatetime(1:gnss0,:) = [];
    square.ccwFast4.gnssRbtPoseData(1:gnss0,:) = [];
    square.ccwFast4.imuPoseDatetime(1:imu0,:) = [];
    square.ccwFast4.imuPoseData(1:imu0,:) = [];
    square.ccwFast4.imuTwistDatetime(1:imu0,:) = [];
    square.ccwFast4.imuTwistData(1:imu0,:) = [];
    square.ccwFast4.imuAccDatetime(1:imu0,:) = [];
    square.ccwFast4.imuAccData(1:imu0,:) = [];
    square.ccwFast4.estMapPoseDatetime(1:est0,:) = [];
    square.ccwFast4.estMapPoseData(1:est0,:) = [];
    square.ccwFast4.estRbtPoseDatetime(1:est0,:) = [];
    square.ccwFast4.estRbtPoseData(1:est0,:) = [];
    square.ccwFast4.estRbtTwistDatetime(1:est0,:) = [];
    square.ccwFast4.estRbtTwistData(1:est0,:) = [];
    square.ccwFast4.estRbtAccDatetime(1:est0,:) = [];
    square.ccwFast4.estRbtAccData(1:est0,:) = [];
    square.ccwFast4.linVelAbsDatetime(1:est0,:) = [];
    square.ccwFast4.linVelAbsData(1:est0,:) = [];
```

```
% CCW Slow 1
gnss0 = 1;
imu0 = 20;
est0 = 90;
square.ccwSlow1.gnssMapPoseDatetime(1:gnss0,:) = [];
square.ccwSlow1.gnssMapPoseData(1:gnss0,:) = [];
square.ccwSlow1.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.ccwSlow1.gnssRbtPoseData(1:gnss0,:) = [];
square.ccwSlow1.imuPoseDatetime(1:imu0,:) = [];
square.ccwSlow1.imuPoseData(1:imu0,:) = [];
square.ccwSlow1.imuTwistDatetime(1:imu0,:) = [];
square.ccwSlow1.imuTwistData(1:imu0,:) = [];
square.ccwSlow1.imuAccDatetime(1:imu0,:) = [];
square.ccwSlow1.imuAccData(1:imu0,:) = [];
square.ccwSlow1.estMapPoseDatetime(1:est0,:) = [];
square.ccwSlow1.estMapPoseData(1:est0,:) = [];
square.ccwSlow1.estRbtPoseDatetime(1:est0,:) = [];
square.ccwSlow1.estRbtPoseData(1:est0,:) = [];
square.ccwSlow1.estRbtTwistDatetime(1:est0,:) = [];
square.ccwSlow1.estRbtTwistData(1:est0,:) = [];
square.ccwSlow1.estRbtAccDatetime(1:est0,:) = [];
square.ccwSlow1.estRbtAccData(1:est0,:) = [];
square.ccwSlow1.linVelAbsDatetime(1:est0,:) = [];
square.ccwSlow1.linVelAbsData(1:est0,:) = [];

% CCW Slow 2
gnss0 = 1;
imu0 = 20;
est0 = 60;
square.ccwSlow2.gnssMapPoseDatetime(1:gnss0,:) = [];
square.ccwSlow2.gnssMapPoseData(1:gnss0,:) = [];
square.ccwSlow2.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.ccwSlow2.gnssRbtPoseData(1:gnss0,:) = [];
square.ccwSlow2.imuPoseDatetime(1:imu0,:) = [];
square.ccwSlow2.imuPoseData(1:imu0,:) = [];
square.ccwSlow2.imuTwistDatetime(1:imu0,:) = [];
square.ccwSlow2.imuTwistData(1:imu0,:) = [];
square.ccwSlow2.imuAccDatetime(1:imu0,:) = [];
square.ccwSlow2.imuAccData(1:imu0,:) = [];
square.ccwSlow2.estMapPoseDatetime(1:est0,:) = [];
square.ccwSlow2.estMapPoseData(1:est0,:) = [];
square.ccwSlow2.estRbtPoseDatetime(1:est0,:) = [];
square.ccwSlow2.estRbtPoseData(1:est0,:) = [];
square.ccwSlow2.estRbtTwistDatetime(1:est0,:) = [];
square.ccwSlow2.estRbtTwistData(1:est0,:) = [];
square.ccwSlow2.estRbtAccDatetime(1:est0,:) = [];
square.ccwSlow2.estRbtAccData(1:est0,:) = [];
square.ccwSlow2.linVelAbsDatetime(1:est0,:) = [];
square.ccwSlow2.linVelAbsData(1:est0,:) = [];

% CCW Slow 3
gnss0 = 0;
imu0 = 23;
est0 = 1;
square.ccwSlow3.gnssMapPoseDatetime(1:gnss0,:) = [];
square.ccwSlow3.gnssMapPoseData(1:gnss0,:) = [];
square.ccwSlow3.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.ccwSlow3.gnssRbtPoseData(1:gnss0,:) = [];
square.ccwSlow3.imuPoseDatetime(1:imu0,:) = [];
square.ccwSlow3.imuPoseData(1:imu0,:) = [];
square.ccwSlow3.imuTwistDatetime(1:imu0,:) = [];
square.ccwSlow3.imuTwistData(1:imu0,:) = [];
square.ccwSlow3.imuAccDatetime(1:imu0,:) = [];
square.ccwSlow3.imuAccData(1:imu0,:) = [];
square.ccwSlow3.estMapPoseDatetime(1:est0,:) = [];
square.ccwSlow3.estMapPoseData(1:est0,:) = [];
square.ccwSlow3.estRbtPoseDatetime(1:est0,:) = [];
square.ccwSlow3.estRbtPoseData(1:est0,:) = [];
square.ccwSlow3.estRbtTwistDatetime(1:est0,:) = [];
square.ccwSlow3.estRbtTwistData(1:est0,:) = [];
square.ccwSlow3.estRbtAccDatetime(1:est0,:) = [];
square.ccwSlow3.estRbtAccData(1:est0,:) = [];
square.ccwSlow3.linVelAbsDatetime(1:est0,:) = [];
square.ccwSlow3.linVelAbsData(1:est0,:) = [];

% CCW Slow 4
gnss0 = 1;
imu0 = 25;
est0 = 80;
square.ccwSlow4.gnssMapPoseDatetime(1:gnss0,:) = [];
square.ccwSlow4.gnssMapPoseData(1:gnss0,:) = [];
square.ccwSlow4.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.ccwSlow4.gnssRbtPoseData(1:gnss0,:) = [];
square.ccwSlow4.imuPoseDatetime(1:imu0,:) = [];
square.ccwSlow4.imuPoseData(1:imu0,:) = [];
square.ccwSlow4.imuTwistDatetime(1:imu0,:) = [];
square.ccwSlow4.imuTwistData(1:imu0,:) = [];
square.ccwSlow4.imuAccDatetime(1:imu0,:) = [];
square.ccwSlow4.imuAccData(1:imu0,:) = [];
square.ccwSlow4.estMapPoseDatetime(1:est0,:) = [];
square.ccwSlow4.estMapPoseData(1:est0,:) = [];
square.ccwSlow4.estRbtPoseDatetime(1:est0,:) = [];
square.ccwSlow4.estRbtPoseData(1:est0,:) = [];
square.ccwSlow4.estRbtTwistDatetime(1:est0,:) = [];
square.ccwSlow4.estRbtTwistData(1:est0,:) = [];
square.ccwSlow4.estRbtAccDatetime(1:est0,:) = [];
square.ccwSlow4.estRbtAccData(1:est0,:) = [];
square.ccwSlow4.linVelAbsDatetime(1:est0,:) = [];
square.ccwSlow4.linVelAbsData(1:est0,:) = [];

% CCW Slow 5
gnss0 = 1;
imu0 = 25;
est0 = 60;
square.ccwSlow5.gnssMapPoseDatetime(1:gnss0,:) = [];
square.ccwSlow5.gnssMapPoseData(1:gnss0,:) = [];
square.ccwSlow5.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.ccwSlow5.gnssRbtPoseData(1:gnss0,:) = [];
square.ccwSlow5.imuPoseDatetime(1:imu0,:) = [];
square.ccwSlow5.imuPoseData(1:imu0,:) = [];
square.ccwSlow5.imuTwistDatetime(1:imu0,:) = [];
square.ccwSlow5.imuTwistData(1:imu0,:) = [];
square.ccwSlow5.imuAccDatetime(1:imu0,:) = [];
square.ccwSlow5.imuAccData(1:imu0,:) = [];
square.ccwSlow5.estMapPoseDatetime(1:est0,:) = [];
square.ccwSlow5.estMapPoseData(1:est0,:) = [];
square.ccwSlow5.estRbtPoseDatetime(1:est0,:) = [];
square.ccwSlow5.estRbtPoseData(1:est0,:) = [];
square.ccwSlow5.estRbtTwistDatetime(1:est0,:) = [];
square.ccwSlow5.estRbtTwistData(1:est0,:) = [];
square.ccwSlow5.estRbtAccDatetime(1:est0,:) = [];
square.ccwSlow5.estRbtAccData(1:est0,:) = [];
square.ccwSlow5.linVelAbsDatetime(1:est0,:) = [];
square.ccwSlow5.linVelAbsData(1:est0,:) = [];

% CCW Slow 6
gnss0 = 2;
imu0 = 25;
est0 = 60;
square.ccwSlow6.gnssMapPoseDatetime(1:gnss0,:) = [];
square.ccwSlow6.gnssMapPoseData(1:gnss0,:) = [];
square.ccwSlow6.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.ccwSlow6.gnssRbtPoseData(1:gnss0,:) = [];
square.ccwSlow6.imuPoseDatetime(1:imu0,:) = [];
square.ccwSlow6.imuPoseData(1:imu0,:) = [];
square.ccwSlow6.imuTwistDatetime(1:imu0,:) = [];
square.ccwSlow6.imuTwistData(1:imu0,:) = [];
square.ccwSlow6.imuAccDatetime(1:imu0,:) = [];
square.ccwSlow6.imuAccData(1:imu0,:) = [];
square.ccwSlow6.estMapPoseDatetime(1:est0,:) = [];
square.ccwSlow6.estMapPoseData(1:est0,:) = [];
square.ccwSlow6.estRbtPoseDatetime(1:est0,:) = [];
square.ccwSlow6.estRbtPoseData(1:est0,:) = [];
square.ccwSlow6.estRbtTwistDatetime(1:est0,:) = [];
square.ccwSlow6.estRbtTwistData(1:est0,:) = [];
square.ccwSlow6.estRbtAccDatetime(1:est0,:) = [];
square.ccwSlow6.estRbtAccData(1:est0,:) = [];
square.ccwSlow6.linVelAbsDatetime(1:est0,:) = [];
square.ccwSlow6.linVelAbsData(1:est0,:) = [];

% CW Fast 1
gnss0 = 0;
imu0 = 25;
est0 = 120;
square.cwFast1.gnssMapPoseDatetime(1:gnss0,:) = [];
square.cwFast1.gnssMapPoseData(1:gnss0,:) = [];
square.cwFast1.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.cwFast1.gnssRbtPoseData(1:gnss0,:) = [];
square.cwFast1.imuPoseDatetime(1:imu0,:) = [];
square.cwFast1.imuPoseData(1:imu0,:) = [];
square.cwFast1.imuTwistDatetime(1:imu0,:) = [];
square.cwFast1.imuTwistData(1:imu0,:) = [];
square.cwFast1.imuAccDatetime(1:imu0,:) = [];
square.cwFast1.imuAccData(1:imu0,:) = [];
square.cwFast1.estMapPoseDatetime(1:est0,:) = [];
square.cwFast1.estMapPoseData(1:est0,:) = [];
square.cwFast1.estRbtPoseDatetime(1:est0,:) = [];
square.cwFast1.estRbtPoseData(1:est0,:) = [];
square.cwFast1.estRbtTwistDatetime(1:est0,:) = [];
square.cwFast1.estRbtTwistData(1:est0,:) = [];
square.cwFast1.estRbtAccDatetime(1:est0,:) = [];
square.cwFast1.estRbtAccData(1:est0,:) = [];
square.cwFast1.linVelAbsDatetime(1:est0,:) = [];
square.cwFast1.linVelAbsData(1:est0,:) = [];
% remove a bunch of estimator entries at the end
endRem = 60;
square.cwFast1.estMapPoseDatetime(end-endRem:end,:) = [];
square.cwFast1.estMapPoseData(end-endRem:end,:) = [];
square.cwFast1.estRbtPoseDatetime(end-endRem:end,:) = [];
square.cwFast1.estRbtPoseData(end-endRem:end,:) = [];
square.cwFast1.estRbtTwistDatetime(end-endRem:end,:) = [];
square.cwFast1.estRbtTwistData(end-endRem:end,:) = [];
square.cwFast1.estRbtAccDatetime(end-endRem:end,:) = [];
square.cwFast1.estRbtAccData(end-endRem:end,:) = [];
square.cwFast1.linVelAbsDatetime(end-endRem:end,:) = [];
square.cwFast1.linVelAbsData(end-endRem:end,:) = [];

% CW Fast 2
gnss0 = 1;
imu0 = 20;
est0 = 110;
square.cwFast2.gnssMapPoseDatetime(1:gnss0,:) = [];
square.cwFast2.gnssMapPoseData(1:gnss0,:) = [];
square.cwFast2.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.cwFast2.gnssRbtPoseData(1:gnss0,:) = [];
square.cwFast2.imuPoseDatetime(1:imu0,:) = [];
square.cwFast2.imuPoseData(1:imu0,:) = [];
square.cwFast2.imuTwistDatetime(1:imu0,:) = [];
square.cwFast2.imuTwistData(1:imu0,:) = [];
square.cwFast2.imuAccDatetime(1:imu0,:) = [];
square.cwFast2.imuAccData(1:imu0,:) = [];
square.cwFast2.estMapPoseDatetime(1:est0,:) = [];
square.cwFast2.estMapPoseData(1:est0,:) = [];
square.cwFast2.estRbtPoseDatetime(1:est0,:) = [];
square.cwFast2.estRbtPoseData(1:est0,:) = [];
square.cwFast2.estRbtTwistDatetime(1:est0,:) = [];
square.cwFast2.estRbtTwistData(1:est0,:) = [];
square.cwFast2.estRbtAccDatetime(1:est0,:) = [];
square.cwFast2.estRbtAccData(1:est0,:) = [];
square.cwFast2.linVelAbsDatetime(1:est0,:) = [];
square.cwFast2.linVelAbsData(1:est0,:) = [];
% remove a bunch of estimator entries at the end
endRem = 40;
square.cwFast2.estMapPoseDatetime(end-endRem:end,:) = [];
square.cwFast2.estMapPoseData(end-endRem:end,:) = [];
square.cwFast2.estRbtPoseDatetime(end-endRem:end,:) = [];
square.cwFast2.estRbtPoseData(end-endRem:end,:) = [];
square.cwFast2.estRbtTwistDatetime(end-endRem:end,:) = [];
square.cwFast2.estRbtTwistData(end-endRem:end,:) = [];
square.cwFast2.estRbtAccDatetime(end-endRem:end,:) = [];
square.cwFast2.estRbtAccData(end-endRem:end,:) = [];
square.cwFast2.linVelAbsDatetime(end-endRem:end,:) = [];
square.cwFast2.linVelAbsData(end-endRem:end,:) = [];

% CW Fast 3
gnss0 = 2;
imu0 = 22;
est0 = 80;
square.cwFast3.gnssMapPoseDatetime(1:gnss0,:) = [];
square.cwFast3.gnssMapPoseData(1:gnss0,:) = [];
square.cwFast3.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.cwFast3.gnssRbtPoseData(1:gnss0,:) = [];
square.cwFast3.imuPoseDatetime(1:imu0,:) = [];
square.cwFast3.imuPoseData(1:imu0,:) = [];
square.cwFast3.imuTwistDatetime(1:imu0,:) = [];
square.cwFast3.imuTwistData(1:imu0,:) = [];
square.cwFast3.imuAccDatetime(1:imu0,:) = [];
square.cwFast3.imuAccData(1:imu0,:) = [];
square.cwFast3.estMapPoseDatetime(1:est0,:) = [];
square.cwFast3.estMapPoseData(1:est0,:) = [];
square.cwFast3.estRbtPoseDatetime(1:est0,:) = [];
square.cwFast3.estRbtPoseData(1:est0,:) = [];
square.cwFast3.estRbtTwistDatetime(1:est0,:) = [];
square.cwFast3.estRbtTwistData(1:est0,:) = [];
square.cwFast3.estRbtAccDatetime(1:est0,:) = [];
square.cwFast3.estRbtAccData(1:est0,:) = [];
square.cwFast3.linVelAbsDatetime(1:est0,:) = [];
square.cwFast3.linVelAbsData(1:est0,:) = [];

% CW Fast 4
gnss0 = 1;
imu0 = 26;
est0 = 80;
square.cwFast4.gnssMapPoseDatetime(1:gnss0,:) = [];
square.cwFast4.gnssMapPoseData(1:gnss0,:) = [];
square.cwFast4.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.cwFast4.gnssRbtPoseData(1:gnss0,:) = [];
square.cwFast4.imuPoseDatetime(1:imu0,:) = [];
square.cwFast4.imuPoseData(1:imu0,:) = [];
square.cwFast4.imuTwistDatetime(1:imu0,:) = [];
square.cwFast4.imuTwistData(1:imu0,:) = [];
square.cwFast4.imuAccDatetime(1:imu0,:) = [];
square.cwFast4.imuAccData(1:imu0,:) = [];
square.cwFast4.estMapPoseDatetime(1:est0,:) = [];
square.cwFast4.estMapPoseData(1:est0,:) = [];
square.cwFast4.estRbtPoseDatetime(1:est0,:) = [];
square.cwFast4.estRbtPoseData(1:est0,:) = [];
square.cwFast4.estRbtTwistDatetime(1:est0,:) = [];
square.cwFast4.estRbtTwistData(1:est0,:) = [];
square.cwFast4.estRbtAccDatetime(1:est0,:) = [];
square.cwFast4.estRbtAccData(1:est0,:) = [];
square.cwFast4.linVelAbsDatetime(1:est0,:) = [];
square.cwFast4.linVelAbsData(1:est0,:) = [];

% CW Slow 1
gnss0 = 2;
imu0 = 25;
est0 = 90;
square.cwSlow1.gnssMapPoseDatetime(1:gnss0,:) = [];
square.cwSlow1.gnssMapPoseData(1:gnss0,:) = [];
square.cwSlow1.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.cwSlow1.gnssRbtPoseData(1:gnss0,:) = [];
square.cwSlow1.imuPoseDatetime(1:imu0,:) = [];
square.cwSlow1.imuPoseData(1:imu0,:) = [];
square.cwSlow1.imuTwistDatetime(1:imu0,:) = [];
square.cwSlow1.imuTwistData(1:imu0,:) = [];
square.cwSlow1.imuAccDatetime(1:imu0,:) = [];
square.cwSlow1.imuAccData(1:imu0,:) = [];
square.cwSlow1.estMapPoseDatetime(1:est0,:) = [];
square.cwSlow1.estMapPoseData(1:est0,:) = [];
square.cwSlow1.estRbtPoseDatetime(1:est0,:) = [];
square.cwSlow1.estRbtPoseData(1:est0,:) = [];
square.cwSlow1.estRbtTwistDatetime(1:est0,:) = [];
square.cwSlow1.estRbtTwistData(1:est0,:) = [];
square.cwSlow1.estRbtAccDatetime(1:est0,:) = [];
square.cwSlow1.estRbtAccData(1:est0,:) = [];
square.cwSlow1.linVelAbsDatetime(1:est0,:) = [];
square.cwSlow1.linVelAbsData(1:est0,:) = [];

% CW Slow 2
gnss0 = 1;
imu0 = 25;
est0 = 60;
square.cwSlow2.gnssMapPoseDatetime(1:gnss0,:) = [];
square.cwSlow2.gnssMapPoseData(1:gnss0,:) = [];
square.cwSlow2.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.cwSlow2.gnssRbtPoseData(1:gnss0,:) = [];
square.cwSlow2.imuPoseDatetime(1:imu0,:) = [];
square.cwSlow2.imuPoseData(1:imu0,:) = [];
square.cwSlow2.imuTwistDatetime(1:imu0,:) = [];
square.cwSlow2.imuTwistData(1:imu0,:) = [];
square.cwSlow2.imuAccDatetime(1:imu0,:) = [];
square.cwSlow2.imuAccData(1:imu0,:) = [];
square.cwSlow2.estMapPoseDatetime(1:est0,:) = [];
square.cwSlow2.estMapPoseData(1:est0,:) = [];
square.cwSlow2.estRbtPoseDatetime(1:est0,:) = [];
square.cwSlow2.estRbtPoseData(1:est0,:) = [];
square.cwSlow2.estRbtTwistDatetime(1:est0,:) = [];
square.cwSlow2.estRbtTwistData(1:est0,:) = [];
square.cwSlow2.estRbtAccDatetime(1:est0,:) = [];
square.cwSlow2.estRbtAccData(1:est0,:) = [];
square.cwSlow2.linVelAbsDatetime(1:est0,:) = [];
square.cwSlow2.linVelAbsData(1:est0,:) = [];

% CW Slow 3
gnss0 = 1;
imu0 = 25;
est0 = 100;
square.cwSlow3.gnssMapPoseDatetime(1:gnss0,:) = [];
square.cwSlow3.gnssMapPoseData(1:gnss0,:) = [];
square.cwSlow3.gnssRbtPoseDatetime(1:gnss0,:) = [];
square.cwSlow3.gnssRbtPoseData(1:gnss0,:) = [];
square.cwSlow3.imuPoseDatetime(1:imu0,:) = [];
square.cwSlow3.imuPoseData(1:imu0,:) = [];
square.cwSlow3.imuTwistDatetime(1:imu0,:) = [];
square.cwSlow3.imuTwistData(1:imu0,:) = [];
square.cwSlow3.imuAccDatetime(1:imu0,:) = [];
square.cwSlow3.imuAccData(1:imu0,:) = [];
square.cwSlow3.estMapPoseDatetime(1:est0,:) = [];
square.cwSlow3.estMapPoseData(1:est0,:) = [];
square.cwSlow3.estRbtPoseDatetime(1:est0,:) = [];
square.cwSlow3.estRbtPoseData(1:est0,:) = [];
square.cwSlow3.estRbtTwistDatetime(1:est0,:) = [];
square.cwSlow3.estRbtTwistData(1:est0,:) = [];
square.cwSlow3.estRbtAccDatetime(1:est0,:) = [];
square.cwSlow3.estRbtAccData(1:est0,:) = [];
square.cwSlow3.linVelAbsDatetime(1:est0,:) = [];
square.cwSlow3.linVelAbsData(1:est0,:) = [];
```

```
    % CW Slow 4
    gnss0 = 1;
    imu0 = 25;
    est0 = 100;
    square.cwSlow4.gnssMapPoseDatetime(1:gnss0,:) = [];
    square.cwSlow4.gnssMapPoseData(1:gnss0,:) = [];
    square.cwSlow4.gnssRbtPoseDatetime(1:gnss0,:) = [];
    square.cwSlow4.gnssRbtPoseData(1:gnss0,:) = [];
    square.cwSlow4.imuPoseDatetime(1:imu0,:) = [];
    square.cwSlow4.imuPoseData(1:imu0,:) = [];
    square.cwSlow4.imuTwistDatetime(1:imu0,:) = [];
    square.cwSlow4.imuTwistData(1:imu0,:) = [];
    square.cwSlow4.imuAccDatetime(1:imu0,:) = [];
    square.cwSlow4.imuAccData(1:imu0,:) = [];
    square.cwSlow4.estMapPoseDatetime(1:est0,:) = [];
    square.cwSlow4.estMapPoseData(1:est0,:) = [];
    square.cwSlow4.estRbtPoseDatetime(1:est0,:) = [];
    square.cwSlow4.estRbtPoseData(1:est0,:) = [];
    square.cwSlow4.estRbtTwistDatetime(1:est0,:) = [];
    square.cwSlow4.estRbtTwistData(1:est0,:) = [];
    square.cwSlow4.estRbtAccDatetime(1:est0,:) = [];
    square.cwSlow4.estRbtAccData(1:est0,:) = [];
    square.cwSlow4.linVelAbsDatetime(1:est0,:) = [];
    square.cwSlow4.linVelAbsData(1:est0,:) = [];

end

function square = zeroMapData(square)

    % CCW Fast 1
    mapZero = square.ccwFast1.gnssMapPoseData(1,:);
    square.ccwFast1.gnssMapPoseData =...
        square.ccwFast1.gnssMapPoseData-mapZero;
    square.ccwFast1.estMapPoseData =...
        square.ccwFast1.estMapPoseData-mapZero;
    rbtZero = square.ccwFast1.gnssRbtPoseData(1,:);
    square.ccwFast1.gnssRbtPoseData =...
        square.ccwFast1.gnssRbtPoseData-rbtZero;
    square.ccwFast1.estRbtPoseData =...
        square.ccwFast1.estRbtPoseData-rbtZero;

    % CCW Fast 2
    mapZero = square.ccwFast2.gnssMapPoseData(1,:);
    square.ccwFast2.gnssMapPoseData =...
        square.ccwFast2.gnssMapPoseData-mapZero;
    square.ccwFast2.estMapPoseData =...
        square.ccwFast2.estMapPoseData-mapZero;
    rbtZero = square.ccwFast2.gnssRbtPoseData(1,:);
    square.ccwFast2.gnssRbtPoseData =...
        square.ccwFast2.gnssRbtPoseData-rbtZero;
    square.ccwFast2.estRbtPoseData =...
        square.ccwFast2.estRbtPoseData-rbtZero;

    % CCW Fast 3
    mapZero = square.ccwFast3.gnssMapPoseData(1,:);
    square.ccwFast3.gnssMapPoseData =...
        square.ccwFast3.gnssMapPoseData-mapZero;
    square.ccwFast3.estMapPoseData =...
        square.ccwFast3.estMapPoseData-mapZero;
    rbtZero = square.ccwFast3.gnssRbtPoseData(1,:);
    square.ccwFast3.gnssRbtPoseData =...
        square.ccwFast3.gnssRbtPoseData-rbtZero;
    square.ccwFast3.estRbtPoseData =...
        square.ccwFast3.estRbtPoseData-rbtZero;

    % CCW Fast 4
    mapZero = square.ccwFast4.gnssMapPoseData(1,:);
    square.ccwFast4.gnssMapPoseData =...
        square.ccwFast4.gnssMapPoseData-mapZero;
    square.ccwFast4.estMapPoseData =...
        square.ccwFast4.estMapPoseData-mapZero;
    rbtZero = square.ccwFast4.gnssRbtPoseData(1,:);
    square.ccwFast4.gnssRbtPoseData =...
        square.ccwFast4.gnssRbtPoseData-rbtZero;
    square.ccwFast4.estRbtPoseData =...
        square.ccwFast4.estRbtPoseData-rbtZero;

    % CCW Slow 1
    mapZero = square.ccwSlow1.gnssMapPoseData(1,:);
    square.ccwSlow1.gnssMapPoseData =...
        square.ccwSlow1.gnssMapPoseData-mapZero;
    square.ccwSlow1.estMapPoseData =...
        square.ccwSlow1.estMapPoseData-mapZero;
    rbtZero = square.ccwSlow1.gnssRbtPoseData(1,:);
    square.ccwSlow1.gnssRbtPoseData =...
        square.ccwSlow1.gnssRbtPoseData-rbtZero;
    square.ccwSlow1.estRbtPoseData =...
        square.ccwSlow1.estRbtPoseData-rbtZero;

    % CCW Slow 2
    mapZero = square.ccwSlow2.gnssMapPoseData(1,:);
    square.ccwSlow2.gnssMapPoseData =...
        square.ccwSlow2.gnssMapPoseData-mapZero;
    square.ccwSlow2.estMapPoseData =...
        square.ccwSlow2.estMapPoseData-mapZero;
    rbtZero = square.ccwSlow2.gnssRbtPoseData(1,:);
    square.ccwSlow2.gnssRbtPoseData =...
        square.ccwSlow2.gnssRbtPoseData-rbtZero;
    square.ccwSlow2.estRbtPoseData =...
        square.ccwSlow2.estRbtPoseData-rbtZero;

    % CCW Slow 3
    mapZero = square.ccwSlow3.gnssMapPoseData(1,:);
    square.ccwSlow3.gnssMapPoseData =...
        square.ccwSlow3.gnssMapPoseData-mapZero;
    square.ccwSlow3.estMapPoseData =...
        square.ccwSlow3.estMapPoseData-mapZero;
    rbtZero = square.ccwSlow3.gnssRbtPoseData(1,:);
    square.ccwSlow3.gnssRbtPoseData =...
        square.ccwSlow3.gnssRbtPoseData-rbtZero;
    square.ccwSlow3.estRbtPoseData =...
        square.ccwSlow3.estRbtPoseData-rbtZero;

    % CCW Slow 4
    mapZero = square.ccwSlow4.gnssMapPoseData(1,:);
    square.ccwSlow4.gnssMapPoseData =...
        square.ccwSlow4.gnssMapPoseData-mapZero;
    square.ccwSlow4.estMapPoseData =...
        square.ccwSlow4.estMapPoseData-mapZero;
    rbtZero = square.ccwSlow4.gnssRbtPoseData(1,:);
    square.ccwSlow4.gnssRbtPoseData =...
        square.ccwSlow4.gnssRbtPoseData-rbtZero;
    square.ccwSlow4.estRbtPoseData =...
        square.ccwSlow4.estRbtPoseData-rbtZero;

    % CCW Slow 5
    mapZero = square.ccwSlow5.gnssMapPoseData(1,:);
    square.ccwSlow5.gnssMapPoseData =...
        square.ccwSlow5.gnssMapPoseData-mapZero;
    square.ccwSlow5.estMapPoseData =...
        square.ccwSlow5.estMapPoseData-mapZero;
    rbtZero = square.ccwSlow5.gnssRbtPoseData(1,:);
    square.ccwSlow5.gnssRbtPoseData =...
        square.ccwSlow5.gnssRbtPoseData-rbtZero;
    square.ccwSlow5.estRbtPoseData =...
        square.ccwSlow5.estRbtPoseData-rbtZero;

    % CCW Slow 6
    mapZero = square.ccwSlow6.gnssMapPoseData(1,:);
    square.ccwSlow6.gnssMapPoseData =...
```

```
    square.ccwSlow6.gnssMapPoseData-mapZero;
square.ccwSlow6.estMapPoseData =...
    square.ccwSlow6.estMapPoseData-mapZero;
rbtZero = square.ccwSlow6.gnssRbtPoseData(1,:);
square.ccwSlow6.gnssRbtPoseData =...
    square.ccwSlow6.gnssRbtPoseData-rbtZero;
square.ccwSlow6.estRbtPoseData =...
    square.ccwSlow6.estRbtPoseData-rbtZero;

estCor = square.ccwSlow6.imuPoseData(1,3)-...
    square.ccwSlow6.estMapPoseData(1,3);
square.ccwSlow6.estMapPoseData(:,3) =...
    square.ccwSlow6.estMapPoseData(:,3)+estCor;
square.ccwSlow6.estRbtPoseData(:,3) =...
    square.ccwSlow6.estRbtPoseData(:,3)+estCor;


% CW Fast 1
mapZero = square.cwFast1.gnssMapPoseData(1,:);
square.cwFast1.gnssMapPoseData =...
    square.cwFast1.gnssMapPoseData-mapZero;
square.cwFast1.estMapPoseData =...
    square.cwFast1.estMapPoseData-mapZero;
rbtZero = square.cwFast1.gnssRbtPoseData(1,:);
square.cwFast1.gnssRbtPoseData =...
    square.cwFast1.gnssRbtPoseData-rbtZero;
square.cwFast1.estRbtPoseData =...
    square.cwFast1.estRbtPoseData-rbtZero;

% CW Fast 2
mapZero = square.cwFast2.gnssMapPoseData(1,:);
square.cwFast2.gnssMapPoseData =...
    square.cwFast2.gnssMapPoseData-mapZero;
square.cwFast2.estMapPoseData =...
    square.cwFast2.estMapPoseData-mapZero;
rbtZero = square.cwFast2.gnssRbtPoseData(1,:);
square.cwFast2.gnssRbtPoseData =...
    square.cwFast2.gnssRbtPoseData-rbtZero;
square.cwFast2.estRbtPoseData =...
    square.cwFast2.estRbtPoseData-rbtZero;

% CW Fast 3
mapZero = square.cwFast3.gnssMapPoseData(1,:);
square.cwFast3.gnssMapPoseData =...
    square.cwFast3.gnssMapPoseData-mapZero;
square.cwFast3.estMapPoseData =...
    square.cwFast3.estMapPoseData-mapZero;
rbtZero = square.cwFast3.gnssRbtPoseData(1,:);
square.cwFast3.gnssRbtPoseData =...
    square.cwFast3.gnssRbtPoseData-rbtZero;
square.cwFast3.estRbtPoseData =...
    square.cwFast3.estRbtPoseData-rbtZero;

% CW Fast 4
mapZero = square.cwFast4.gnssMapPoseData(1,:);
square.cwFast4.gnssMapPoseData =...
    square.cwFast4.gnssMapPoseData-mapZero;
square.cwFast4.estMapPoseData =...
    square.cwFast4.estMapPoseData-mapZero;
rbtZero = square.cwFast4.gnssRbtPoseData(1,:);
square.cwFast4.gnssRbtPoseData =...
    square.cwFast4.gnssRbtPoseData-rbtZero;
square.cwFast4.estRbtPoseData =...
    square.cwFast4.estRbtPoseData-rbtZero;

% CW Slow 1
mapZero = square.cwSlow1.gnssMapPoseData(1,:);
square.cwSlow1.gnssMapPoseData =...
    square.cwSlow1.gnssMapPoseData-mapZero;
square.cwSlow1.estMapPoseData =...
```

```
        square.cwSlow1.estMapPoseData-mapZero;
    rbtZero = square.cwSlow1.gnssRbtPoseData(1,:);
    square.cwSlow1.gnssRbtPoseData =...
        square.cwSlow1.gnssRbtPoseData-rbtZero;
    square.cwSlow1.estRbtPoseData =...
        square.cwSlow1.estRbtPoseData-rbtZero;

    % CW Slow 2
    mapZero = square.cwSlow2.gnssMapPoseData(1,:);
    square.cwSlow2.gnssMapPoseData =...
        square.cwSlow2.gnssMapPoseData-mapZero;
    square.cwSlow2.estMapPoseData =...
        square.cwSlow2.estMapPoseData-mapZero;
    rbtZero = square.cwSlow2.gnssRbtPoseData(1,:);
    square.cwSlow2.gnssRbtPoseData =...
        square.cwSlow2.gnssRbtPoseData-rbtZero;
    square.cwSlow2.estRbtPoseData =...
        square.cwSlow2.estRbtPoseData-rbtZero;

    % CW Slow 3
    mapZero = square.cwSlow3.gnssMapPoseData(1,:);
    square.cwSlow3.gnssMapPoseData =...
        square.cwSlow3.gnssMapPoseData-mapZero;
    square.cwSlow3.estMapPoseData =...
        square.cwSlow3.estMapPoseData-mapZero;
    rbtZero = square.cwSlow3.gnssRbtPoseData(1,:);
    square.cwSlow3.gnssRbtPoseData =...
        square.cwSlow3.gnssRbtPoseData-rbtZero;
    square.cwSlow3.estRbtPoseData =...
        square.cwSlow3.estRbtPoseData-rbtZero;

    % CW Slow 4
    mapZero = square.cwSlow4.gnssMapPoseData(1,:);
    square.cwSlow4.gnssMapPoseData =...
        square.cwSlow4.gnssMapPoseData-mapZero;
    square.cwSlow4.estMapPoseData =...
        square.cwSlow4.estMapPoseData-mapZero;
    rbtZero = square.cwSlow4.gnssRbtPoseData(1,:);
    square.cwSlow4.gnssRbtPoseData =...
        square.cwSlow4.gnssRbtPoseData-rbtZero;
    square.cwSlow4.estRbtPoseData =...
        square.cwSlow4.estRbtPoseData-rbtZero;

    % Save to file
    save('squareCorrected.mat','square');

end

function square = createGroundTruthRectangle(square)

    % Properties of the rectangle (can't change with arguments)
    longLeg = 24.8;
    shortLeg = 18.6;
    per = longLeg*2+shortLeg*2;
    rot = deg2rad(-6);

    % CCW Fast 1 (28 s)
    start = [0,0];
    runTime = 28;
    res = 1E4;
    c1 = start+shortLeg*[-sin(rot),cos(rot)];
    c2 = c1+longLeg*[-cos(rot),-sin(rot)];
    c3 = c2+shortLeg*[sin(rot),-cos(rot)];
    c4 = c3+longLeg*[cos(rot),sin(rot)];
    timeVec = linspace(0,runTime,res)';
    [row,~] = size(timeVec);
    shortTimeVecLength = round(row*shortLeg/per);
    longTimeVecLength = round(row*longLeg/per);
    s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
    s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
```

```
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwFast1.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwFast1.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwFast1.groundTruthDatetime = dt;

% CCW Fast 2 (28 s)
start = [0,0];
runTime = 28;
res = 1E4;
c1 = start+shortLeg*[-sin(rot),cos(rot)];
c2 = c1+longLeg*[-cos(rot),-sin(rot)];
c3 = c2+shortLeg*[sin(rot),-cos(rot)];
c4 = c3+longLeg*[cos(rot),sin(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwFast2.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwFast2.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwFast2.groundTruthDatetime = dt;

% CCW Fast 3 (28 s)
start = [0,0];
runTime = 28;
res = 1E4;
c1 = start+shortLeg*[-sin(rot),cos(rot)];
c2 = c1+longLeg*[-cos(rot),-sin(rot)];
c3 = c2+shortLeg*[sin(rot),-cos(rot)];
c4 = c3+longLeg*[cos(rot),sin(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
```

```
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwFast3.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwFast3.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwFast3.groundTruthDatetime = dt;

% CCW Fast 4 (30.5 s)
start = [0,0];
runTime = 30.5;
res = 1E4;
c1 = start+shortLeg*[-sin(rot),cos(rot)];
c2 = c1+longLeg*[-cos(rot),-sin(rot)];
c3 = c2+shortLeg*[sin(rot),-cos(rot)];
c4 = c3+longLeg*[cos(rot),sin(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwFast4.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwFast4.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwFast4.groundTruthDatetime = dt;

% CCW Slow 1 (62.5 s)
start = [0,0];
runTime = 62.5;
res = 1E4;
c1 = start+shortLeg*[-sin(rot),cos(rot)];
c2 = c1+longLeg*[-cos(rot),-sin(rot)];
c3 = c2+shortLeg*[sin(rot),-cos(rot)];
c4 = c3+longLeg*[cos(rot),sin(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwSlow1.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwSlow1.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
```

```
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwSlow1.groundTruthDatetime = dt;

% CCW Slow 2 (63 s)
start = [0,0];
runTime = 63;
res = 1E4;
c1 = start+shortLeg*[-sin(rot),cos(rot)];
c2 = c1+longLeg*[-cos(rot),-sin(rot)];
c3 = c2+shortLeg*[sin(rot),-cos(rot)];
c4 = c3+longLeg*[cos(rot),sin(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwSlow2.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwSlow2.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwSlow2.groundTruthDatetime = dt;

% CCW Slow 3 (61 s)
start = [0,0];
runTime = 61;
res = 1E4;
c1 = start+shortLeg*[-sin(rot),cos(rot)];
c2 = c1+longLeg*[-cos(rot),-sin(rot)];
c3 = c2+shortLeg*[sin(rot),-cos(rot)];
c4 = c3+longLeg*[cos(rot),sin(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwSlow3.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwSlow3.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
```

```
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwSlow3.groundTruthDatetime = dt;

% CCW Slow 4 (67.5 s, 1.3 m/s)
start = [0,0];
runTime = 67.5;
res = 1E4;
c1 = start+shortLeg*[-sin(rot),cos(rot)];
c2 = c1+longLeg*[-cos(rot),-sin(rot)];
c3 = c2+shortLeg*[sin(rot),-cos(rot)];
c4 = c3+longLeg*[cos(rot),sin(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwSlow4.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwSlow4.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwSlow4.groundTruthDatetime = dt;

% CCW Slow 5 (68 s)
start = [0,0];
runTime = 68;
res = 1E4;
c1 = start+shortLeg*[-sin(rot),cos(rot)];
c2 = c1+longLeg*[-cos(rot),-sin(rot)];
c3 = c2+shortLeg*[sin(rot),-cos(rot)];
c4 = c3+longLeg*[cos(rot),sin(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwSlow5.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwSlow5.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwSlow5.groundTruthDatetime = dt;

% CCW Slow 6 (66 s)
start = [0,0];
runTime = 66;
res = 1E4;
c1 = start+shortLeg*[-sin(rot),cos(rot)];
c2 = c1+longLeg*[-cos(rot),-sin(rot)];
c3 = c2+shortLeg*[sin(rot),-cos(rot)];
c4 = c3+longLeg*[cos(rot),sin(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.ccwSlow6.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.ccwSlow6.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.ccwSlow6.groundTruthDatetime = dt;

% CW Fast 1 (29.5 s)
start = [0,0];
runTime = 29.5;
res = 1E4;
c1 = start+longLeg*[-cos(rot),-sin(rot)];
c2 = c1+shortLeg*[-sin(rot),cos(rot)];
c3 = c2+longLeg*[cos(rot),sin(rot)];
c4 = c3+shortLeg*[sin(rot),-cos(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.cwFast1.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.cwFast1.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.cwFast1.groundTruthDatetime = dt;

% CW Fast 2 (28 s)
```

```
start = [0,0];
runTime = 28;
res = 1E4;
c1 = start+longLeg*[-cos(rot),-sin(rot)];
c2 = c1+shortLeg*[-sin(rot),cos(rot)];
c3 = c2+longLeg*[cos(rot),sin(rot)];
c4 = c3+shortLeg*[sin(rot),-cos(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.cwFast2.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.cwFast2.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.cwFast2.groundTruthDatetime = dt;

% CW Fast 3 (29.5 s)
start = [0,0];
runTime = 29.5;
res = 1E4;
c1 = start+longLeg*[-cos(rot),-sin(rot)];
c2 = c1+shortLeg*[-sin(rot),cos(rot)];
c3 = c2+longLeg*[cos(rot),sin(rot)];
c4 = c3+shortLeg*[sin(rot),-cos(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.cwFast3.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.cwFast3.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.cwFast3.groundTruthDatetime = dt;

% CW Fast 4 (30.5 s)
start = [0,0];
runTime = 30.5;
res = 1E4;
c1 = start+longLeg*[-cos(rot),-sin(rot)];
```

```
c2 = c1+shortLeg*[-sin(rot),cos(rot)];
c3 = c2+longLeg*[cos(rot),sin(rot)];
c4 = c3+shortLeg*[sin(rot),-cos(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.cwFast4.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.cwFast4.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.cwFast4.groundTruthDatetime = dt;

% CW Slow 1 (60.5 s)
start = [0,0];
runTime = 60.5;
res = 1E4;
c1 = start+longLeg*[-cos(rot),-sin(rot)];
c2 = c1+shortLeg*[-sin(rot),cos(rot)];
c3 = c2+longLeg*[cos(rot),sin(rot)];
c4 = c3+shortLeg*[sin(rot),-cos(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.cwSlow1.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.cwSlow1.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.cwSlow1.groundTruthDatetime = dt;

% CW Slow 2 (57 s)
start = [0,0];
runTime = 57;
res = 1E4;
c1 = start+longLeg*[-cos(rot),-sin(rot)];
c2 = c1+shortLeg*[-sin(rot),cos(rot)];
c3 = c2+longLeg*[cos(rot),sin(rot)];
c4 = c3+shortLeg*[sin(rot),-cos(rot)];
timeVec = linspace(0,runTime,res)';
```

```
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.cwSlow2.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.cwSlow2.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.cwSlow2.groundTruthDatetime = dt;

% CW Slow 3 (55 s)
start = [0,0];
runTime = 55;
res = 1E4;
c1 = start+longLeg*[-cos(rot),-sin(rot)];
c2 = c1+shortLeg*[-sin(rot),cos(rot)];
c3 = c2+longLeg*[cos(rot),sin(rot)];
c4 = c3+shortLeg*[sin(rot),-cos(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
square.cwSlow3.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
square.cwSlow3.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
square.cwSlow3.groundTruthDatetime = dt;

% CW Slow 4 (64.5 s)
start = [0,0];
runTime = 64.5;
res = 1E4;
c1 = start+longLeg*[-cos(rot),-sin(rot)];
c2 = c1+shortLeg*[-sin(rot),cos(rot)];
c3 = c2+longLeg*[cos(rot),sin(rot)];
c4 = c3+shortLeg*[sin(rot),-cos(rot)];
timeVec = linspace(0,runTime,res)';
[row,~] = size(timeVec);
shortTimeVecLength = round(row*shortLeg/per);
longTimeVecLength = round(row*longLeg/per);
s1x = linspace(start(1,1),c1(1,1),shortTimeVecLength)';
```

```
    s1y = linspace(start(1,2),c1(1,2),shortTimeVecLength)';
    s2x = linspace(c1(1,1),c2(1,1),longTimeVecLength)';
    s2y = linspace(c1(1,2),c2(1,2),longTimeVecLength)';
    s3x = linspace(c2(1,1),c3(1,1),shortTimeVecLength)';
    s3y = linspace(c2(1,2),c3(1,2),shortTimeVecLength)';
    s4x = linspace(c3(1,1),c4(1,1),longTimeVecLength)';
    s4y = linspace(c3(1,2),c4(1,2),longTimeVecLength)';
    square.cwSlow4.groundTruthData(:,1) = [s1x;s2x;s3x;s4x];
    square.cwSlow4.groundTruthData(:,2) = [s1y;s2y;s3y;s4y];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    square.cwSlow4.groundTruthDatetime = dt;

    % Save to file
    save('squareCorrectedWithRectangle.mat','square');

end

function square = calculateAbsoluteError(square)

    % CCW Fast 1 pose accuracy
    square.ccwFast1.groundTruthTimetable =...
        array2timetable(square.ccwFast1.groundTruthData,...
        'RowTimes',square.ccwFast1.groundTruthDatetime);        % create timetable for
groundTruth
    square.ccwFast1.gnssMapPoseTimetable =...
        array2timetable(square.ccwFast1.gnssMapPoseData,...
        'RowTimes',square.ccwFast1.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwFast1.estMapPoseTimetable =...
        array2timetable(square.ccwFast1.estMapPoseData,...
        'RowTimes',square.ccwFast1.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwFast1.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwFast1.gnssMapPoseTimetable,...
        square.ccwFast1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.ccwFast1.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwFast1.estMapPoseTimetable,...
        square.ccwFast1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.ccwFast1.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwFast1.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwFast1.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwFast1.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwFast1.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwFast1.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwFast1.absErrorMeanGnssMapXY =...
        sqrt(square.ccwFast1.absErrorMeanGnssMapX^2+...
        square.ccwFast1.absErrorMeanGnssMapY^2);
    square.ccwFast1.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwFast1.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwFast1.absErrorStdGnssMapXY =...
        sqrt(square.ccwFast1.absErrorStdGnssMapX^2+...
        square.ccwFast1.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwFast1.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
```

```
        square.ccwFast1.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwFast1.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwFast1.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwFast1.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwFast1.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwFast1.absErrorMeanEstMapXY =...
        sqrt(square.ccwFast1.absErrorMeanEstMapX^2+...
        square.ccwFast1.absErrorMeanEstMapY^2);
    square.ccwFast1.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwFast1.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwFast1.absErrorStdEstMapXY =...
        sqrt(square.ccwFast1.absErrorStdEstMapX^2+...
        square.ccwFast1.absErrorStdEstMapY^2);
    square.ccwFast1.gnssToEstMeanErrorChange =...
        (square.ccwFast1.absErrorMeanEstMapXY-...
        square.ccwFast1.absErrorMeanGnssMapXY)/...
        square.ccwFast1.absErrorMeanGnssMapXY;
    square.ccwFast1.gnssToEstStdErrorChange =...
        (square.ccwFast1.absErrorStdEstMapXY-...
        square.ccwFast1.absErrorStdGnssMapXY)/...
        square.ccwFast1.absErrorStdGnssMapXY;

    % CCW Fast 2 pose accuracy
    square.ccwFast2.groundTruthTimetable =...
        array2timetable(square.ccwFast2.groundTruthData,...
        'RowTimes',square.ccwFast2.groundTruthDatetime);        % create timetable for
groundTruth
    square.ccwFast2.gnssMapPoseTimetable =...
        array2timetable(square.ccwFast2.gnssMapPoseData,...
        'RowTimes',square.ccwFast2.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwFast2.estMapPoseTimetable =...
        array2timetable(square.ccwFast2.estMapPoseData,...
        'RowTimes',square.ccwFast2.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwFast2.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwFast2.gnssMapPoseTimetable,...
        square.ccwFast2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.ccwFast2.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwFast2.estMapPoseTimetable,...
        square.ccwFast2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.ccwFast2.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwFast2.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwFast2.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwFast2.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwFast2.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwFast2.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwFast2.absErrorMeanGnssMapXY =...
        sqrt(square.ccwFast2.absErrorMeanGnssMapX^2+...
        square.ccwFast2.absErrorMeanGnssMapY^2);
    square.ccwFast2.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwFast2.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwFast2.absErrorStdGnssMapXY =...
        sqrt(square.ccwFast2.absErrorStdGnssMapX^2+...
        square.ccwFast2.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwFast2.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwFast2.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwFast2.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwFast2.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwFast2.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwFast2.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwFast2.absErrorMeanEstMapXY =...
        sqrt(square.ccwFast2.absErrorMeanEstMapX^2+...
        square.ccwFast2.absErrorMeanEstMapY^2);
    square.ccwFast2.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwFast2.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwFast2.absErrorStdEstMapXY =...
        sqrt(square.ccwFast2.absErrorStdEstMapX^2+...
        square.ccwFast2.absErrorStdEstMapY^2);
    square.ccwFast2.gnssToEstMeanErrorChange =...
        (square.ccwFast2.absErrorMeanEstMapXY-...
        square.ccwFast2.absErrorMeanGnssMapXY)/...
        square.ccwFast2.absErrorMeanGnssMapXY;
    square.ccwFast2.gnssToEstStdErrorChange =...
        (square.ccwFast2.absErrorStdEstMapXY-...
        square.ccwFast2.absErrorStdGnssMapXY)/...
        square.ccwFast2.absErrorStdGnssMapXY;

    % CCW Fast 3 pose accuracy
    square.ccwFast3.groundTruthTimetable =...
        array2timetable(square.ccwFast3.groundTruthData,...
        'RowTimes',square.ccwFast3.groundTruthDatetime);        % create timetable for
groundTruth
    square.ccwFast3.gnssMapPoseTimetable =...
        array2timetable(square.ccwFast3.gnssMapPoseData,...
        'RowTimes',square.ccwFast3.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwFast3.estMapPoseTimetable =...
        array2timetable(square.ccwFast3.estMapPoseData,...
        'RowTimes',square.ccwFast3.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwFast3.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwFast3.gnssMapPoseTimetable,...
        square.ccwFast3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.ccwFast3.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwFast3.estMapPoseTimetable,...
        square.ccwFast3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.ccwFast3.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwFast3.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwFast3.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwFast3.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwFast3.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwFast3.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwFast3.absErrorMeanGnssMapXY =...
        sqrt(square.ccwFast3.absErrorMeanGnssMapX^2+...
        square.ccwFast3.absErrorMeanGnssMapY^2);
    square.ccwFast3.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwFast3.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwFast3.absErrorStdGnssMapXY =...
        sqrt(square.ccwFast3.absErrorStdGnssMapX^2+...
        square.ccwFast3.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwFast3.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwFast3.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
```

```
        square.ccwFast3.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwFast3.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwFast3.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwFast3.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwFast3.absErrorMeanEstMapXY =...
        sqrt(square.ccwFast3.absErrorMeanEstMapX^2+...
        square.ccwFast3.absErrorMeanEstMapY^2);
    square.ccwFast3.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwFast3.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwFast3.absErrorStdEstMapXY =...
        sqrt(square.ccwFast3.absErrorStdEstMapX^2+...
        square.ccwFast3.absErrorStdEstMapY^2);
    square.ccwFast3.gnssToEstMeanErrorChange =...
        (square.ccwFast3.absErrorMeanEstMapXY-...
        square.ccwFast3.absErrorMeanGnssMapXY)/...
        square.ccwFast3.absErrorMeanGnssMapXY;
    square.ccwFast3.gnssToEstStdErrorChange =...
        (square.ccwFast3.absErrorStdEstMapXY-...
        square.ccwFast3.absErrorStdGnssMapXY)/...
        square.ccwFast3.absErrorStdGnssMapXY;

    % CCW Fast 4 pose accuracy
    square.ccwFast4.groundTruthTimetable =...
        array2timetable(square.ccwFast4.groundTruthData,...
        'RowTimes',square.ccwFast4.groundTruthDatetime);        % create timetable for
groundTruth
    square.ccwFast4.gnssMapPoseTimetable =...
        array2timetable(square.ccwFast4.gnssMapPoseData,...
        'RowTimes',square.ccwFast4.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwFast4.estMapPoseTimetable =...
        array2timetable(square.ccwFast4.estMapPoseData,...
        'RowTimes',square.ccwFast4.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwFast4.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwFast4.gnssMapPoseTimetable,...
        square.ccwFast4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.ccwFast4.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwFast4.estMapPoseTimetable,...
        square.ccwFast4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.ccwFast4.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwFast4.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwFast4.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwFast4.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwFast4.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwFast4.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwFast4.absErrorMeanGnssMapXY =...
        sqrt(square.ccwFast4.absErrorMeanGnssMapX^2+...
        square.ccwFast4.absErrorMeanGnssMapY^2);
    square.ccwFast4.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwFast4.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwFast4.absErrorStdGnssMapXY =...
        sqrt(square.ccwFast4.absErrorStdGnssMapX^2+...
        square.ccwFast4.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwFast4.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwFast4.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwFast4.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwFast4.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwFast4.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwFast4.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwFast4.absErrorMeanEstMapXY =...
        sqrt(square.ccwFast4.absErrorMeanEstMapX^2+...
        square.ccwFast4.absErrorMeanEstMapY^2);
    square.ccwFast4.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwFast4.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwFast4.absErrorStdEstMapXY =...
        sqrt(square.ccwFast4.absErrorStdEstMapX^2+...
        square.ccwFast4.absErrorStdEstMapY^2);
    square.ccwFast4.gnssToEstMeanErrorChange =...
        (square.ccwFast4.absErrorMeanEstMapXY-...
        square.ccwFast4.absErrorMeanGnssMapXY)/...
        square.ccwFast4.absErrorMeanGnssMapXY;
    square.ccwFast4.gnssToEstStdErrorChange =...
        (square.ccwFast4.absErrorStdEstMapXY-...
        square.ccwFast4.absErrorStdGnssMapXY)/...
        square.ccwFast4.absErrorStdGnssMapXY;

    % CCW Slow 1 pose accuracy
    square.ccwSlow1.groundTruthTimetable =...
        array2timetable(square.ccwSlow1.groundTruthData,...
        'RowTimes',square.ccwSlow1.groundTruthDatetime);        % create timetable for
groundTruth
    square.ccwSlow1.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow1.gnssMapPoseData,...
        'RowTimes',square.ccwSlow1.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow1.estMapPoseTimetable =...
        array2timetable(square.ccwSlow1.estMapPoseData,...
        'RowTimes',square.ccwSlow1.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow1.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwSlow1.gnssMapPoseTimetable,...
        square.ccwSlow1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.ccwSlow1.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwSlow1.estMapPoseTimetable,...
        square.ccwSlow1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.ccwSlow1.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow1.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwSlow1.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow1.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwSlow1.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwSlow1.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwSlow1.absErrorMeanGnssMapXY =...
        sqrt(square.ccwSlow1.absErrorMeanGnssMapX^2+...
        square.ccwSlow1.absErrorMeanGnssMapY^2);
    square.ccwSlow1.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwSlow1.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwSlow1.absErrorStdGnssMapXY =...
        sqrt(square.ccwSlow1.absErrorStdGnssMapX^2+...
        square.ccwSlow1.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwSlow1.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow1.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow1.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
```

```
        square.ccwSlow1.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwSlow1.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwSlow1.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwSlow1.absErrorMeanEstMapXY =...
        sqrt(square.ccwSlow1.absErrorMeanEstMapX^2+...
        square.ccwSlow1.absErrorMeanEstMapY^2);
    square.ccwSlow1.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwSlow1.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwSlow1.absErrorStdEstMapXY =...
        sqrt(square.ccwSlow1.absErrorStdEstMapX^2+...
        square.ccwSlow1.absErrorStdEstMapY^2);
    square.ccwSlow1.gnssToEstMeanErrorChange =...
        (square.ccwSlow1.absErrorMeanEstMapXY-...
        square.ccwSlow1.absErrorMeanGnssMapXY)/...
        square.ccwSlow1.absErrorMeanGnssMapXY;
    square.ccwSlow1.gnssToEstStdErrorChange =...
        (square.ccwSlow1.absErrorStdEstMapXY-...
        square.ccwSlow1.absErrorStdGnssMapXY)/...
        square.ccwSlow1.absErrorStdGnssMapXY;

    % CCW Slow 2 pose accuracy
    square.ccwSlow2.groundTruthTimetable =...
        array2timetable(square.ccwSlow2.groundTruthData,...
        'RowTimes',square.ccwSlow2.groundTruthDatetime);        % create timetable for
groundTruth
    square.ccwSlow2.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow2.gnssMapPoseData,...
        'RowTimes',square.ccwSlow2.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow2.estMapPoseTimetable =...
        array2timetable(square.ccwSlow2.estMapPoseData,...
        'RowTimes',square.ccwSlow2.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow2.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwSlow2.gnssMapPoseTimetable,...
        square.ccwSlow2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.ccwSlow2.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwSlow2.estMapPoseTimetable,...
        square.ccwSlow2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.ccwSlow2.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow2.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwSlow2.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow2.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwSlow2.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwSlow2.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwSlow2.absErrorMeanGnssMapXY =...
        sqrt(square.ccwSlow2.absErrorMeanGnssMapX^2+...
        square.ccwSlow2.absErrorMeanGnssMapY^2);
    square.ccwSlow2.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwSlow2.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwSlow2.absErrorStdGnssMapXY =...
        sqrt(square.ccwSlow2.absErrorStdGnssMapX^2+...
        square.ccwSlow2.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwSlow2.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow2.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow2.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow2.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwSlow2.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwSlow2.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwSlow2.absErrorMeanEstMapXY =...
        sqrt(square.ccwSlow2.absErrorMeanEstMapX^2+...
        square.ccwSlow2.absErrorMeanEstMapY^2);
    square.ccwSlow2.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwSlow2.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwSlow2.absErrorStdEstMapXY =...
        sqrt(square.ccwSlow2.absErrorStdEstMapX^2+...
        square.ccwSlow2.absErrorStdEstMapY^2);
    square.ccwSlow2.gnssToEstMeanErrorChange =...
        (square.ccwSlow2.absErrorMeanEstMapXY-...
        square.ccwSlow2.absErrorMeanGnssMapXY)/...
        square.ccwSlow2.absErrorMeanGnssMapXY;
    square.ccwSlow2.gnssToEstStdErrorChange =...
        (square.ccwSlow2.absErrorStdEstMapXY-...
        square.ccwSlow2.absErrorStdGnssMapXY)/...
        square.ccwSlow2.absErrorStdGnssMapXY;

    % CCW Slow 3 pose accuracy
    square.ccwSlow3.groundTruthTimetable =...
        array2timetable(square.ccwSlow3.groundTruthData,...
        'RowTimes',square.ccwSlow3.groundTruthDatetime);        % create timetable for
groundTruth
    square.ccwSlow3.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow3.gnssMapPoseData,...
        'RowTimes',square.ccwSlow3.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow3.estMapPoseTimetable =...
        array2timetable(square.ccwSlow3.estMapPoseData,...
        'RowTimes',square.ccwSlow3.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow3.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwSlow3.gnssMapPoseTimetable,...
        square.ccwSlow3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.ccwSlow3.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwSlow3.estMapPoseTimetable,...
        square.ccwSlow3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.ccwSlow3.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow3.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwSlow3.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow3.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwSlow3.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwSlow3.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwSlow3.absErrorMeanGnssMapXY =...
        sqrt(square.ccwSlow3.absErrorMeanGnssMapX^2+...
        square.ccwSlow3.absErrorMeanGnssMapY^2);
    square.ccwSlow3.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwSlow3.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwSlow3.absErrorStdGnssMapXY =...
        sqrt(square.ccwSlow3.absErrorStdGnssMapX^2+...
        square.ccwSlow3.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwSlow3.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow3.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow3.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow3.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwSlow3.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwSlow3.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwSlow3.absErrorMeanEstMapXY =...
        sqrt(square.ccwSlow3.absErrorMeanEstMapX^2+...
        square.ccwSlow3.absErrorMeanEstMapY^2);
    square.ccwSlow3.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwSlow3.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwSlow3.absErrorStdEstMapXY =...
        sqrt(square.ccwSlow3.absErrorStdEstMapX^2+...
        square.ccwSlow3.absErrorStdEstMapY^2);
    square.ccwSlow3.gnssToEstMeanErrorChange =...
        (square.ccwSlow3.absErrorMeanEstMapXY-...
        square.ccwSlow3.absErrorMeanGnssMapXY)/...
        square.ccwSlow3.absErrorMeanGnssMapXY;
    square.ccwSlow3.gnssToEstStdErrorChange =...
        (square.ccwSlow3.absErrorStdEstMapXY-...
        square.ccwSlow3.absErrorStdGnssMapXY)/...
        square.ccwSlow3.absErrorStdGnssMapXY;

    % CCW Slow 4 pose accuracy
    square.ccwSlow4.groundTruthTimetable =...
        array2timetable(square.ccwSlow4.groundTruthData,...
        'RowTimes',square.ccwSlow4.groundTruthDatetime);        % create timetable for
groundTruth
    square.ccwSlow4.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow4.gnssMapPoseData,...
        'RowTimes',square.ccwSlow4.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow4.estMapPoseTimetable =...
        array2timetable(square.ccwSlow4.estMapPoseData,...
        'RowTimes',square.ccwSlow4.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow4.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwSlow4.gnssMapPoseTimetable,...
        square.ccwSlow4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.ccwSlow4.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwSlow4.estMapPoseTimetable,...
        square.ccwSlow4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.ccwSlow4.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow4.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwSlow4.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow4.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwSlow4.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwSlow4.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwSlow4.absErrorMeanGnssMapXY =...
        sqrt(square.ccwSlow4.absErrorMeanGnssMapX^2+...
        square.ccwSlow4.absErrorMeanGnssMapY^2);
    square.ccwSlow4.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwSlow4.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwSlow4.absErrorStdGnssMapXY =...
        sqrt(square.ccwSlow4.absErrorStdGnssMapX^2+...
        square.ccwSlow4.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwSlow4.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow4.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow4.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow4.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwSlow4.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwSlow4.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwSlow4.absErrorMeanEstMapXY =...
        sqrt(square.ccwSlow4.absErrorMeanEstMapX^2+...
        square.ccwSlow4.absErrorMeanEstMapY^2);
    square.ccwSlow4.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwSlow4.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwSlow4.absErrorStdEstMapXY =...
        sqrt(square.ccwSlow4.absErrorStdEstMapX^2+...
        square.ccwSlow4.absErrorStdEstMapY^2);
    square.ccwSlow4.gnssToEstMeanErrorChange =...
        (square.ccwSlow4.absErrorMeanEstMapXY-...
        square.ccwSlow4.absErrorMeanGnssMapXY)/...
        square.ccwSlow4.absErrorMeanGnssMapXY;
    square.ccwSlow4.gnssToEstStdErrorChange =...
        (square.ccwSlow4.absErrorStdEstMapXY-...
        square.ccwSlow4.absErrorStdGnssMapXY)/...
        square.ccwSlow4.absErrorStdGnssMapXY;

    % CCW Slow 5 pose accuracy
    square.ccwSlow5.groundTruthTimetable =...
        array2timetable(square.ccwSlow5.groundTruthData,...
        'RowTimes',square.ccwSlow5.groundTruthDatetime);        % create timetable for 
groundTruth
    square.ccwSlow5.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow5.gnssMapPoseData,...
        'RowTimes',square.ccwSlow5.gnssMapPoseDatetime);        % create timetable for 
gnssMapPose
    square.ccwSlow5.estMapPoseTimetable =...
        array2timetable(square.ccwSlow5.estMapPoseData,...
        'RowTimes',square.ccwSlow5.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow5.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwSlow5.gnssMapPoseTimetable,...
        square.ccwSlow5.groundTruthTimetable,'first','linear'); % synchronize groundTruth and 
gnssMapPose
    square.ccwSlow5.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwSlow5.estMapPoseTimetable,...
        square.ccwSlow5.groundTruthTimetable,'first','linear'); % synchronize groundTruth and 
estMapPose
    syncGnssX =...
        square.ccwSlow5.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow5.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwSlow5.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow5.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwSlow5.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwSlow5.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwSlow5.absErrorMeanGnssMapXY =...
        sqrt(square.ccwSlow5.absErrorMeanGnssMapX^2+...
        square.ccwSlow5.absErrorMeanGnssMapY^2);
    square.ccwSlow5.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwSlow5.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwSlow5.absErrorStdGnssMapXY =...
        sqrt(square.ccwSlow5.absErrorStdGnssMapX^2+...
        square.ccwSlow5.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwSlow5.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow5.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow5.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow5.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwSlow5.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwSlow5.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwSlow5.absErrorMeanEstMapXY =...
        sqrt(square.ccwSlow5.absErrorMeanEstMapX^2+...
        square.ccwSlow5.absErrorMeanEstMapY^2);
    square.ccwSlow5.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwSlow5.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwSlow5.absErrorStdEstMapXY =...
        sqrt(square.ccwSlow5.absErrorStdEstMapX^2+...
        square.ccwSlow5.absErrorStdEstMapY^2);
    square.ccwSlow5.gnssToEstMeanErrorChange =...
        (square.ccwSlow5.absErrorMeanEstMapXY-...
        square.ccwSlow5.absErrorMeanGnssMapXY)/...
        square.ccwSlow5.absErrorMeanGnssMapXY;
    square.ccwSlow5.gnssToEstStdErrorChange =...
        (square.ccwSlow5.absErrorStdEstMapXY-...
        square.ccwSlow5.absErrorStdGnssMapXY)/...
        square.ccwSlow5.absErrorStdGnssMapXY;

    % CCW Slow 6 pose accuracy
    square.ccwSlow6.groundTruthTimetable =...
        array2timetable(square.ccwSlow6.groundTruthData,...
        'RowTimes',square.ccwSlow6.groundTruthDatetime);        % create timetable for
groundTruth
    square.ccwSlow6.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow6.gnssMapPoseData,...
        'RowTimes',square.ccwSlow6.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow6.estMapPoseTimetable =...
        array2timetable(square.ccwSlow6.estMapPoseData,...
        'RowTimes',square.ccwSlow6.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow6.groundTruthGnssMapErrorTimetable =...
        synchronize(square.ccwSlow6.gnssMapPoseTimetable,...
        square.ccwSlow6.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.ccwSlow6.groundTruthEstMapErrorTimetable =...
        synchronize(square.ccwSlow6.estMapPoseTimetable,...
        square.ccwSlow6.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.ccwSlow6.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow6.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.ccwSlow6.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow6.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.ccwSlow6.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.ccwSlow6.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.ccwSlow6.absErrorMeanGnssMapXY =...
        sqrt(square.ccwSlow6.absErrorMeanGnssMapX^2+...
        square.ccwSlow6.absErrorMeanGnssMapY^2);
    square.ccwSlow6.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.ccwSlow6.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.ccwSlow6.absErrorStdGnssMapXY =...
        sqrt(square.ccwSlow6.absErrorStdGnssMapX^2+...
        square.ccwSlow6.absErrorStdGnssMapY^2);
    syncEstX =...
        square.ccwSlow6.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.ccwSlow6.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow6.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.ccwSlow6.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.ccwSlow6.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.ccwSlow6.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.ccwSlow6.absErrorMeanEstMapXY =...
        sqrt(square.ccwSlow6.absErrorMeanEstMapX^2+...
        square.ccwSlow6.absErrorMeanEstMapY^2);
    square.ccwSlow6.absErrorStdEstMapX = std(errorVecEstMapX);
    square.ccwSlow6.absErrorStdEstMapY = std(errorVecEstMapY);
    square.ccwSlow6.absErrorStdEstMapXY =...
        sqrt(square.ccwSlow6.absErrorStdEstMapX^2+...
        square.ccwSlow6.absErrorStdEstMapY^2);
    square.ccwSlow6.gnssToEstMeanErrorChange =...
        (square.ccwSlow6.absErrorMeanEstMapXY-...
        square.ccwSlow6.absErrorMeanGnssMapXY)/...
        square.ccwSlow6.absErrorMeanGnssMapXY;
    square.ccwSlow6.gnssToEstStdErrorChange =...
        (square.ccwSlow6.absErrorStdEstMapXY-...
        square.ccwSlow6.absErrorStdGnssMapXY)/...
        square.ccwSlow6.absErrorStdGnssMapXY;

    % CW Fast 1 pose accuracy
    square.cwFast1.groundTruthTimetable =...
        array2timetable(square.cwFast1.groundTruthData,...
        'RowTimes',square.cwFast1.groundTruthDatetime);        % create timetable for groundTruth
    square.cwFast1.gnssMapPoseTimetable =...
        array2timetable(square.cwFast1.gnssMapPoseData,...
        'RowTimes',square.cwFast1.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwFast1.estMapPoseTimetable =...
        array2timetable(square.cwFast1.estMapPoseData,...
        'RowTimes',square.cwFast1.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwFast1.groundTruthGnssMapErrorTimetable =...
        synchronize(square.cwFast1.gnssMapPoseTimetable,...
        square.cwFast1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.cwFast1.groundTruthEstMapErrorTimetable =...
        synchronize(square.cwFast1.estMapPoseTimetable,...
        square.cwFast1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.cwFast1.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwFast1.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.cwFast1.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwFast1.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.cwFast1.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.cwFast1.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.cwFast1.absErrorMeanGnssMapXY =...
        sqrt(square.cwFast1.absErrorMeanGnssMapX^2+...
        square.cwFast1.absErrorMeanGnssMapY^2);
    square.cwFast1.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.cwFast1.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.cwFast1.absErrorStdGnssMapXY =...
        sqrt(square.cwFast1.absErrorStdGnssMapX^2+...
        square.cwFast1.absErrorStdGnssMapY^2);
    syncEstX =...
        square.cwFast1.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwFast1.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.cwFast1.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwFast1.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.cwFast1.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.cwFast1.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.cwFast1.absErrorMeanEstMapXY =...
        sqrt(square.cwFast1.absErrorMeanEstMapX^2+...
        square.cwFast1.absErrorMeanEstMapY^2);
    square.cwFast1.absErrorStdEstMapX = std(errorVecEstMapX);
    square.cwFast1.absErrorStdEstMapY = std(errorVecEstMapY);
    square.cwFast1.absErrorStdEstMapXY =...
        sqrt(square.cwFast1.absErrorStdEstMapX^2+...
        square.cwFast1.absErrorStdEstMapY^2);
    square.cwFast1.gnssToEstMeanErrorChange =...
        (square.cwFast1.absErrorMeanEstMapXY-...
        square.cwFast1.absErrorMeanGnssMapXY)/...
        square.cwFast1.absErrorMeanGnssMapXY;
    square.cwFast1.gnssToEstStdErrorChange =...
        (square.cwFast1.absErrorStdEstMapXY-...
        square.cwFast1.absErrorStdGnssMapXY)/...
        square.cwFast1.absErrorStdGnssMapXY;

    % CW Fast 2 pose accuracy
    square.cwFast2.groundTruthTimetable =...
        array2timetable(square.cwFast2.groundTruthData,...
        'RowTimes',square.cwFast2.groundTruthDatetime);        % create timetable for groundTruth
    square.cwFast2.gnssMapPoseTimetable =...
        array2timetable(square.cwFast2.gnssMapPoseData,...
        'RowTimes',square.cwFast2.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwFast2.estMapPoseTimetable =...
        array2timetable(square.cwFast2.estMapPoseData,...
        'RowTimes',square.cwFast2.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwFast2.groundTruthGnssMapErrorTimetable =...
        synchronize(square.cwFast2.gnssMapPoseTimetable,...
        square.cwFast2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.cwFast2.groundTruthEstMapErrorTimetable =...
        synchronize(square.cwFast2.estMapPoseTimetable,...
        square.cwFast2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.cwFast2.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwFast2.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.cwFast2.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwFast2.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.cwFast2.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.cwFast2.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.cwFast2.absErrorMeanGnssMapXY =...
        sqrt(square.cwFast2.absErrorMeanGnssMapX^2+...
        square.cwFast2.absErrorMeanGnssMapY^2);
    square.cwFast2.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.cwFast2.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.cwFast2.absErrorStdGnssMapXY =...
        sqrt(square.cwFast2.absErrorStdGnssMapX^2+...
        square.cwFast2.absErrorStdGnssMapY^2);
    syncEstX =...
        square.cwFast2.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwFast2.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.cwFast2.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwFast2.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.cwFast2.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.cwFast2.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.cwFast2.absErrorMeanEstMapXY =...
        sqrt(square.cwFast2.absErrorMeanEstMapX^2+...
        square.cwFast2.absErrorMeanEstMapY^2);
    square.cwFast2.absErrorStdEstMapX = std(errorVecEstMapX);
    square.cwFast2.absErrorStdEstMapY = std(errorVecEstMapY);
    square.cwFast2.absErrorStdEstMapXY =...
```

```
        sqrt(square.cwFast2.absErrorStdEstMapX^2+...
        square.cwFast2.absErrorStdEstMapY^2);
    square.cwFast2.gnssToEstMeanErrorChange =...
        (square.cwFast2.absErrorMeanEstMapXY-...
        square.cwFast2.absErrorMeanGnssMapXY)/...
        square.cwFast2.absErrorMeanGnssMapXY;
    square.cwFast2.gnssToEstStdErrorChange =...
        (square.cwFast2.absErrorStdEstMapXY-...
        square.cwFast2.absErrorStdGnssMapXY)/...
        square.cwFast2.absErrorStdGnssMapXY;

    % CW Fast 3 pose accuracy
    square.cwFast3.groundTruthTimetable =...
        array2timetable(square.cwFast3.groundTruthData,...
        'RowTimes',square.cwFast3.groundTruthDatetime);        % create timetable for groundTruth
    square.cwFast3.gnssMapPoseTimetable =...
        array2timetable(square.cwFast3.gnssMapPoseData,...
        'RowTimes',square.cwFast3.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwFast3.estMapPoseTimetable =...
        array2timetable(square.cwFast3.estMapPoseData,...
        'RowTimes',square.cwFast3.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwFast3.groundTruthGnssMapErrorTimetable =...
        synchronize(square.cwFast3.gnssMapPoseTimetable,...
        square.cwFast3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.cwFast3.groundTruthEstMapErrorTimetable =...
        synchronize(square.cwFast3.estMapPoseTimetable,...
        square.cwFast3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.cwFast3.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwFast3.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.cwFast3.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwFast3.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.cwFast3.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.cwFast3.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.cwFast3.absErrorMeanGnssMapXY =...
        sqrt(square.cwFast3.absErrorMeanGnssMapX^2+...
        square.cwFast3.absErrorMeanGnssMapY^2);
    square.cwFast3.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.cwFast3.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.cwFast3.absErrorStdGnssMapXY =...
        sqrt(square.cwFast3.absErrorStdGnssMapX^2+...
        square.cwFast3.absErrorStdGnssMapY^2);
    syncEstX =...
        square.cwFast3.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwFast3.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.cwFast3.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwFast3.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.cwFast3.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.cwFast3.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.cwFast3.absErrorMeanEstMapXY =...
        sqrt(square.cwFast3.absErrorMeanEstMapX^2+...
        square.cwFast3.absErrorMeanEstMapY^2);
    square.cwFast3.absErrorStdEstMapX = std(errorVecEstMapX);
    square.cwFast3.absErrorStdEstMapY = std(errorVecEstMapY);
    square.cwFast3.absErrorStdEstMapXY =...
        sqrt(square.cwFast3.absErrorStdEstMapX^2+...
        square.cwFast3.absErrorStdEstMapY^2);
    square.cwFast3.gnssToEstMeanErrorChange =...
```

```
        (square.cwFast3.absErrorMeanEstMapXY-...
        square.cwFast3.absErrorMeanGnssMapXY)/...
        square.cwFast3.absErrorMeanGnssMapXY;
    square.cwFast3.gnssToEstStdErrorChange =...
        (square.cwFast3.absErrorStdEstMapXY-...
        square.cwFast3.absErrorStdGnssMapXY)/...
        square.cwFast3.absErrorStdGnssMapXY;

    % CW Fast 4 pose accuracy
    square.cwFast4.groundTruthTimetable =...
        array2timetable(square.cwFast4.groundTruthData,...
        'RowTimes',square.cwFast4.groundTruthDatetime);        % create timetable for groundTruth
    square.cwFast4.gnssMapPoseTimetable =...
        array2timetable(square.cwFast4.gnssMapPoseData,...
        'RowTimes',square.cwFast4.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwFast4.estMapPoseTimetable =...
        array2timetable(square.cwFast4.estMapPoseData,...
        'RowTimes',square.cwFast4.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwFast4.groundTruthGnssMapErrorTimetable =...
        synchronize(square.cwFast4.gnssMapPoseTimetable,...
        square.cwFast4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.cwFast4.groundTruthEstMapErrorTimetable =...
        synchronize(square.cwFast4.estMapPoseTimetable,...
        square.cwFast4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.cwFast4.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwFast4.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.cwFast4.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwFast4.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.cwFast4.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.cwFast4.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.cwFast4.absErrorMeanGnssMapXY =...
        sqrt(square.cwFast4.absErrorMeanGnssMapX^2+...
        square.cwFast4.absErrorMeanGnssMapY^2);
    square.cwFast4.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.cwFast4.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.cwFast4.absErrorStdGnssMapXY =...
        sqrt(square.cwFast4.absErrorStdGnssMapX^2+...
        square.cwFast4.absErrorStdGnssMapY^2);
    syncEstX =...
        square.cwFast4.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwFast4.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.cwFast4.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwFast4.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.cwFast4.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.cwFast4.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.cwFast4.absErrorMeanEstMapXY =...
        sqrt(square.cwFast4.absErrorMeanEstMapX^2+...
        square.cwFast4.absErrorMeanEstMapY^2);
    square.cwFast4.absErrorStdEstMapX = std(errorVecEstMapX);
    square.cwFast4.absErrorStdEstMapY = std(errorVecEstMapY);
    square.cwFast4.absErrorStdEstMapXY =...
        sqrt(square.cwFast4.absErrorStdEstMapX^2+...
        square.cwFast4.absErrorStdEstMapY^2);
    square.cwFast4.gnssToEstMeanErrorChange =...
        (square.cwFast4.absErrorMeanEstMapXY-...
        square.cwFast4.absErrorMeanGnssMapXY)/...
        square.cwFast4.absErrorMeanGnssMapXY;
    square.cwFast4.gnssToEstStdErrorChange =...
        (square.cwFast4.absErrorStdEstMapXY-...
        square.cwFast4.absErrorStdGnssMapXY)/...
        square.cwFast4.absErrorStdGnssMapXY;

    % CW Slow 1 pose accuracy
    square.cwSlow1.groundTruthTimetable =...
        array2timetable(square.cwSlow1.groundTruthData,...
        'RowTimes',square.cwSlow1.groundTruthDatetime);         % create timetable for groundTruth
    square.cwSlow1.gnssMapPoseTimetable =...
        array2timetable(square.cwSlow1.gnssMapPoseData,...
        'RowTimes',square.cwSlow1.gnssMapPoseDatetime);         % create timetable for gnssMapPose
    square.cwSlow1.estMapPoseTimetable =...
        array2timetable(square.cwSlow1.estMapPoseData,...
        'RowTimes',square.cwSlow1.estMapPoseDatetime);          % create timetable for estMapPose
    square.cwSlow1.groundTruthGnssMapErrorTimetable =...
        synchronize(square.cwSlow1.gnssMapPoseTimetable,...
        square.cwSlow1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.cwSlow1.groundTruthEstMapErrorTimetable =...
        synchronize(square.cwSlow1.estMapPoseTimetable,...
        square.cwSlow1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.cwSlow1.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwSlow1.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.cwSlow1.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwSlow1.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.cwSlow1.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.cwSlow1.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.cwSlow1.absErrorMeanGnssMapXY =...
        sqrt(square.cwSlow1.absErrorMeanGnssMapX^2+...
        square.cwSlow1.absErrorMeanGnssMapY^2);
    square.cwSlow1.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.cwSlow1.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.cwSlow1.absErrorStdGnssMapXY =...
        sqrt(square.cwSlow1.absErrorStdGnssMapX^2+...
        square.cwSlow1.absErrorStdGnssMapY^2);
    syncEstX =...
        square.cwSlow1.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwSlow1.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.cwSlow1.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwSlow1.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.cwSlow1.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.cwSlow1.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.cwSlow1.absErrorMeanEstMapXY =...
        sqrt(square.cwSlow1.absErrorMeanEstMapX^2+...
        square.cwSlow1.absErrorMeanEstMapY^2);
    square.cwSlow1.absErrorStdEstMapX = std(errorVecEstMapX);
    square.cwSlow1.absErrorStdEstMapY = std(errorVecEstMapY);
    square.cwSlow1.absErrorStdEstMapXY =...
        sqrt(square.cwSlow1.absErrorStdEstMapX^2+...
        square.cwSlow1.absErrorStdEstMapY^2);
    square.cwSlow1.gnssToEstMeanErrorChange =...
        (square.cwSlow1.absErrorMeanEstMapXY-...
        square.cwSlow1.absErrorMeanGnssMapXY)/...
        square.cwSlow1.absErrorMeanGnssMapXY;
    square.cwSlow1.gnssToEstStdErrorChange =...
        (square.cwSlow1.absErrorStdEstMapXY-...
        square.cwSlow1.absErrorStdGnssMapXY)/...
        square.cwSlow1.absErrorStdGnssMapXY;

    % CW Slow 2 pose accuracy
    square.cwSlow2.groundTruthTimetable =...
        array2timetable(square.cwSlow2.groundTruthData,...
        'RowTimes',square.cwSlow2.groundTruthDatetime);        % create timetable for groundTruth
    square.cwSlow2.gnssMapPoseTimetable =...
        array2timetable(square.cwSlow2.gnssMapPoseData,...
        'RowTimes',square.cwSlow2.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwSlow2.estMapPoseTimetable =...
        array2timetable(square.cwSlow2.estMapPoseData,...
        'RowTimes',square.cwSlow2.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwSlow2.groundTruthGnssMapErrorTimetable =...
        synchronize(square.cwSlow2.gnssMapPoseTimetable,...
        square.cwSlow2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.cwSlow2.groundTruthEstMapErrorTimetable =...
        synchronize(square.cwSlow2.estMapPoseTimetable,...
        square.cwSlow2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.cwSlow2.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwSlow2.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.cwSlow2.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwSlow2.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.cwSlow2.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.cwSlow2.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.cwSlow2.absErrorMeanGnssMapXY =...
        sqrt(square.cwSlow2.absErrorMeanGnssMapX^2+...
        square.cwSlow2.absErrorMeanGnssMapY^2);
    square.cwSlow2.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.cwSlow2.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.cwSlow2.absErrorStdGnssMapXY =...
        sqrt(square.cwSlow2.absErrorStdGnssMapX^2+...
        square.cwSlow2.absErrorStdGnssMapY^2);
    syncEstX =...
        square.cwSlow2.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwSlow2.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.cwSlow2.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwSlow2.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.cwSlow2.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.cwSlow2.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.cwSlow2.absErrorMeanEstMapXY =...
        sqrt(square.cwSlow2.absErrorMeanEstMapX^2+...
        square.cwSlow2.absErrorMeanEstMapY^2);
    square.cwSlow2.absErrorStdEstMapX = std(errorVecEstMapX);
    square.cwSlow2.absErrorStdEstMapY = std(errorVecEstMapY);
    square.cwSlow2.absErrorStdEstMapXY =...
        sqrt(square.cwSlow2.absErrorStdEstMapX^2+...
        square.cwSlow2.absErrorStdEstMapY^2);
    square.cwSlow2.gnssToEstMeanErrorChange =...
        (square.cwSlow2.absErrorMeanEstMapXY-...
        square.cwSlow2.absErrorMeanGnssMapXY)/...
        square.cwSlow2.absErrorMeanGnssMapXY;
    square.cwSlow2.gnssToEstStdErrorChange =...
        (square.cwSlow2.absErrorStdEstMapXY-...
        square.cwSlow2.absErrorStdGnssMapXY)/...
        square.cwSlow2.absErrorStdGnssMapXY;

    % CW Slow 3 pose accuracy
    square.cwSlow3.groundTruthTimetable =...
        array2timetable(square.cwSlow3.groundTruthData,...
        'RowTimes',square.cwSlow3.groundTruthDatetime);        % create timetable for groundTruth
    square.cwSlow3.gnssMapPoseTimetable =...
        array2timetable(square.cwSlow3.gnssMapPoseData,...
        'RowTimes',square.cwSlow3.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwSlow3.estMapPoseTimetable =...
        array2timetable(square.cwSlow3.estMapPoseData,...
        'RowTimes',square.cwSlow3.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwSlow3.groundTruthGnssMapErrorTimetable =...
        synchronize(square.cwSlow3.gnssMapPoseTimetable,...
        square.cwSlow3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.cwSlow3.groundTruthEstMapErrorTimetable =...
        synchronize(square.cwSlow3.estMapPoseTimetable,...
        square.cwSlow3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.cwSlow3.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwSlow3.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.cwSlow3.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwSlow3.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.cwSlow3.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.cwSlow3.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.cwSlow3.absErrorMeanGnssMapXY =...
        sqrt(square.cwSlow3.absErrorMeanGnssMapX^2+...
        square.cwSlow3.absErrorMeanGnssMapY^2);
    square.cwSlow3.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.cwSlow3.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.cwSlow3.absErrorStdGnssMapXY =...
        sqrt(square.cwSlow3.absErrorStdGnssMapX^2+...
        square.cwSlow3.absErrorStdGnssMapY^2);
    syncEstX =...
        square.cwSlow3.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwSlow3.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.cwSlow3.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwSlow3.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.cwSlow3.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.cwSlow3.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.cwSlow3.absErrorMeanEstMapXY =...
        sqrt(square.cwSlow3.absErrorMeanEstMapX^2+...
        square.cwSlow3.absErrorMeanEstMapY^2);
    square.cwSlow3.absErrorStdEstMapX = std(errorVecEstMapX);
    square.cwSlow3.absErrorStdEstMapY = std(errorVecEstMapY);
    square.cwSlow3.absErrorStdEstMapXY =...
        sqrt(square.cwSlow3.absErrorStdEstMapX^2+...
        square.cwSlow3.absErrorStdEstMapY^2);
    square.cwSlow3.gnssToEstMeanErrorChange =...
        (square.cwSlow3.absErrorMeanEstMapXY-...
        square.cwSlow3.absErrorMeanGnssMapXY)/...
        square.cwSlow3.absErrorMeanGnssMapXY;
    square.cwSlow3.gnssToEstStdErrorChange =...
        (square.cwSlow3.absErrorStdEstMapXY-...
        square.cwSlow3.absErrorStdGnssMapXY)/...
        square.cwSlow3.absErrorStdGnssMapXY;

    % CW Slow 4 pose accuracy
    square.cwSlow4.groundTruthTimetable =...
        array2timetable(square.cwSlow4.groundTruthData,...
        'RowTimes',square.cwSlow4.groundTruthDatetime);        % create timetable for groundTruth
    square.cwSlow4.gnssMapPoseTimetable =...
        array2timetable(square.cwSlow4.gnssMapPoseData,...
        'RowTimes',square.cwSlow4.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwSlow4.estMapPoseTimetable =...
        array2timetable(square.cwSlow4.estMapPoseData,...
        'RowTimes',square.cwSlow4.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwSlow4.groundTruthGnssMapErrorTimetable =...
        synchronize(square.cwSlow4.gnssMapPoseTimetable,...
        square.cwSlow4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    square.cwSlow4.groundTruthEstMapErrorTimetable =...
        synchronize(square.cwSlow4.estMapPoseTimetable,...
        square.cwSlow4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        square.cwSlow4.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwSlow4.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        square.cwSlow4.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwSlow4.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    square.cwSlow4.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    square.cwSlow4.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    square.cwSlow4.absErrorMeanGnssMapXY =...
        sqrt(square.cwSlow4.absErrorMeanGnssMapX^2+...
        square.cwSlow4.absErrorMeanGnssMapY^2);
    square.cwSlow4.absErrorStdGnssMapX = std(errorVecGnssMapX);
    square.cwSlow4.absErrorStdGnssMapY = std(errorVecGnssMapY);
    square.cwSlow4.absErrorStdGnssMapXY =...
        sqrt(square.cwSlow4.absErrorStdGnssMapX^2+...
        square.cwSlow4.absErrorStdGnssMapY^2);
    syncEstX =...
        square.cwSlow4.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        square.cwSlow4.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        square.cwSlow4.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        square.cwSlow4.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    square.cwSlow4.absErrorMeanEstMapX = mean(errorVecEstMapX);
    square.cwSlow4.absErrorMeanEstMapY = mean(errorVecEstMapY);
    square.cwSlow4.absErrorMeanEstMapXY =...
        sqrt(square.cwSlow4.absErrorMeanEstMapX^2+...
        square.cwSlow4.absErrorMeanEstMapY^2);
    square.cwSlow4.absErrorStdEstMapX = std(errorVecEstMapX);
    square.cwSlow4.absErrorStdEstMapY = std(errorVecEstMapY);
    square.cwSlow4.absErrorStdEstMapXY =...
        sqrt(square.cwSlow4.absErrorStdEstMapX^2+...
        square.cwSlow4.absErrorStdEstMapY^2);
    square.cwSlow4.gnssToEstMeanErrorChange =...
        (square.cwSlow4.absErrorMeanEstMapXY-...
        square.cwSlow4.absErrorMeanGnssMapXY)/...
        square.cwSlow4.absErrorMeanGnssMapXY;
    square.cwSlow4.gnssToEstStdErrorChange =...
        (square.cwSlow4.absErrorStdEstMapXY-...
        square.cwSlow4.absErrorStdGnssMapXY)/...
        square.cwSlow4.absErrorStdGnssMapXY;

end

function square = calculateInterpolationError(square)

    % CCW Fast 1 interpolation accuracy
    square.ccwFast1.gnssMapPoseTimetable =...
        array2timetable(square.ccwFast1.gnssMapPoseData,...
        'RowTimes',square.ccwFast1.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwFast1.estMapPoseTimetable =...
        array2timetable(square.ccwFast1.estMapPoseData,...
        'RowTimes',square.ccwFast1.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwFast1.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwFast1.estMapPoseTimetable,...
        square.ccwFast1.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwFast1.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwFast1.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwFast1.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwFast1.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwFast1.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwFast1.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwFast1.interpErrorStdX = std(errorVecEstMapX);
    square.ccwFast1.interpErrorStdY = std(errorVecEstMapY);
    square.ccwFast1.interpErrorMeanXY =...
        sqrt(square.ccwFast1.interpErrorMeanX^2+...
        square.ccwFast1.interpErrorMeanY^2);
    square.ccwFast1.interpErrorStdXY =...
        sqrt(square.ccwFast1.interpErrorStdX^2+...
        square.ccwFast1.interpErrorStdY^2);


    % CCW Fast 2 interpolation accuracy
    square.ccwFast2.gnssMapPoseTimetable =...
        array2timetable(square.ccwFast2.gnssMapPoseData,...
        'RowTimes',square.ccwFast2.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwFast2.estMapPoseTimetable =...
        array2timetable(square.ccwFast2.estMapPoseData,...
        'RowTimes',square.ccwFast2.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwFast2.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwFast2.estMapPoseTimetable,...
        square.ccwFast2.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwFast2.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwFast2.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwFast2.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwFast2.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwFast2.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwFast2.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwFast2.interpErrorStdX = std(errorVecEstMapX);
    square.ccwFast2.interpErrorStdY = std(errorVecEstMapY);
    square.ccwFast2.interpErrorMeanXY =...
        sqrt(square.ccwFast2.interpErrorMeanX^2+...
        square.ccwFast2.interpErrorMeanY^2);
    square.ccwFast2.interpErrorStdXY =...
        sqrt(square.ccwFast2.interpErrorStdX^2+...
        square.ccwFast2.interpErrorStdY^2);

    % CCW Fast 3 pose accuracy
    square.ccwFast3.gnssMapPoseTimetable =...
        array2timetable(square.ccwFast3.gnssMapPoseData,...
        'RowTimes',square.ccwFast3.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwFast3.estMapPoseTimetable =...
```

```
        array2timetable(square.ccwFast3.estMapPoseData,...
        'RowTimes',square.ccwFast3.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwFast3.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwFast3.estMapPoseTimetable,...
        square.ccwFast3.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwFast3.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwFast3.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwFast3.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwFast3.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwFast3.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwFast3.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwFast3.interpErrorStdX = std(errorVecEstMapX);
    square.ccwFast3.interpErrorStdY = std(errorVecEstMapY);
    square.ccwFast3.interpErrorMeanXY =...
        sqrt(square.ccwFast3.interpErrorMeanX^2+...
        square.ccwFast3.interpErrorMeanY^2);
    square.ccwFast3.interpErrorStdXY =...
        sqrt(square.ccwFast3.interpErrorStdX^2+...
        square.ccwFast3.interpErrorStdY^2);

    % CCW Fast 4 pose accuracy
    square.ccwFast4.gnssMapPoseTimetable =...
        array2timetable(square.ccwFast4.gnssMapPoseData,...
        'RowTimes',square.ccwFast4.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwFast4.estMapPoseTimetable =...
        array2timetable(square.ccwFast4.estMapPoseData,...
        'RowTimes',square.ccwFast4.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwFast4.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwFast4.estMapPoseTimetable,...
        square.ccwFast4.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwFast4.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwFast4.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwFast4.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwFast4.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwFast4.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwFast4.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwFast4.interpErrorStdX = std(errorVecEstMapX);
    square.ccwFast4.interpErrorStdY = std(errorVecEstMapY);
    square.ccwFast4.interpErrorMeanXY =...
        sqrt(square.ccwFast4.interpErrorMeanX^2+...
        square.ccwFast4.interpErrorMeanY^2);
    square.ccwFast4.interpErrorStdXY =...
        sqrt(square.ccwFast4.interpErrorStdX^2+...
        square.ccwFast4.interpErrorStdY^2);

    % CCW Slow 1 interpolation accuracy
    square.ccwSlow1.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow1.gnssMapPoseData,...
        'RowTimes',square.ccwSlow1.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow1.estMapPoseTimetable =...
        array2timetable(square.ccwSlow1.estMapPoseData,...
        'RowTimes',square.ccwSlow1.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow1.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwSlow1.estMapPoseTimetable,...
        square.ccwSlow1.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwSlow1.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwSlow1.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow1.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwSlow1.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwSlow1.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwSlow1.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwSlow1.interpErrorStdX = std(errorVecEstMapX);
    square.ccwSlow1.interpErrorStdY = std(errorVecEstMapY);
    square.ccwSlow1.interpErrorMeanXY =...
        sqrt(square.ccwSlow1.interpErrorMeanX^2+...
        square.ccwSlow1.interpErrorMeanY^2);
    square.ccwSlow1.interpErrorStdXY =...
        sqrt(square.ccwSlow1.interpErrorStdX^2+...
        square.ccwSlow1.interpErrorStdY^2);

    % CCW Slow 2 interpolation accuracy
    square.ccwSlow2.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow2.gnssMapPoseData,...
        'RowTimes',square.ccwSlow2.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow2.estMapPoseTimetable =...
        array2timetable(square.ccwSlow2.estMapPoseData,...
        'RowTimes',square.ccwSlow2.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow2.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwSlow2.estMapPoseTimetable,...
        square.ccwSlow2.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwSlow2.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwSlow2.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow2.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwSlow2.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwSlow2.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwSlow2.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwSlow2.interpErrorStdX = std(errorVecEstMapX);
    square.ccwSlow2.interpErrorStdY = std(errorVecEstMapY);
    square.ccwSlow2.interpErrorMeanXY =...
        sqrt(square.ccwSlow2.interpErrorMeanX^2+...
        square.ccwSlow2.interpErrorMeanY^2);
    square.ccwSlow2.interpErrorStdXY =...
        sqrt(square.ccwSlow2.interpErrorStdX^2+...
        square.ccwSlow2.interpErrorStdY^2);

    % CCW Slow 3 pose accuracy
    square.ccwSlow3.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow3.gnssMapPoseData,...
        'RowTimes',square.ccwSlow3.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow3.estMapPoseTimetable =...
        array2timetable(square.ccwSlow3.estMapPoseData,...
        'RowTimes',square.ccwSlow3.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow3.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwSlow3.estMapPoseTimetable,...
        square.ccwSlow3.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwSlow3.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwSlow3.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow3.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwSlow3.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwSlow3.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwSlow3.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwSlow3.interpErrorStdX = std(errorVecEstMapX);
    square.ccwSlow3.interpErrorStdY = std(errorVecEstMapY);
    square.ccwSlow3.interpErrorMeanXY =...
        sqrt(square.ccwSlow3.interpErrorMeanX^2+...
        square.ccwSlow3.interpErrorMeanY^2);
    square.ccwSlow3.interpErrorStdXY =...
        sqrt(square.ccwSlow3.interpErrorStdX^2+...
        square.ccwSlow3.interpErrorStdY^2);

    % CCW Slow 4 pose accuracy
    square.ccwSlow4.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow4.gnssMapPoseData,...
        'RowTimes',square.ccwSlow4.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow4.estMapPoseTimetable =...
        array2timetable(square.ccwSlow4.estMapPoseData,...
        'RowTimes',square.ccwSlow4.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow4.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwSlow4.estMapPoseTimetable,...
        square.ccwSlow4.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwSlow4.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwSlow4.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow4.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwSlow4.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwSlow4.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwSlow4.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwSlow4.interpErrorStdX = std(errorVecEstMapX);
    square.ccwSlow4.interpErrorStdY = std(errorVecEstMapY);
    square.ccwSlow4.interpErrorMeanXY =...
        sqrt(square.ccwSlow4.interpErrorMeanX^2+...
        square.ccwSlow4.interpErrorMeanY^2);
    square.ccwSlow4.interpErrorStdXY =...
        sqrt(square.ccwSlow4.interpErrorStdX^2+...
        square.ccwSlow4.interpErrorStdY^2);

    % CCW Slow 5 interpolation accuracy
    square.ccwSlow5.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow5.gnssMapPoseData,...
        'RowTimes',square.ccwSlow5.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow5.estMapPoseTimetable =...
        array2timetable(square.ccwSlow5.estMapPoseData,...
        'RowTimes',square.ccwSlow5.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow5.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwSlow5.estMapPoseTimetable,...
        square.ccwSlow5.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwSlow5.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwSlow5.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow5.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwSlow5.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwSlow5.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwSlow5.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwSlow5.interpErrorStdX = std(errorVecEstMapX);
    square.ccwSlow5.interpErrorStdY = std(errorVecEstMapY);
    square.ccwSlow5.interpErrorMeanXY =...
        sqrt(square.ccwSlow5.interpErrorMeanX^2+...
        square.ccwSlow5.interpErrorMeanY^2);
    square.ccwSlow5.interpErrorStdXY =...
        sqrt(square.ccwSlow5.interpErrorStdX^2+...
        square.ccwSlow5.interpErrorStdY^2);

    % CCW Slow 6 interpolation accuracy
    square.ccwSlow6.gnssMapPoseTimetable =...
        array2timetable(square.ccwSlow6.gnssMapPoseData,...
        'RowTimes',square.ccwSlow6.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    square.ccwSlow6.estMapPoseTimetable =...
        array2timetable(square.ccwSlow6.estMapPoseData,...
        'RowTimes',square.ccwSlow6.estMapPoseDatetime);         % create timetable for estMapPose
    square.ccwSlow6.gnssEstInterpErrorTimetable =...
        synchronize(square.ccwSlow6.estMapPoseTimetable,...
        square.ccwSlow6.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.ccwSlow6.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.ccwSlow6.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.ccwSlow6.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.ccwSlow6.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.ccwSlow6.interpErrorMeanX = mean(errorVecEstMapX);
    square.ccwSlow6.interpErrorMeanY = mean(errorVecEstMapY);
    square.ccwSlow6.interpErrorStdX = std(errorVecEstMapX);
    square.ccwSlow6.interpErrorStdY = std(errorVecEstMapY);
    square.ccwSlow6.interpErrorMeanXY =...
        sqrt(square.ccwSlow6.interpErrorMeanX^2+...
        square.ccwSlow6.interpErrorMeanY^2);
    square.ccwSlow6.interpErrorStdXY =...
        sqrt(square.ccwSlow6.interpErrorStdX^2+...
        square.ccwSlow6.interpErrorStdY^2);

    % CW Fast 1 interpolation accuracy
    square.cwFast1.gnssMapPoseTimetable =...
        array2timetable(square.cwFast1.gnssMapPoseData,...
        'RowTimes',square.cwFast1.gnssMapPoseDatetime);         % create timetable for gnssMapPose
    square.cwFast1.estMapPoseTimetable =...
        array2timetable(square.cwFast1.estMapPoseData,...
        'RowTimes',square.cwFast1.estMapPoseDatetime);          % create timetable for estMapPose
    square.cwFast1.gnssEstInterpErrorTimetable =...
        synchronize(square.cwFast1.estMapPoseTimetable,...
        square.cwFast1.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.cwFast1.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.cwFast1.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.cwFast1.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.cwFast1.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.cwFast1.interpErrorMeanX = mean(errorVecEstMapX);
    square.cwFast1.interpErrorMeanY = mean(errorVecEstMapY);
    square.cwFast1.interpErrorStdX = std(errorVecEstMapX);
    square.cwFast1.interpErrorStdY = std(errorVecEstMapY);
    square.cwFast1.interpErrorMeanXY =...
        sqrt(square.cwFast1.interpErrorMeanX^2+...
        square.cwFast1.interpErrorMeanY^2);
    square.cwFast1.interpErrorStdXY =...
        sqrt(square.cwFast1.interpErrorStdX^2+...
        square.cwFast1.interpErrorStdY^2);


    % CW Fast 2 interpolation accuracy
    square.cwFast2.gnssMapPoseTimetable =...
        array2timetable(square.cwFast2.gnssMapPoseData,...
        'RowTimes',square.cwFast2.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwFast2.estMapPoseTimetable =...
        array2timetable(square.cwFast2.estMapPoseData,...
        'RowTimes',square.cwFast2.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwFast2.gnssEstInterpErrorTimetable =...
        synchronize(square.cwFast2.estMapPoseTimetable,...
        square.cwFast2.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.cwFast2.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.cwFast2.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.cwFast2.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.cwFast2.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.cwFast2.interpErrorMeanX = mean(errorVecEstMapX);
    square.cwFast2.interpErrorMeanY = mean(errorVecEstMapY);
    square.cwFast2.interpErrorStdX = std(errorVecEstMapX);
    square.cwFast2.interpErrorStdY = std(errorVecEstMapY);
    square.cwFast2.interpErrorMeanXY =...
        sqrt(square.cwFast2.interpErrorMeanX^2+...
        square.cwFast2.interpErrorMeanY^2);
    square.cwFast2.interpErrorStdXY =...
        sqrt(square.cwFast2.interpErrorStdX^2+...
        square.cwFast2.interpErrorStdY^2);

    % CW Fast 3 pose accuracy
    square.cwFast3.gnssMapPoseTimetable =...
        array2timetable(square.cwFast3.gnssMapPoseData,...
        'RowTimes',square.cwFast3.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwFast3.estMapPoseTimetable =...
        array2timetable(square.cwFast3.estMapPoseData,...
        'RowTimes',square.cwFast3.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwFast3.gnssEstInterpErrorTimetable =...
        synchronize(square.cwFast3.estMapPoseTimetable,...
        square.cwFast3.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.cwFast3.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.cwFast3.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.cwFast3.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.cwFast3.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.cwFast3.interpErrorMeanX = mean(errorVecEstMapX);
    square.cwFast3.interpErrorMeanY = mean(errorVecEstMapY);
    square.cwFast3.interpErrorStdX = std(errorVecEstMapX);
    square.cwFast3.interpErrorStdY = std(errorVecEstMapY);
    square.cwFast3.interpErrorMeanXY =...
        sqrt(square.cwFast3.interpErrorMeanX^2+...
        square.cwFast3.interpErrorMeanY^2);
    square.cwFast3.interpErrorStdXY =...
        sqrt(square.cwFast3.interpErrorStdX^2+...
        square.cwFast3.interpErrorStdY^2);

    % CW Fast 4 pose accuracy
    square.cwFast4.gnssMapPoseTimetable =...
        array2timetable(square.cwFast4.gnssMapPoseData,...
        'RowTimes',square.cwFast4.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwFast4.estMapPoseTimetable =...
        array2timetable(square.cwFast4.estMapPoseData,...
        'RowTimes',square.cwFast4.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwFast4.gnssEstInterpErrorTimetable =...
        synchronize(square.cwFast4.estMapPoseTimetable,...
        square.cwFast4.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.cwFast4.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.cwFast4.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.cwFast4.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.cwFast4.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.cwFast4.interpErrorMeanX = mean(errorVecEstMapX);
    square.cwFast4.interpErrorMeanY = mean(errorVecEstMapY);
    square.cwFast4.interpErrorStdX = std(errorVecEstMapX);
    square.cwFast4.interpErrorStdY = std(errorVecEstMapY);
    square.cwFast4.interpErrorMeanXY =...
        sqrt(square.cwFast4.interpErrorMeanX^2+...
        square.cwFast4.interpErrorMeanY^2);
    square.cwFast4.interpErrorStdXY =...
        sqrt(square.cwFast4.interpErrorStdX^2+...
        square.cwFast4.interpErrorStdY^2);

    % CW Slow 1 interpolation accuracy
    square.cwSlow1.gnssMapPoseTimetable =...
        array2timetable(square.cwSlow1.gnssMapPoseData,...
        'RowTimes',square.cwSlow1.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwSlow1.estMapPoseTimetable =...
        array2timetable(square.cwSlow1.estMapPoseData,...
        'RowTimes',square.cwSlow1.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwSlow1.gnssEstInterpErrorTimetable =...
        synchronize(square.cwSlow1.estMapPoseTimetable,...
        square.cwSlow1.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.cwSlow1.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.cwSlow1.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.cwSlow1.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.cwSlow1.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.cwSlow1.interpErrorMeanX = mean(errorVecEstMapX);
    square.cwSlow1.interpErrorMeanY = mean(errorVecEstMapY);
    square.cwSlow1.interpErrorStdX = std(errorVecEstMapX);
    square.cwSlow1.interpErrorStdY = std(errorVecEstMapY);
    square.cwSlow1.interpErrorMeanXY =...
        sqrt(square.cwSlow1.interpErrorMeanX^2+...
        square.cwSlow1.interpErrorMeanY^2);
    square.cwSlow1.interpErrorStdXY =...
        sqrt(square.cwSlow1.interpErrorStdX^2+...
        square.cwSlow1.interpErrorStdY^2);

    % CW Slow 2 interpolation accuracy
    square.cwSlow2.gnssMapPoseTimetable =...
        array2timetable(square.cwSlow2.gnssMapPoseData,...
        'RowTimes',square.cwSlow2.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwSlow2.estMapPoseTimetable =...
        array2timetable(square.cwSlow2.estMapPoseData,...
        'RowTimes',square.cwSlow2.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwSlow2.gnssEstInterpErrorTimetable =...
        synchronize(square.cwSlow2.estMapPoseTimetable,...
        square.cwSlow2.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.cwSlow2.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.cwSlow2.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.cwSlow2.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.cwSlow2.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.cwSlow2.interpErrorMeanX = mean(errorVecEstMapX);
    square.cwSlow2.interpErrorMeanY = mean(errorVecEstMapY);
    square.cwSlow2.interpErrorStdX = std(errorVecEstMapX);
    square.cwSlow2.interpErrorStdY = std(errorVecEstMapY);
    square.cwSlow2.interpErrorMeanXY =...
        sqrt(square.cwSlow2.interpErrorMeanX^2+...
        square.cwSlow2.interpErrorMeanY^2);
    square.cwSlow2.interpErrorStdXY =...
        sqrt(square.cwSlow2.interpErrorStdX^2+...
        square.cwSlow2.interpErrorStdY^2);

    % CW Slow 3 pose accuracy
    square.cwSlow3.gnssMapPoseTimetable =...
        array2timetable(square.cwSlow3.gnssMapPoseData,...
        'RowTimes',square.cwSlow3.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwSlow3.estMapPoseTimetable =...
        array2timetable(square.cwSlow3.estMapPoseData,...
        'RowTimes',square.cwSlow3.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwSlow3.gnssEstInterpErrorTimetable =...
        synchronize(square.cwSlow3.estMapPoseTimetable,...
        square.cwSlow3.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.cwSlow3.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.cwSlow3.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.cwSlow3.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.cwSlow3.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.cwSlow3.interpErrorMeanX = mean(errorVecEstMapX);
    square.cwSlow3.interpErrorMeanY = mean(errorVecEstMapY);
    square.cwSlow3.interpErrorStdX = std(errorVecEstMapX);
    square.cwSlow3.interpErrorStdY = std(errorVecEstMapY);
    square.cwSlow3.interpErrorMeanXY =...
        sqrt(square.cwSlow3.interpErrorMeanX^2+...
        square.cwSlow3.interpErrorMeanY^2);
    square.cwSlow3.interpErrorStdXY =...
        sqrt(square.cwSlow3.interpErrorStdX^2+...
        square.cwSlow3.interpErrorStdY^2);

    % CW Slow 4 pose accuracy
    square.cwSlow4.gnssMapPoseTimetable =...
        array2timetable(square.cwSlow4.gnssMapPoseData,...
        'RowTimes',square.cwSlow4.gnssMapPoseDatetime);        % create timetable for gnssMapPose
    square.cwSlow4.estMapPoseTimetable =...
        array2timetable(square.cwSlow4.estMapPoseData,...
        'RowTimes',square.cwSlow4.estMapPoseDatetime);         % create timetable for estMapPose
    square.cwSlow4.gnssEstInterpErrorTimetable =...
        synchronize(square.cwSlow4.estMapPoseTimetable,...
        square.cwSlow4.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        square.cwSlow4.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        square.cwSlow4.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        square.cwSlow4.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        square.cwSlow4.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    square.cwSlow4.interpErrorMeanX = mean(errorVecEstMapX);
    square.cwSlow4.interpErrorMeanY = mean(errorVecEstMapY);
    square.cwSlow4.interpErrorStdX = std(errorVecEstMapX);
    square.cwSlow4.interpErrorStdY = std(errorVecEstMapY);
    square.cwSlow4.interpErrorMeanXY =...
        sqrt(square.cwSlow4.interpErrorMeanX^2+...
        square.cwSlow4.interpErrorMeanY^2);
    square.cwSlow4.interpErrorStdXY =...
        sqrt(square.cwSlow4.interpErrorStdX^2+...
        square.cwSlow4.interpErrorStdY^2);

    % Save to file
    save('squareCorrectedWithRectangleAndError.mat','square');

end

function plotAllData(square)

    % Load structure from file
    load('squareCorrectedWithRectangleAndError.mat');

    % CCW Slow 4 (CCW Slow Trial 1 in paper)
    plotTrial(square.ccwSlow4,1,...

'$CCW\hspace{1mm}Rectangle\hspace{1mm}Slow\hspace{1mm}Trial\hspace{1mm}1\hspace{1mm}(\bar{v}\appr
ox1.3\hspace{1mm}[m/s])$');

    % CCW Slow 5 (CCW Slow Trial 2 in paper)
    plotTrial(square.ccwSlow5,4,...

'$CCW\hspace{1mm}Rectangle\hspace{1mm}Slow\hspace{1mm}Trial\hspace{1mm}2\hspace{1mm}(\bar{v}\appr
ox1.3\hspace{1mm}[m/s])$');

    % CCW Slow 6 (CCW Slow Trial 3 in paper)
    plotTrial(square.ccwSlow6,7,...

'$CCW\hspace{1mm}Rectangle\hspace{1mm}Slow\hspace{1mm}Trial\hspace{1mm}3\hspace{1mm}(\bar{v}\appr
ox1.3\hspace{1mm}[m/s])$');

    % CCW Fast 1 (CCW Fast Trial 1 in paper)
    plotTrial(square.ccwFast1,10,...

'$CCW\hspace{1mm}Rectangle\hspace{1mm}Fast\hspace{1mm}Trial\hspace{1mm}1\hspace{1mm}(\bar{v}\appr
ox3.1\hspace{1mm}[m/s])$');

    % CCW Fast 3 (CCW Fast Trial 2 in paper)
    plotTrial(square.ccwFast3,13,...

'$CCW\hspace{1mm}Rectangle\hspace{1mm}Fast\hspace{1mm}Trial\hspace{1mm}2\hspace{1mm}(\bar{v}\appr
ox2.8\hspace{1mm}[m/s])$');

    % CCW Fast 4 (CCW Fast Trial 3 in paper)
    plotTrial(square.ccwFast4,16,...

'$CCW\hspace{1mm}Rectangle\hspace{1mm}Fast\hspace{1mm}Trial\hspace{1mm}3\hspace{1mm}(\bar{v}\appr
ox3.1\hspace{1mm}[m/s])$');
```

```
%
    % CW Slow 1 (CW Slow Trial 1 in paper)
    plotTrial(square.cwSlow1,19,...

'$CW\hspace{1mm}Rectangle\hspace{1mm}Slow\hspace{1mm}Trial\hspace{1mm}1\hspace{1mm}(\bar{v}\appro
x1.5\hspace{1mm}[m/s])$');

    % CW Slow 2 (CW Slow Trial 2 in paper)
    plotTrial(square.cwSlow2,22,...

'$CW\hspace{1mm}Rectangle\hspace{1mm}Slow\hspace{1mm}Trial\hspace{1mm}2\hspace{1mm}(\bar{v}\appro
x1.5\hspace{1mm}[m/s])$');

    % CW Slow 4 (CW Slow Trial 3 in paper)
    plotTrial(square.cwSlow4,25,...

'$CW\hspace{1mm}Rectangle\hspace{1mm}Slow\hspace{1mm}Trial\hspace{1mm}3\hspace{1mm}(\bar{v}\appro
x1.3\hspace{1mm}[m/s])$');

    % CW Fast 1 (CW Fast Trial 1 in paper)
    plotTrial(square.cwFast1,28,...

'$CW\hspace{1mm}Rectangle\hspace{1mm}Fast\hspace{1mm}Trial\hspace{1mm}1\hspace{1mm}(\bar{v}\appro
x2.9\hspace{1mm}[m/s])$');

    % CW Fast 2 (CW Fast Trial 2 in paper)
    plotTrial(square.cwFast2,31,...

'$CW\hspace{1mm}Rectangle\hspace{1mm}Fast\hspace{1mm}Trial\hspace{1mm}2\hspace{1mm}(\bar{v}\appro
x3.1\hspace{1mm}[m/s])$');

    % CW Fast 4 (CW Fast Trial 3 in paper)
    plotTrial(square.cwFast4,34,...

'$CW\hspace{1mm}Rectangle\hspace{1mm}Fast\hspace{1mm}Trial\hspace{1mm}3\hspace{1mm}(\bar{v}\appro
x2.8\hspace{1mm}[m/s])$');

end

function plotTrial(squareTrial,figNumStart,trialTitleString)

    % Set font size
    titleFontSize = 32;
    defaultFontSize = 28;
    markerSize = 10;

    % Plot
    figure(figNumStart);
    clf;
    hold on;
    plot(squareTrial.groundTruthData(:,1),...
        squareTrial.groundTruthData(:,2),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(squareTrial.gnssMapPoseData(:,1),...
        squareTrial.gnssMapPoseData(:,2),...
        'r','LineStyle','-','Marker','o','LineWidth',1,'MarkerSize',markerSize);
    plot(squareTrial.estMapPoseData(:,1),...
        squareTrial.estMapPoseData(:,2),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    axis equal;
    grid on;
    ylabel('$y_m\hspace{1mm}[m]$','Interpreter','latex');
    xlabel('$x_m\hspace{1mm}[m]$','Interpreter','latex');
    legend('$ground\hspace{1mm}truth$',...
        '$GNSS\hspace{1mm}measurement$',...
        '$estimator\hspace{1mm}output$',...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
    title(trialTitleString,...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    set(gcf,'Position',[0 0 1920 1280]);

    figure(figNumStart+1);
    fontSizeMod = 4;
    clf;
    rx = subplot(3,3,1);
    hold on;
    plot(squareTrial.gnssRbtPoseDatetime,...
        squareTrial.gnssRbtPoseData(:,1),...
        'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/1.5);
    plot(squareTrial.estRbtPoseDatetime,...
        squareTrial.estRbtPoseData(:,1),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$x_r\hspace{1mm}[m]$','Interpreter','latex');
    xlabel('$time [s]$','Interpreter','latex');
    datetick('x','MM:ss','keeplimits','keepticks')
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    title('$Robot\hspace{1mm}Body\hspace{1mm}x_r\hspace{1mm}position$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    ry = subplot(3,3,4);
    hold on;
    plot(squareTrial.gnssRbtPoseDatetime,...
        squareTrial.gnssRbtPoseData(:,2),...
        'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/1.5);
    plot(squareTrial.estRbtPoseDatetime,...
        squareTrial.estRbtPoseData(:,2),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$y_r\hspace{1mm}[m]$','Interpreter','latex');
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    datetick('x','MM:ss','keeplimits','keepticks')
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    title('$Robot\hspace{1mm}Body\hspace{1mm}y_r\hspace{1mm}position$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    rp = subplot(3,3,7);
    hold on;
    plot(squareTrial.imuPoseDatetime,...
        squareTrial.imuPoseData(:,3),...
        'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/2);
    plot(squareTrial.estRbtPoseDatetime,...
        squareTrial.estRbtPoseData(:,3),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\psi_r\hspace{1mm}[rad]$','Interpreter','latex');
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    datetick('x','MM:ss','keeplimits','keepticks')
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    title('$Robot\hspace{1mm}Body\hspace{1mm}\psi_r\hspace{1mm}position$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    rdx = subplot(3,3,2);
    hold on;
    plot(squareTrial.estRbtTwistDatetime,...
        squareTrial.estRbtTwistData(:,2),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    plot(squareTrial.linVelAbsDatetime,...
        squareTrial.linVelAbsData(:,3),...
        'c','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{x}\hspace{1mm}[m/s]$','Interpreter','latex');
    xlabel('$time [s]$','Interpreter','latex');
```

```
datetick('x','MM:ss','keeplimits','keepticks')
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{x}_r\hspace{1mm}velocity$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rdy = subplot(3,3,5);
hold on;
plot(squareTrial.estRbtTwistDatetime,...
    squareTrial.estRbtTwistData(:,1),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
plot(squareTrial.linVelAbsDatetime,...
    squareTrial.linVelAbsData(:,3),...
    'c','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\dot{y}\hspace{1mm}[m/s]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
datetick('x','MM:ss','keeplimits','keepticks')
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{y}_r\hspace{1mm}velocity$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rdp = subplot(3,3,8);
hold on;
plot(squareTrial.imuTwistDatetime,...
    squareTrial.imuTwistData(:,3),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/2);
plot(squareTrial.estRbtTwistDatetime,...
    squareTrial.estRbtTwistData(:,3),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\dot{\psi}\hspace{1mm}[rad/s]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
datetick('x','MM:ss','keeplimits','keepticks')
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{\psi}_r\hspace{1mm}velocity$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rddx = subplot(3,3,3);
hold on;
plot(squareTrial.imuAccDatetime,...
    squareTrial.imuAccData(:,1),...
    'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
plot(squareTrial.estRbtAccDatetime,...
    squareTrial.estRbtAccData(:,1),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\ddot{x}_r\hspace{1mm}[m/s^2]$','Interpreter','latex');
xlabel('$time [s]$','Interpreter','latex');
datetick('x','MM:ss','keeplimits','keepticks')
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
title('$Robot\hspace{1mm}Body\hspace{1mm}\ddot{x}_r\hspace{1mm}acceleration$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rddy = subplot(3,3,6);
hold on;
plot(squareTrial.imuAccDatetime,...
    squareTrial.imuAccData(:,2),...
    'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
plot(squareTrial.estRbtAccDatetime,...
    squareTrial.estRbtAccData(:,2),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\ddot{y}_r\hspace{1mm}[m/s^2]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
datetick('x','MM:ss','keeplimits','keepticks')
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
```

```
title('$Robot\hspace{1mm}Body\hspace{1mm}\ddot{y}_r\hspace{1mm}acceleration$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
legend('$measurement$',...
    '$estimator\hspace{1mm}output$',...
    'Interpreter','latex');
linkaxes([rx,ry,rp,rdx,rdy,rdp,rddx,rddy],'x');
set(gcf,'Position',[0 0 1920 1280]);

clear rdx rdy rdp;
figure(figNumStart+2);
fontSizeMod = 4;
clf;
mx = subplot(3,3,1);
hold on;
plot(squareTrial.groundTruthDatetime,...
    squareTrial.groundTruthData(:,1),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(squareTrial.gnssMapPoseDatetime,...
    squareTrial.gnssMapPoseData(:,1),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/1.5);
plot(squareTrial.estMapPoseDatetime,...
    squareTrial.estMapPoseData(:,1),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$x_m\hspace{1mm}[m]$','Interpreter','latex');
xlabel('$time [s]$','Interpreter','latex');
datetick('x','MM:ss','keeplimits','keepticks')
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
title('$Map\hspace{1mm}x_m\hspace{1mm}position$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
legend('$ground\hspace{1mm}truth$',...
    '$measurement$',...
    '$estimator\hspace{1mm}output$',...
    'Interpreter','latex');
my = subplot(3,3,4);
hold on;
plot(squareTrial.groundTruthDatetime,...
    squareTrial.groundTruthData(:,2),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(squareTrial.gnssMapPoseDatetime,...
    squareTrial.gnssMapPoseData(:,2),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/1.5);
plot(squareTrial.estMapPoseDatetime,...
    squareTrial.estMapPoseData(:,2),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$y_m\hspace{1mm}[m]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
datetick('x','MM:ss','keeplimits','keepticks')
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
title('$Map\hspace{1mm}y_m\hspace{1mm}position$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
mp = subplot(3,3,7);
hold on;
plot(squareTrial.imuPoseDatetime,...
    squareTrial.imuPoseData(:,3),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/2);
plot(squareTrial.estMapPoseDatetime,...
    squareTrial.estMapPoseData(:,3),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\psi_m\hspace{1mm}[rad]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
datetick('x','MM:ss','keeplimits','keepticks')
```

```
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    title('$Map\hspace{1mm}\psi_m\hspace{1mm}position$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    rdx = subplot(3,3,2);
    hold on;
    plot(squareTrial.estRbtTwistDatetime,...
        squareTrial.estRbtTwistData(:,2),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    plot(squareTrial.linVelAbsDatetime,...
        squareTrial.linVelAbsData(:,3),...
        'c','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{x}_r\hspace{1mm}[m/s]$','Interpreter','latex');
    xlabel('$time [s]$','Interpreter','latex');
    datetick('x','MM:ss','keeplimits','keepticks')
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{x}_r\hspace{1mm}velocity$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    rdy = subplot(3,3,5);
    hold on;
    plot(squareTrial.estRbtTwistDatetime,...
        squareTrial.estRbtTwistData(:,1),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    plot(squareTrial.linVelAbsDatetime,...
        squareTrial.linVelAbsData(:,3),...
        'c','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{y}_r\hspace{1mm}[m/s]$','Interpreter','latex');
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    datetick('x','MM:ss','keeplimits','keepticks')
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{y}_r\hspace{1mm}velocity$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    rdp = subplot(3,3,8);
    hold on;
    plot(squareTrial.imuTwistDatetime,...
        squareTrial.imuTwistData(:,3),...
        'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/2);
    plot(squareTrial.estRbtTwistDatetime,...
        squareTrial.estRbtTwistData(:,3),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{\psi}_r\hspace{1mm}[rad/s]$','Interpreter','latex');
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    datetick('x','MM:ss','keeplimits','keepticks')
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{\psi}_r\hspace{1mm}velocity$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    linkaxes([mx,my,mp,rdx,rdy,rdp],'x');
    set(gcf,'Position',[0 0 1920 1280]);

end
```

8.4. Post-processing Matlab code for circle ground truth .csv data

```
function processCircleBags()

    % Import data runs, set argument to 0 to import from .mat (fast), set
    % to 1 to import from actual .csv (slow)
    circle = importSquareData(0);

    % Manual data repairs
    circle = manualDataRepairs(circle);

    % Zero map frame data
    circle = zeroMapData(circle);

    % Create ground truth rectangle
    circle = createGroundTruthCircle(circle);

    % Calculate errors
    circle = calculateAbsoluteError(circle);
    circle = calculateInterpolationError(circle);

    % Plot stuff
    plotAllData(circle);

    % Output to workspace (optional)
    assignin('base','circle',circle);

end

function circle = importSquareData(csvImport)

    % Import from actual .csv files
    if (csvImport == 1)

        % CCW Fast 1
        circle.ccwFast1.gnssMapPose = 'circleCcwFast1/circleCcwFast1_estimated-est-gnss-map-
pose.csv';
        circle.ccwFast1.gnssRbtPose = 'circleCcwFast1/circleCcwFast1_estimated-est-gnss-rbt-
pose.csv';
        circle.ccwFast1.imuPose = 'circleCcwFast1/circleCcwFast1_estimated-est-imu-rbt-pose.csv';
        circle.ccwFast1.imuTwist = 'circleCcwFast1/circleCcwFast1_estimated-est-imu-rbt-
twist.csv';
        circle.ccwFast1.imuAcc = 'circleCcwFast1/circleCcwFast1_estimated-est-imu-rbt-acc.csv';
        circle.ccwFast1.estMapPose = 'circleCcwFast1/circleCcwFast1_estimated-est-map-pose.csv';
        circle.ccwFast1.estRbtPose = 'circleCcwFast1/circleCcwFast1_estimated-est-rbt-pose.csv';
        circle.ccwFast1.estRbtTwist = 'circleCcwFast1/circleCcwFast1_estimated-est-rbt-
twist.csv';
        circle.ccwFast1.estRbtAcc = 'circleCcwFast1/circleCcwFast1_estimated-est-rbt-acc.csv';
        circle.ccwFast1.estRbtLinVelAbs = 'circleCcwFast1/circleCcwFast1_estimated-est-rbt-
linVelAbs.csv';
        circle.ccwFast1 = importEstimatorBag(circle.ccwFast1);

        % CCW Fast 2
        circle.ccwFast2.gnssMapPose = 'circleCcwFast2/circleCcwFast2_estimated-est-gnss-map-
pose.csv';
        circle.ccwFast2.gnssRbtPose = 'circleCcwFast2/circleCcwFast2_estimated-est-gnss-rbt-
pose.csv';
        circle.ccwFast2.imuPose = 'circleCcwFast2/circleCcwFast2_estimated-est-imu-rbt-pose.csv';
        circle.ccwFast2.imuTwist = 'circleCcwFast2/circleCcwFast2_estimated-est-imu-rbt-
twist.csv';
        circle.ccwFast2.imuAcc = 'circleCcwFast2/circleCcwFast2_estimated-est-imu-rbt-acc.csv';
        circle.ccwFast2.estMapPose = 'circleCcwFast2/circleCcwFast2_estimated-est-map-pose.csv';
        circle.ccwFast2.estRbtPose = 'circleCcwFast2/circleCcwFast2_estimated-est-rbt-pose.csv';
        circle.ccwFast2.estRbtTwist = 'circleCcwFast2/circleCcwFast2_estimated-est-rbt-
twist.csv';
        circle.ccwFast2.estRbtAcc = 'circleCcwFast2/circleCcwFast2_estimated-est-rbt-acc.csv';
        circle.ccwFast2.estRbtLinVelAbs = 'circleCcwFast2/circleCcwFast2_estimated-est-rbt-
linVelAbs.csv';
        circle.ccwFast2 = importEstimatorBag(circle.ccwFast2);

        % CCW Fast 3
        circle.ccwFast3.gnssMapPose = 'circleCcwFast3/circleCcwFast3_estimated-est-gnss-map-
pose.csv';
        circle.ccwFast3.gnssRbtPose = 'circleCcwFast3/circleCcwFast3_estimated-est-gnss-rbt-
pose.csv';
        circle.ccwFast3.imuPose = 'circleCcwFast3/circleCcwFast3_estimated-est-imu-rbt-pose.csv';
        circle.ccwFast3.imuTwist = 'circleCcwFast3/circleCcwFast3_estimated-est-imu-rbt-
twist.csv';
        circle.ccwFast3.imuAcc = 'circleCcwFast3/circleCcwFast3_estimated-est-imu-rbt-acc.csv';
        circle.ccwFast3.estMapPose = 'circleCcwFast3/circleCcwFast3_estimated-est-map-pose.csv';
        circle.ccwFast3.estRbtPose = 'circleCcwFast3/circleCcwFast3_estimated-est-rbt-pose.csv';
        circle.ccwFast3.estRbtTwist = 'circleCcwFast3/circleCcwFast3_estimated-est-rbt-
twist.csv';
        circle.ccwFast3.estRbtAcc = 'circleCcwFast3/circleCcwFast3_estimated-est-rbt-acc.csv';
        circle.ccwFast3.estRbtLinVelAbs = 'circleCcwFast3/circleCcwFast3_estimated-est-rbt-
linVelAbs.csv';
        circle.ccwFast3 = importEstimatorBag(circle.ccwFast3);

        % CCW Slow 1
        circle.ccwSlow1.gnssMapPose = 'circleCcwSlow1/circleCcwSlow1_estimated-est-gnss-map-
pose.csv';
        circle.ccwSlow1.gnssRbtPose = 'circleCcwSlow1/circleCcwSlow1_estimated-est-gnss-rbt-
pose.csv';
        circle.ccwSlow1.imuPose = 'circleCcwSlow1/circleCcwSlow1_estimated-est-imu-rbt-pose.csv';
        circle.ccwSlow1.imuTwist = 'circleCcwSlow1/circleCcwSlow1_estimated-est-imu-rbt-
twist.csv';
        circle.ccwSlow1.imuAcc = 'circleCcwSlow1/circleCcwSlow1_estimated-est-imu-rbt-acc.csv';
        circle.ccwSlow1.estMapPose = 'circleCcwSlow1/circleCcwSlow1_estimated-est-map-pose.csv';
        circle.ccwSlow1.estRbtPose = 'circleCcwSlow1/circleCcwSlow1_estimated-est-rbt-pose.csv';
        circle.ccwSlow1.estRbtTwist = 'circleCcwSlow1/circleCcwSlow1_estimated-est-rbt-
twist.csv';
        circle.ccwSlow1.estRbtAcc = 'circleCcwSlow1/circleCcwSlow1_estimated-est-rbt-acc.csv';
        circle.ccwSlow1.estRbtLinVelAbs = 'circleCcwSlow1/circleCcwSlow1_estimated-est-rbt-
linVelAbs.csv';
        circle.ccwSlow1 = importEstimatorBag(circle.ccwSlow1);

        % CCW Slow 2
        circle.ccwSlow2.gnssMapPose = 'circleCcwSlow2/circleCcwSlow2_estimated-est-gnss-map-
pose.csv';
        circle.ccwSlow2.gnssRbtPose = 'circleCcwSlow2/circleCcwSlow2_estimated-est-gnss-rbt-
pose.csv';
        circle.ccwSlow2.imuPose = 'circleCcwSlow2/circleCcwSlow2_estimated-est-imu-rbt-pose.csv';
        circle.ccwSlow2.imuTwist = 'circleCcwSlow2/circleCcwSlow2_estimated-est-imu-rbt-
twist.csv';
        circle.ccwSlow2.imuAcc = 'circleCcwSlow2/circleCcwSlow2_estimated-est-imu-rbt-acc.csv';
        circle.ccwSlow2.estMapPose = 'circleCcwSlow2/circleCcwSlow2_estimated-est-map-pose.csv';
        circle.ccwSlow2.estRbtPose = 'circleCcwSlow2/circleCcwSlow2_estimated-est-rbt-pose.csv';
        circle.ccwSlow2.estRbtTwist = 'circleCcwSlow2/circleCcwSlow2_estimated-est-rbt-
twist.csv';
        circle.ccwSlow2.estRbtAcc = 'circleCcwSlow2/circleCcwSlow2_estimated-est-rbt-acc.csv';
        circle.ccwSlow2.estRbtLinVelAbs = 'circleCcwSlow2/circleCcwSlow2_estimated-est-rbt-
linVelAbs.csv';
        circle.ccwSlow2 = importEstimatorBag(circle.ccwSlow2);

        % CCW Slow 3
        circle.ccwSlow3.gnssMapPose = 'circleCcwSlow3/circleCcwSlow3_estimated-est-gnss-map-
pose.csv';
        circle.ccwSlow3.gnssRbtPose = 'circleCcwSlow3/circleCcwSlow3_estimated-est-gnss-rbt-
pose.csv';
        circle.ccwSlow3.imuPose = 'circleCcwSlow3/circleCcwSlow3_estimated-est-imu-rbt-pose.csv';
        circle.ccwSlow3.imuTwist = 'circleCcwSlow3/circleCcwSlow3_estimated-est-imu-rbt-
twist.csv';
        circle.ccwSlow3.imuAcc = 'circleCcwSlow3/circleCcwSlow3_estimated-est-imu-rbt-acc.csv';
        circle.ccwSlow3.estMapPose = 'circleCcwSlow3/circleCcwSlow3_estimated-est-map-pose.csv';
        circle.ccwSlow3.estRbtPose = 'circleCcwSlow3/circleCcwSlow3_estimated-est-rbt-pose.csv';
        circle.ccwSlow3.estRbtTwist = 'circleCcwSlow3/circleCcwSlow3_estimated-est-rbt-
twist.csv';
        circle.ccwSlow3.estRbtAcc = 'circleCcwSlow3/circleCcwSlow3_estimated-est-rbt-acc.csv';
        circle.ccwSlow3.estRbtLinVelAbs = 'circleCcwSlow3/circleCcwSlow3_estimated-est-rbt-
linVelAbs.csv';
        circle.ccwSlow3 = importEstimatorBag(circle.ccwSlow3);

        % CCW Slow 4
        circle.ccwSlow4.gnssMapPose = 'circleCcwSlow4/circleCcwSlow4_estimated-est-gnss-map-
pose.csv';
```

```
        circle.ccwSlow4.gnssRbtPose = 'circleCcwSlow4/circleCcwSlow4_estimated-est-gnss-rbt-
pose.csv';
        circle.ccwSlow4.imuPose = 'circleCcwSlow4/circleCcwSlow4_estimated-est-imu-rbt-pose.csv';
        circle.ccwSlow4.imuTwist = 'circleCcwSlow4/circleCcwSlow4_estimated-est-imu-rbt-
twist.csv';
        circle.ccwSlow4.imuAcc = 'circleCcwSlow4/circleCcwSlow4_estimated-est-imu-rbt-acc.csv';
        circle.ccwSlow4.estMapPose = 'circleCcwSlow4/circleCcwSlow4_estimated-est-map-pose.csv';
        circle.ccwSlow4.estRbtPose = 'circleCcwSlow4/circleCcwSlow4_estimated-est-rbt-pose.csv';
        circle.ccwSlow4.estRbtTwist = 'circleCcwSlow4/circleCcwSlow4_estimated-est-rbt-
twist.csv';
        circle.ccwSlow4.estRbtAcc = 'circleCcwSlow4/circleCcwSlow4_estimated-est-rbt-acc.csv';
        circle.ccwSlow4.estRbtLinVelAbs = 'circleCcwSlow4/circleCcwSlow4_estimated-est-rbt-
linVelAbs.csv';
        circle.ccwSlow4 = importEstimatorBag(circle.ccwSlow4);

        % Save to file
        save('circle.mat','circle');

    % Import from .mat file
    else

        load('circle.mat');

    end

end

function trial = importEstimatorBag(trial)

    % Sensor refresh rate for approximate time vectors
    gnssRR = 1;     % GNSS refresh rate in [Hz]
    imuRR = 20;     % IMU refresh rate in [Hz]
    estRR = 40;     % estimator refresh rate in [Hz]

    % GNSS map pose
    string = readmatrix(trial.gnssMapPose,...
        'Delimiter',',','OutputType','string');     % read in GNSS map pose as a string (for time
data)
    num = readmatrix(trial.gnssMapPose,...
        'Delimiter',',');                           % read in GNSS map pose as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/gnssRR:(row-2)*(1/gnssRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.gnssMapPoseDatetime = dt;
    trial.gnssMapPoseData = num(:,2:4);

    % GNSS robot pose
    string = readmatrix(trial.gnssRbtPose,...
        'Delimiter',',','OutputType','string');     % read in GNSS robot pose as a string (for
time data)
    num = readmatrix(trial.gnssRbtPose,...
        'Delimiter',',');                           % read in GNSS robot pose as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/gnssRR:(row-2)*(1/gnssRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.gnssRbtPoseDatetime = dt;
    trial.gnssRbtPoseData = num(:,2:4);

    % IMU pose
    string = readmatrix(trial.imuPose,...
        'Delimiter',',','OutputType','string');     % read in IMU pose as a string (for time
data)
    num = readmatrix(trial.imuPose,...
        'Delimiter',',');                           % read in IMU pose as doubles (for actual
data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/imuRR:(row-2)*(1/imuRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.imuPoseDatetime = dt;
    trial.imuPoseData = num(:,2:4);

    % IMU twist
    string = readmatrix(trial.imuTwist,...
        'Delimiter',',','OutputType','string');     % read in IMU twist as a string (for time
data)
    num = readmatrix(trial.imuTwist,...
        'Delimiter',',');                           % read in IMU twist as doubles (for actual
data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/imuRR:(row-2)*(1/imuRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.imuTwistDatetime = dt;
    trial.imuTwistData = num(:,2:4);

    % IMU acceleration
    string = readmatrix(trial.imuAcc,...
        'Delimiter',',','OutputType','string');     % read in IMU acceleration as a string (for
time data)
    num = readmatrix(trial.imuAcc,...
        'Delimiter',',');                           % read in IMU acceleration as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/imuRR:(row-2)*(1/imuRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.imuAccDatetime = dt;
    trial.imuAccData = num(:,2:4);

    % Estimator map pose
    string = readmatrix(trial.estMapPose,...
        'Delimiter',',','OutputType','string');     % read in estimator map pose as a string (for
time data)
    num = readmatrix(trial.estMapPose,...
        'Delimiter',',');                           % read in estimator map pose as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estMapPoseDatetime = dt;
    trial.estMapPoseData = num(:,2:4);

    % Estimator robot pose
    string = readmatrix(trial.estRbtPose,...
        'Delimiter',',','OutputType','string');     % read in estimator robot pose as a string
(for time data)
    num = readmatrix(trial.estRbtPose,...
        'Delimiter',',');                           % read in estimator robot pose as doubles
(for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estRbtPoseDatetime = dt;
    trial.estRbtPoseData = num(:,2:4);

    % Estimator robot twist
    string = readmatrix(trial.estRbtTwist,...
        'Delimiter',',','OutputType','string');     % read in estimator robot twist as a string
(for time data)
    num = readmatrix(trial.estRbtTwist,...
        'Delimiter',',');                            % read in estimator robot twist as doubles
(for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estRbtTwistDatetime = dt;
    trial.estRbtTwistData = num(:,2:4);

    % Estimator robot acceleration
    string = readmatrix(trial.estRbtAcc,...
        'Delimiter',',','OutputType','string');     % read in estimator robot acceleration as a
string (for time data)
    num = readmatrix(trial.estRbtAcc,...
        'Delimiter',',');                            % read in estimator robot acceleration as
doubles (for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estRbtAccDatetime = dt;
    trial.estRbtAccData = num(:,2:4);

    % Estimator robot absolute linear velocity vector
    string = readmatrix(trial.estRbtLinVelAbs,...
        'Delimiter',',','OutputType','string');     % read in estimator robot absolute linear
velocity vector as a string (for time data)
    num = readmatrix(trial.estRbtLinVelAbs,...
        'Delimiter',',');                           % read in estimator robot absolute linear
velocity vector as doubles (for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.linVelAbsDatetime = dt;
    trial.linVelAbsData = num(:,2:4);

end

function circle = manualDataRepairs(circle)

    % CCW Fast 1
    gnss0 = 1;
    imu0 = 25;
    est0 = 135;
    circle.ccwFast1.gnssMapPoseDatetime(1:gnss0,:) = [];
    circle.ccwFast1.gnssMapPoseData(1:gnss0,:) = [];
    circle.ccwFast1.gnssRbtPoseDatetime(1:gnss0,:) = [];
    circle.ccwFast1.gnssRbtPoseData(1:gnss0,:) = [];
    circle.ccwFast1.imuPoseDatetime(1:imu0,:) = [];
    circle.ccwFast1.imuPoseData(1:imu0,:) = [];
    circle.ccwFast1.imuTwistDatetime(1:imu0,:) = [];
    circle.ccwFast1.imuTwistData(1:imu0,:) = [];
    circle.ccwFast1.imuAccDatetime(1:imu0,:) = [];
    circle.ccwFast1.imuAccData(1:imu0,:) = [];
    circle.ccwFast1.estMapPoseDatetime(1:est0,:) = [];
    circle.ccwFast1.estMapPoseData(1:est0,:) = [];
    circle.ccwFast1.estRbtPoseDatetime(1:est0,:) = [];
    circle.ccwFast1.estRbtPoseData(1:est0,:) = [];
    circle.ccwFast1.estRbtTwistDatetime(1:est0,:) = [];
    circle.ccwFast1.estRbtTwistData(1:est0,:) = [];
    circle.ccwFast1.estRbtAccDatetime(1:est0,:) = [];
    circle.ccwFast1.estRbtAccData(1:est0,:) = [];
    circle.ccwFast1.linVelAbsDatetime(1:est0,:) = [];
    circle.ccwFast1.linVelAbsData(1:est0,:) = [];

    % CCW Fast 2
    gnss0 = 0;
    imu0 = 25;
    est0 = 60;
    circle.ccwFast2.gnssMapPoseDatetime(1:gnss0,:) = [];
    circle.ccwFast2.gnssMapPoseData(1:gnss0,:) = [];
    circle.ccwFast2.gnssRbtPoseDatetime(1:gnss0,:) = [];
    circle.ccwFast2.gnssRbtPoseData(1:gnss0,:) = [];
    circle.ccwFast2.imuPoseDatetime(1:imu0,:) = [];
    circle.ccwFast2.imuPoseData(1:imu0,:) = [];
    circle.ccwFast2.imuTwistDatetime(1:imu0,:) = [];
    circle.ccwFast2.imuTwistData(1:imu0,:) = [];
    circle.ccwFast2.imuAccDatetime(1:imu0,:) = [];
    circle.ccwFast2.imuAccData(1:imu0,:) = [];
    circle.ccwFast2.estMapPoseDatetime(1:est0,:) = [];
    circle.ccwFast2.estMapPoseData(1:est0,:) = [];
    circle.ccwFast2.estRbtPoseDatetime(1:est0,:) = [];
    circle.ccwFast2.estRbtPoseData(1:est0,:) = [];
    circle.ccwFast2.estRbtTwistDatetime(1:est0,:) = [];
    circle.ccwFast2.estRbtTwistData(1:est0,:) = [];
    circle.ccwFast2.estRbtAccDatetime(1:est0,:) = [];
    circle.ccwFast2.estRbtAccData(1:est0,:) = [];
    circle.ccwFast2.linVelAbsDatetime(1:est0,:) = [];
    circle.ccwFast2.linVelAbsData(1:est0,:) = [];

    % CCW Fast 3
    gnss0 = 1;
    imu0 = 22;
    est0 = 135;
    circle.ccwFast3.gnssMapPoseDatetime(1:gnss0,:) = [];
    circle.ccwFast3.gnssMapPoseData(1:gnss0,:) = [];
    circle.ccwFast3.gnssRbtPoseDatetime(1:gnss0,:) = [];
    circle.ccwFast3.gnssRbtPoseData(1:gnss0,:) = [];
    circle.ccwFast3.imuPoseDatetime(1:imu0,:) = [];
    circle.ccwFast3.imuPoseData(1:imu0,:) = [];
    circle.ccwFast3.imuTwistDatetime(1:imu0,:) = [];
    circle.ccwFast3.imuTwistData(1:imu0,:) = [];
    circle.ccwFast3.imuAccDatetime(1:imu0,:) = [];
    circle.ccwFast3.imuAccData(1:imu0,:) = [];
    circle.ccwFast3.estMapPoseDatetime(1:est0,:) = [];
    circle.ccwFast3.estMapPoseData(1:est0,:) = [];
    circle.ccwFast3.estRbtPoseDatetime(1:est0,:) = [];
    circle.ccwFast3.estRbtPoseData(1:est0,:) = [];
    circle.ccwFast3.estRbtTwistDatetime(1:est0,:) = [];
    circle.ccwFast3.estRbtTwistData(1:est0,:) = [];
    circle.ccwFast3.estRbtAccDatetime(1:est0,:) = [];
    circle.ccwFast3.estRbtAccData(1:est0,:) = [];
    circle.ccwFast3.linVelAbsDatetime(1:est0,:) = [];
    circle.ccwFast3.linVelAbsData(1:est0,:) = [];

    % CCW Slow 1
    gnss0 = 0;
    imu0 = 25;
    est0 = 60;
    circle.ccwSlow1.gnssMapPoseDatetime(1:gnss0,:) = [];
    circle.ccwSlow1.gnssMapPoseData(1:gnss0,:) = [];
    circle.ccwSlow1.gnssRbtPoseDatetime(1:gnss0,:) = [];
    circle.ccwSlow1.gnssRbtPoseData(1:gnss0,:) = [];
    circle.ccwSlow1.imuPoseDatetime(1:imu0,:) = [];
    circle.ccwSlow1.imuPoseData(1:imu0,:) = [];
    circle.ccwSlow1.imuTwistDatetime(1:imu0,:) = [];
    circle.ccwSlow1.imuTwistData(1:imu0,:) = [];
    circle.ccwSlow1.imuAccDatetime(1:imu0,:) = [];
    circle.ccwSlow1.imuAccData(1:imu0,:) = [];
    circle.ccwSlow1.estMapPoseDatetime(1:est0,:) = [];
    circle.ccwSlow1.estMapPoseData(1:est0,:) = [];
    circle.ccwSlow1.estRbtPoseDatetime(1:est0,:) = [];
    circle.ccwSlow1.estRbtPoseData(1:est0,:) = [];
    circle.ccwSlow1.estRbtTwistDatetime(1:est0,:) = [];
    circle.ccwSlow1.estRbtTwistData(1:est0,:) = [];
    circle.ccwSlow1.estRbtAccDatetime(1:est0,:) = [];
    circle.ccwSlow1.estRbtAccData(1:est0,:) = [];
    circle.ccwSlow1.linVelAbsDatetime(1:est0,:) = [];
    circle.ccwSlow1.linVelAbsData(1:est0,:) = [];

    % CCW Slow 2
    gnss0 = 1;
    imu0 = 24;
    est0 = 60;
    circle.ccwSlow2.gnssMapPoseDatetime(1:gnss0,:) = [];
    circle.ccwSlow2.gnssMapPoseData(1:gnss0,:) = [];
    circle.ccwSlow2.gnssRbtPoseDatetime(1:gnss0,:) = [];
    circle.ccwSlow2.gnssRbtPoseData(1:gnss0,:) = [];
    circle.ccwSlow2.imuPoseDatetime(1:imu0,:) = [];
    circle.ccwSlow2.imuPoseData(1:imu0,:) = [];
    circle.ccwSlow2.imuTwistDatetime(1:imu0,:) = [];
```

```
    circle.ccwSlow2.imuTwistData(1:imu0,:) = [];
    circle.ccwSlow2.imuAccDatetime(1:imu0,:) = [];
    circle.ccwSlow2.imuAccData(1:imu0,:) = [];
    circle.ccwSlow2.estMapPoseDatetime(1:est0,:) = [];
    circle.ccwSlow2.estMapPoseData(1:est0,:) = [];
    circle.ccwSlow2.estRbtPoseDatetime(1:est0,:) = [];
    circle.ccwSlow2.estRbtPoseData(1:est0,:) = [];
    circle.ccwSlow2.estRbtTwistDatetime(1:est0,:) = [];
    circle.ccwSlow2.estRbtTwistData(1:est0,:) = [];
    circle.ccwSlow2.estRbtAccDatetime(1:est0,:) = [];
    circle.ccwSlow2.estRbtAccData(1:est0,:) = [];
    circle.ccwSlow2.linVelAbsDatetime(1:est0,:) = [];
    circle.ccwSlow2.linVelAbsData(1:est0,:) = [];

    % CCW Slow 3
    gnss0 = 2;
    imu0 = 26;
    est0 = 60;
    circle.ccwSlow3.gnssMapPoseDatetime(1:gnss0,:) = [];
    circle.ccwSlow3.gnssMapPoseData(1:gnss0,:) = [];
    circle.ccwSlow3.gnssRbtPoseDatetime(1:gnss0,:) = [];
    circle.ccwSlow3.gnssRbtPoseData(1:gnss0,:) = [];
    circle.ccwSlow3.imuPoseDatetime(1:imu0,:) = [];
    circle.ccwSlow3.imuPoseData(1:imu0,:) = [];
    circle.ccwSlow3.imuTwistDatetime(1:imu0,:) = [];
    circle.ccwSlow3.imuTwistData(1:imu0,:) = [];
    circle.ccwSlow3.imuAccDatetime(1:imu0,:) = [];
    circle.ccwSlow3.imuAccData(1:imu0,:) = [];
    circle.ccwSlow3.estMapPoseDatetime(1:est0,:) = [];
    circle.ccwSlow3.estMapPoseData(1:est0,:) = [];
    circle.ccwSlow3.estRbtPoseDatetime(1:est0,:) = [];
    circle.ccwSlow3.estRbtPoseData(1:est0,:) = [];
    circle.ccwSlow3.estRbtTwistDatetime(1:est0,:) = [];
    circle.ccwSlow3.estRbtTwistData(1:est0,:) = [];
    circle.ccwSlow3.estRbtAccDatetime(1:est0,:) = [];
    circle.ccwSlow3.estRbtAccData(1:est0,:) = [];
    circle.ccwSlow3.linVelAbsDatetime(1:est0,:) = [];
    circle.ccwSlow3.linVelAbsData(1:est0,:) = [];

    % CCW Slow 4
    gnss0 = 1;
    imu0 = 26;
    est0 = 90;
    circle.ccwSlow4.gnssMapPoseDatetime(1:gnss0,:) = [];
    circle.ccwSlow4.gnssMapPoseData(1:gnss0,:) = [];
    circle.ccwSlow4.gnssRbtPoseDatetime(1:gnss0,:) = [];
    circle.ccwSlow4.gnssRbtPoseData(1:gnss0,:) = [];
    circle.ccwSlow4.imuPoseDatetime(1:imu0,:) = [];
    circle.ccwSlow4.imuPoseData(1:imu0,:) = [];
    circle.ccwSlow4.imuTwistDatetime(1:imu0,:) = [];
    circle.ccwSlow4.imuTwistData(1:imu0,:) = [];
    circle.ccwSlow4.imuAccDatetime(1:imu0,:) = [];
    circle.ccwSlow4.imuAccData(1:imu0,:) = [];
    circle.ccwSlow4.estMapPoseDatetime(1:est0,:) = [];
    circle.ccwSlow4.estMapPoseData(1:est0,:) = [];
    circle.ccwSlow4.estRbtPoseDatetime(1:est0,:) = [];
    circle.ccwSlow4.estRbtPoseData(1:est0,:) = [];
    circle.ccwSlow4.estRbtTwistDatetime(1:est0,:) = [];
    circle.ccwSlow4.estRbtTwistData(1:est0,:) = [];
    circle.ccwSlow4.estRbtAccDatetime(1:est0,:) = [];
    circle.ccwSlow4.estRbtAccData(1:est0,:) = [];
    circle.ccwSlow4.linVelAbsDatetime(1:est0,:) = [];
    circle.ccwSlow4.linVelAbsData(1:est0,:) = [];

end

function circle = zeroMapData(circle)

    % CCW Fast 1
    mapZero = circle.ccwFast1.gnssMapPoseData(1,:);
```

```
circle.ccwFast1.gnssMapPoseData =...
    circle.ccwFast1.gnssMapPoseData-mapZero;
circle.ccwFast1.estMapPoseData =...
    circle.ccwFast1.estMapPoseData-mapZero;
rbtZero = circle.ccwFast1.gnssRbtPoseData(1,:);
circle.ccwFast1.gnssRbtPoseData =...
    circle.ccwFast1.gnssRbtPoseData-rbtZero;
circle.ccwFast1.estRbtPoseData =...
    circle.ccwFast1.estRbtPoseData-rbtZero;

% CCW Fast 2
mapZero = circle.ccwFast2.gnssMapPoseData(1,:);
circle.ccwFast2.gnssMapPoseData =...
    circle.ccwFast2.gnssMapPoseData-mapZero;
circle.ccwFast2.estMapPoseData =...
    circle.ccwFast2.estMapPoseData-mapZero;
rbtZero = circle.ccwFast2.gnssRbtPoseData(1,:);
circle.ccwFast2.gnssRbtPoseData =...
    circle.ccwFast2.gnssRbtPoseData-rbtZero;
circle.ccwFast2.estRbtPoseData =...
    circle.ccwFast2.estRbtPoseData-rbtZero;

% CCW Fast 3
mapZero = circle.ccwFast3.gnssMapPoseData(1,:);
circle.ccwFast3.gnssMapPoseData =...
    circle.ccwFast3.gnssMapPoseData-mapZero;
circle.ccwFast3.estMapPoseData =...
    circle.ccwFast3.estMapPoseData-mapZero;
rbtZero = circle.ccwFast3.gnssRbtPoseData(1,:);
circle.ccwFast3.gnssRbtPoseData =...
    circle.ccwFast3.gnssRbtPoseData-rbtZero;
circle.ccwFast3.estRbtPoseData =...
    circle.ccwFast3.estRbtPoseData-rbtZero;

% CCW Slow 1
mapZero = circle.ccwSlow1.gnssMapPoseData(1,:);
circle.ccwSlow1.gnssMapPoseData =...
    circle.ccwSlow1.gnssMapPoseData-mapZero;
circle.ccwSlow1.estMapPoseData =...
    circle.ccwSlow1.estMapPoseData-mapZero;
rbtZero = circle.ccwSlow1.gnssRbtPoseData(1,:);
circle.ccwSlow1.gnssRbtPoseData =...
    circle.ccwSlow1.gnssRbtPoseData-rbtZero;
circle.ccwSlow1.estRbtPoseData =...
    circle.ccwSlow1.estRbtPoseData-rbtZero;

% CCW Slow 2
mapZero = circle.ccwSlow2.gnssMapPoseData(1,:);
circle.ccwSlow2.gnssMapPoseData =...
    circle.ccwSlow2.gnssMapPoseData-mapZero;
circle.ccwSlow2.estMapPoseData =...
    circle.ccwSlow2.estMapPoseData-mapZero;
rbtZero = circle.ccwSlow2.gnssRbtPoseData(1,:);
circle.ccwSlow2.gnssRbtPoseData =...
    circle.ccwSlow2.gnssRbtPoseData-rbtZero;
circle.ccwSlow2.estRbtPoseData =...
    circle.ccwSlow2.estRbtPoseData-rbtZero;

% CCW Slow 3
mapZero = circle.ccwSlow3.gnssMapPoseData(1,:);
circle.ccwSlow3.gnssMapPoseData =...
    circle.ccwSlow3.gnssMapPoseData-mapZero;
circle.ccwSlow3.estMapPoseData =...
    circle.ccwSlow3.estMapPoseData-mapZero;
rbtZero = circle.ccwSlow3.gnssRbtPoseData(1,:);
circle.ccwSlow3.gnssRbtPoseData =...
    circle.ccwSlow3.gnssRbtPoseData-rbtZero;
circle.ccwSlow3.estRbtPoseData =...
    circle.ccwSlow3.estRbtPoseData-rbtZero;
    % CCW Slow 4
    mapZero = circle.ccwSlow4.gnssMapPoseData(1,:);
    circle.ccwSlow4.gnssMapPoseData =...
        circle.ccwSlow4.gnssMapPoseData-mapZero;
    circle.ccwSlow4.estMapPoseData =...
        circle.ccwSlow4.estMapPoseData-mapZero;
    rbtZero = circle.ccwSlow4.gnssRbtPoseData(1,:);
    circle.ccwSlow4.gnssRbtPoseData =...
        circle.ccwSlow4.gnssRbtPoseData-rbtZero;
    circle.ccwSlow4.estRbtPoseData =...
        circle.ccwSlow4.estRbtPoseData-rbtZero;

    % Save to file
    save('circleCorrected.mat','circle');

end

function circle = createGroundTruthCircle(circle)

    % Properties of the circle (can't change with arguments)
    diameter = 30.5;
    per = diameter*pi;
    radius = diameter/2;
    center = [-radius,0];

    % CCW Fast 1 (37.3 s)
    circle.ccwFast1.runTime = 37.3;
    circle.ccwFast1.meanVel = per/circle.ccwFast1.runTime;
    res = 1E4;
    theta = linspace(0,2*pi,res);
    circle.ccwFast1.groundTruthData(:,1) =...
        radius*cos(theta)+center(1,1);
    circle.ccwFast1.groundTruthData(:,2) =...
        radius*sin(theta)+center(1,2);
    timeVec = linspace(0,circle.ccwFast1.runTime,res);
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    circle.ccwFast1.groundTruthDatetime = dt;

    % CCW Fast 2 (37.2 s)
    circle.ccwFast2.runTime = 37.2;
    circle.ccwFast2.meanVel = per/circle.ccwFast2.runTime;
    res = 1E4;
    theta = linspace(0,2*pi,res);
    circle.ccwFast2.groundTruthData(:,1) =...
        radius*cos(theta)+center(1,1);
    circle.ccwFast2.groundTruthData(:,2) =...
        radius*sin(theta)+center(1,2);
    timeVec = linspace(0,circle.ccwFast2.runTime,res);
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    circle.ccwFast2.groundTruthDatetime = dt;

    % CCW Fast 3 (37.2 s)
    circle.ccwFast3.runTime = 37.2;
    circle.ccwFast3.meanVel = per/circle.ccwFast3.runTime;
```

```
res = 1E4;
theta = linspace(0,2*pi,res);
circle.ccwFast3.groundTruthData(:,1) =...
    radius*cos(theta)+center(1,1);
circle.ccwFast3.groundTruthData(:,2) =...
    radius*sin(theta)+center(1,2);
timeVec = linspace(0,circle.ccwFast3.runTime,res);
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
circle.ccwFast3.groundTruthDatetime = dt;

% CCW Slow 1 (77.8 s)
circle.ccwSlow1.runTime = 77.8;
circle.ccwSlow1.meanVel = per/circle.ccwSlow1.runTime;
res = 1E4;
theta = linspace(0,2*pi,res);
circle.ccwSlow1.groundTruthData(:,1) =...
    radius*cos(theta)+center(1,1);
circle.ccwSlow1.groundTruthData(:,2) =...
    radius*sin(theta)+center(1,2);
timeVec = linspace(0,circle.ccwSlow1.runTime,res);
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
circle.ccwSlow1.groundTruthDatetime = dt;

% CCW Slow 2 (77.6 s)
circle.ccwSlow2.runTime = 77.6;
circle.ccwSlow2.meanVel = per/circle.ccwSlow2.runTime;
res = 1E4;
theta = linspace(0,2*pi,res);
circle.ccwSlow2.groundTruthData(:,1) =...
    radius*cos(theta)+center(1,1);
circle.ccwSlow2.groundTruthData(:,2) =...
    radius*sin(theta)+center(1,2);
timeVec = linspace(0,circle.ccwSlow2.runTime,res);
Y = (ones(size(timeVec))*2019);
M = (ones(size(timeVec))*1);
D = (ones(size(timeVec))*1);
H = (ones(size(timeVec))*0);
MI = (ones(size(timeVec))*0);
S = floor(timeVec);
MS = (timeVec-floor(timeVec))*1E3;
dt = datetime(Y,M,D,H,MI,S,MS,...
    'Format','yyyy-MM-dd HH:mm:ss.SSS');
circle.ccwSlow2.groundTruthDatetime = dt;

% CCW Slow 3 (76.9 s)
circle.ccwSlow3.runTime = 76.9;
circle.ccwSlow3.meanVel = per/circle.ccwSlow3.runTime;
res = 1E4;
theta = linspace(0,2*pi,res);
circle.ccwSlow3.groundTruthData(:,1) =...
    radius*cos(theta)+center(1,1);
circle.ccwSlow3.groundTruthData(:,2) =...
    radius*sin(theta)+center(1,2);
timeVec = linspace(0,circle.ccwSlow3.runTime,res);
```

```
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    circle.ccwSlow3.groundTruthDatetime = dt;

    % CCW Slow 4 (75.9 s)
    circle.ccwSlow4.runTime = 75.9;
    circle.ccwSlow4.meanVel = per/circle.ccwSlow4.runTime;
    res = 1E4;
    theta = linspace(0,2*pi,res);
    circle.ccwSlow4.groundTruthData(:,1) =...
        radius*cos(theta)+center(1,1);
    circle.ccwSlow4.groundTruthData(:,2) =...
        radius*sin(theta)+center(1,2);
    timeVec = linspace(0,circle.ccwSlow4.runTime,res);
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    circle.ccwSlow4.groundTruthDatetime = dt;

    % Save to file
    save('circleCorrectedWithCircle.mat','circle');

end

function circle = calculateAbsoluteError(circle)

    % CCW Fast 1 pose accuracy
    circle.ccwFast1.groundTruthTimetable =...
        array2timetable(circle.ccwFast1.groundTruthData,...
        'RowTimes',circle.ccwFast1.groundTruthDatetime);        % create timetable for
groundTruth
    circle.ccwFast1.gnssMapPoseTimetable =...
        array2timetable(circle.ccwFast1.gnssMapPoseData,...
        'RowTimes',circle.ccwFast1.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwFast1.estMapPoseTimetable =...
        array2timetable(circle.ccwFast1.estMapPoseData,...
        'RowTimes',circle.ccwFast1.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwFast1.groundTruthGnssMapErrorTimetable =...
        synchronize(circle.ccwFast1.gnssMapPoseTimetable,...
        circle.ccwFast1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    circle.ccwFast1.groundTruthEstMapErrorTimetable =...
        synchronize(circle.ccwFast1.estMapPoseTimetable,...
        circle.ccwFast1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        circle.ccwFast1.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwFast1.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        circle.ccwFast1.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwFast1.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    circle.ccwFast1.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    circle.ccwFast1.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    circle.ccwFast1.absErrorMeanGnssMapXY =...
        sqrt(circle.ccwFast1.absErrorMeanGnssMapX^2+...
        circle.ccwFast1.absErrorMeanGnssMapY^2);
    circle.ccwFast1.absErrorStdGnssMapX = std(errorVecGnssMapX);
    circle.ccwFast1.absErrorStdGnssMapY = std(errorVecGnssMapY);
    circle.ccwFast1.absErrorStdGnssMapXY =...
        sqrt(circle.ccwFast1.absErrorStdGnssMapX^2+...
        circle.ccwFast1.absErrorStdGnssMapY^2);
    syncEstX =...
        circle.ccwFast1.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwFast1.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwFast1.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwFast1.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    circle.ccwFast1.absErrorMeanEstMapX = mean(errorVecEstMapX);
    circle.ccwFast1.absErrorMeanEstMapY = mean(errorVecEstMapY);
    circle.ccwFast1.absErrorMeanEstMapXY =...
        sqrt(circle.ccwFast1.absErrorMeanEstMapX^2+...
        circle.ccwFast1.absErrorMeanEstMapY^2);
    circle.ccwFast1.absErrorStdEstMapX = std(errorVecEstMapX);
    circle.ccwFast1.absErrorStdEstMapY = std(errorVecEstMapY);
    circle.ccwFast1.absErrorStdEstMapXY =...
        sqrt(circle.ccwFast1.absErrorStdEstMapX^2+...
        circle.ccwFast1.absErrorStdEstMapY^2);
    circle.ccwFast1.gnssToEstMeanErrorChange =...
        (circle.ccwFast1.absErrorMeanEstMapXY-...
        circle.ccwFast1.absErrorMeanGnssMapXY)/...
        circle.ccwFast1.absErrorMeanGnssMapXY;
    circle.ccwFast1.gnssToEstStdErrorChange =...
        (circle.ccwFast1.absErrorStdEstMapXY-...
        circle.ccwFast1.absErrorStdGnssMapXY)/...
        circle.ccwFast1.absErrorStdGnssMapXY;

    % CCW Fast 2 pose accuracy
    circle.ccwFast2.groundTruthTimetable =...
        array2timetable(circle.ccwFast2.groundTruthData,...
        'RowTimes',circle.ccwFast2.groundTruthDatetime);        % create timetable for
groundTruth
    circle.ccwFast2.gnssMapPoseTimetable =...
        array2timetable(circle.ccwFast2.gnssMapPoseData,...
        'RowTimes',circle.ccwFast2.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwFast2.estMapPoseTimetable =...
        array2timetable(circle.ccwFast2.estMapPoseData,...
        'RowTimes',circle.ccwFast2.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwFast2.groundTruthGnssMapErrorTimetable =...
        synchronize(circle.ccwFast2.gnssMapPoseTimetable,...
        circle.ccwFast2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    circle.ccwFast2.groundTruthEstMapErrorTimetable =...
        synchronize(circle.ccwFast2.estMapPoseTimetable,...
        circle.ccwFast2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        circle.ccwFast2.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwFast2.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        circle.ccwFast2.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwFast2.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    circle.ccwFast2.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    circle.ccwFast2.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    circle.ccwFast2.absErrorMeanGnssMapXY =...
        sqrt(circle.ccwFast2.absErrorMeanGnssMapX^2+...
        circle.ccwFast2.absErrorMeanGnssMapY^2);
    circle.ccwFast2.absErrorStdGnssMapX = std(errorVecGnssMapX);
    circle.ccwFast2.absErrorStdGnssMapY = std(errorVecGnssMapY);
    circle.ccwFast2.absErrorStdGnssMapXY =...
        sqrt(circle.ccwFast2.absErrorStdGnssMapX^2+...
        circle.ccwFast2.absErrorStdGnssMapY^2);
    syncEstX =...
        circle.ccwFast2.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwFast2.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwFast2.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwFast2.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    circle.ccwFast2.absErrorMeanEstMapX = mean(errorVecEstMapX);
    circle.ccwFast2.absErrorMeanEstMapY = mean(errorVecEstMapY);
    circle.ccwFast2.absErrorMeanEstMapXY =...
        sqrt(circle.ccwFast2.absErrorMeanEstMapX^2+...
        circle.ccwFast2.absErrorMeanEstMapY^2);
    circle.ccwFast2.absErrorStdEstMapX = std(errorVecEstMapX);
    circle.ccwFast2.absErrorStdEstMapY = std(errorVecEstMapY);
    circle.ccwFast2.absErrorStdEstMapXY =...
        sqrt(circle.ccwFast2.absErrorStdEstMapX^2+...
        circle.ccwFast2.absErrorStdEstMapY^2);
    circle.ccwFast2.gnssToEstMeanErrorChange =...
        (circle.ccwFast2.absErrorMeanEstMapXY-...
        circle.ccwFast2.absErrorMeanGnssMapXY)/...
        circle.ccwFast2.absErrorMeanGnssMapXY;
    circle.ccwFast2.gnssToEstStdErrorChange =...
        (circle.ccwFast2.absErrorStdEstMapXY-...
        circle.ccwFast2.absErrorStdGnssMapXY)/...
        circle.ccwFast2.absErrorStdGnssMapXY;

    % CCW Fast 3 pose accuracy
    circle.ccwFast3.groundTruthTimetable =...
        array2timetable(circle.ccwFast3.groundTruthData,...
        'RowTimes',circle.ccwFast3.groundTruthDatetime);        % create timetable for
groundTruth
    circle.ccwFast3.gnssMapPoseTimetable =...
        array2timetable(circle.ccwFast3.gnssMapPoseData,...
        'RowTimes',circle.ccwFast3.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwFast3.estMapPoseTimetable =...
        array2timetable(circle.ccwFast3.estMapPoseData,...
        'RowTimes',circle.ccwFast3.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwFast3.groundTruthGnssMapErrorTimetable =...
        synchronize(circle.ccwFast3.gnssMapPoseTimetable,...
        circle.ccwFast3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    circle.ccwFast3.groundTruthEstMapErrorTimetable =...
        synchronize(circle.ccwFast3.estMapPoseTimetable,...
        circle.ccwFast3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        circle.ccwFast3.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwFast3.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        circle.ccwFast3.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwFast3.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    circle.ccwFast3.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    circle.ccwFast3.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    circle.ccwFast3.absErrorMeanGnssMapXY =...
```

```
        sqrt(circle.ccwFast3.absErrorMeanGnssMapX^2+...
        circle.ccwFast3.absErrorMeanGnssMapY^2);
    circle.ccwFast3.absErrorStdGnssMapX = std(errorVecGnssMapX);
    circle.ccwFast3.absErrorStdGnssMapY = std(errorVecGnssMapY);
    circle.ccwFast3.absErrorStdGnssMapXY =...
        sqrt(circle.ccwFast3.absErrorStdGnssMapX^2+...
        circle.ccwFast3.absErrorStdGnssMapY^2);
    syncEstX =...
        circle.ccwFast3.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwFast3.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwFast3.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwFast3.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    circle.ccwFast3.absErrorMeanEstMapX = mean(errorVecEstMapX);
    circle.ccwFast3.absErrorMeanEstMapY = mean(errorVecEstMapY);
    circle.ccwFast3.absErrorMeanEstMapXY =...
        sqrt(circle.ccwFast3.absErrorMeanEstMapX^2+...
        circle.ccwFast3.absErrorMeanEstMapY^2);
    circle.ccwFast3.absErrorStdEstMapX = std(errorVecEstMapX);
    circle.ccwFast3.absErrorStdEstMapY = std(errorVecEstMapY);
    circle.ccwFast3.absErrorStdEstMapXY =...
        sqrt(circle.ccwFast3.absErrorStdEstMapX^2+...
        circle.ccwFast3.absErrorStdEstMapY^2);
    circle.ccwFast3.gnssToEstMeanErrorChange =...
        (circle.ccwFast3.absErrorMeanEstMapXY-...
        circle.ccwFast3.absErrorMeanGnssMapXY)/...
        circle.ccwFast3.absErrorMeanGnssMapXY;
    circle.ccwFast3.gnssToEstStdErrorChange =...
        (circle.ccwFast3.absErrorStdEstMapXY-...
        circle.ccwFast3.absErrorStdGnssMapXY)/...
        circle.ccwFast3.absErrorStdGnssMapXY;

    % CCW Slow 1 pose accuracy
    circle.ccwSlow1.groundTruthTimetable =...
        array2timetable(circle.ccwSlow1.groundTruthData,...
        'RowTimes',circle.ccwSlow1.groundTruthDatetime);        % create timetable for
groundTruth
    circle.ccwSlow1.gnssMapPoseTimetable =...
        array2timetable(circle.ccwSlow1.gnssMapPoseData,...
        'RowTimes',circle.ccwSlow1.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwSlow1.estMapPoseTimetable =...
        array2timetable(circle.ccwSlow1.estMapPoseData,...
        'RowTimes',circle.ccwSlow1.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwSlow1.groundTruthGnssMapErrorTimetable =...
        synchronize(circle.ccwSlow1.gnssMapPoseTimetable,...
        circle.ccwSlow1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    circle.ccwSlow1.groundTruthEstMapErrorTimetable =...
        synchronize(circle.ccwSlow1.estMapPoseTimetable,...
        circle.ccwSlow1.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        circle.ccwSlow1.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwSlow1.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        circle.ccwSlow1.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwSlow1.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    circle.ccwSlow1.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    circle.ccwSlow1.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    circle.ccwSlow1.absErrorMeanGnssMapXY =...
        sqrt(circle.ccwSlow1.absErrorMeanGnssMapX^2+...
        circle.ccwSlow1.absErrorMeanGnssMapY^2);
    circle.ccwSlow1.absErrorStdGnssMapX = std(errorVecGnssMapX);
    circle.ccwSlow1.absErrorStdGnssMapY = std(errorVecGnssMapY);
    circle.ccwSlow1.absErrorStdGnssMapXY =...
        sqrt(circle.ccwSlow1.absErrorStdGnssMapX^2+...
        circle.ccwSlow1.absErrorStdGnssMapY^2);
    syncEstX =...
        circle.ccwSlow1.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwSlow1.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwSlow1.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwSlow1.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    circle.ccwSlow1.absErrorMeanEstMapX = mean(errorVecEstMapX);
    circle.ccwSlow1.absErrorMeanEstMapY = mean(errorVecEstMapY);
    circle.ccwSlow1.absErrorMeanEstMapXY =...
        sqrt(circle.ccwSlow1.absErrorMeanEstMapX^2+...
        circle.ccwSlow1.absErrorMeanEstMapY^2);
    circle.ccwSlow1.absErrorStdEstMapX = std(errorVecEstMapX);
    circle.ccwSlow1.absErrorStdEstMapY = std(errorVecEstMapY);
    circle.ccwSlow1.absErrorStdEstMapXY =...
        sqrt(circle.ccwSlow1.absErrorStdEstMapX^2+...
        circle.ccwSlow1.absErrorStdEstMapY^2);
    circle.ccwSlow1.gnssToEstMeanErrorChange =...
        (circle.ccwSlow1.absErrorMeanEstMapXY-...
        circle.ccwSlow1.absErrorMeanGnssMapXY)/...
        circle.ccwSlow1.absErrorMeanGnssMapXY;
    circle.ccwSlow1.gnssToEstStdErrorChange =...
        (circle.ccwSlow1.absErrorStdEstMapXY-...
        circle.ccwSlow1.absErrorStdGnssMapXY)/...
        circle.ccwSlow1.absErrorStdGnssMapXY;

    % CCW Slow 2 pose accuracy
    circle.ccwSlow2.groundTruthTimetable =...
        array2timetable(circle.ccwSlow2.groundTruthData,...
        'RowTimes',circle.ccwSlow2.groundTruthDatetime);        % create timetable for
groundTruth
    circle.ccwSlow2.gnssMapPoseTimetable =...
        array2timetable(circle.ccwSlow2.gnssMapPoseData,...
        'RowTimes',circle.ccwSlow2.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwSlow2.estMapPoseTimetable =...
        array2timetable(circle.ccwSlow2.estMapPoseData,...
        'RowTimes',circle.ccwSlow2.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwSlow2.groundTruthGnssMapErrorTimetable =...
        synchronize(circle.ccwSlow2.gnssMapPoseTimetable,...
        circle.ccwSlow2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    circle.ccwSlow2.groundTruthEstMapErrorTimetable =...
        synchronize(circle.ccwSlow2.estMapPoseTimetable,...
        circle.ccwSlow2.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        circle.ccwSlow2.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwSlow2.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        circle.ccwSlow2.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwSlow2.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    circle.ccwSlow2.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    circle.ccwSlow2.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    circle.ccwSlow2.absErrorMeanGnssMapXY =...
        sqrt(circle.ccwSlow2.absErrorMeanGnssMapX^2+...
        circle.ccwSlow2.absErrorMeanGnssMapY^2);
    circle.ccwSlow2.absErrorStdGnssMapX = std(errorVecGnssMapX);
    circle.ccwSlow2.absErrorStdGnssMapY = std(errorVecGnssMapY);
    circle.ccwSlow2.absErrorStdGnssMapXY =...
        sqrt(circle.ccwSlow2.absErrorStdGnssMapX^2+...
        circle.ccwSlow2.absErrorStdGnssMapY^2);
    syncEstX =...
        circle.ccwSlow2.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwSlow2.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwSlow2.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwSlow2.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    circle.ccwSlow2.absErrorMeanEstMapX = mean(errorVecEstMapX);
    circle.ccwSlow2.absErrorMeanEstMapY = mean(errorVecEstMapY);
    circle.ccwSlow2.absErrorMeanEstMapXY =...
        sqrt(circle.ccwSlow2.absErrorMeanEstMapX^2+...
        circle.ccwSlow2.absErrorMeanEstMapY^2);
    circle.ccwSlow2.absErrorStdEstMapX = std(errorVecEstMapX);
    circle.ccwSlow2.absErrorStdEstMapY = std(errorVecEstMapY);
    circle.ccwSlow2.absErrorStdEstMapXY =...
        sqrt(circle.ccwSlow2.absErrorStdEstMapX^2+...
        circle.ccwSlow2.absErrorStdEstMapY^2);
    circle.ccwSlow2.gnssToEstMeanErrorChange =...
        (circle.ccwSlow2.absErrorMeanEstMapXY-...
        circle.ccwSlow2.absErrorMeanGnssMapXY)/...
        circle.ccwSlow2.absErrorMeanGnssMapXY;
    circle.ccwSlow2.gnssToEstStdErrorChange =...
        (circle.ccwSlow2.absErrorStdEstMapXY-...
        circle.ccwSlow2.absErrorStdGnssMapXY)/...
        circle.ccwSlow2.absErrorStdGnssMapXY;

    % CCW Slow 3 pose accuracy
    circle.ccwSlow3.groundTruthTimetable =...
        array2timetable(circle.ccwSlow3.groundTruthData,...
        'RowTimes',circle.ccwSlow3.groundTruthDatetime);        % create timetable for
groundTruth
    circle.ccwSlow3.gnssMapPoseTimetable =...
        array2timetable(circle.ccwSlow3.gnssMapPoseData,...
        'RowTimes',circle.ccwSlow3.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwSlow3.estMapPoseTimetable =...
        array2timetable(circle.ccwSlow3.estMapPoseData,...
        'RowTimes',circle.ccwSlow3.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwSlow3.groundTruthGnssMapErrorTimetable =...
        synchronize(circle.ccwSlow3.gnssMapPoseTimetable,...
        circle.ccwSlow3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    circle.ccwSlow3.groundTruthEstMapErrorTimetable =...
        synchronize(circle.ccwSlow3.estMapPoseTimetable,...
        circle.ccwSlow3.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        circle.ccwSlow3.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwSlow3.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        circle.ccwSlow3.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwSlow3.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    circle.ccwSlow3.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    circle.ccwSlow3.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    circle.ccwSlow3.absErrorMeanGnssMapXY =...
        sqrt(circle.ccwSlow3.absErrorMeanGnssMapX^2+...
        circle.ccwSlow3.absErrorMeanGnssMapY^2);
    circle.ccwSlow3.absErrorStdGnssMapX = std(errorVecGnssMapX);
    circle.ccwSlow3.absErrorStdGnssMapY = std(errorVecGnssMapY);
    circle.ccwSlow3.absErrorStdGnssMapXY =...
        sqrt(circle.ccwSlow3.absErrorStdGnssMapX^2+...
        circle.ccwSlow3.absErrorStdGnssMapY^2);
    syncEstX =...
        circle.ccwSlow3.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwSlow3.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwSlow3.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwSlow3.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    circle.ccwSlow3.absErrorMeanEstMapX = mean(errorVecEstMapX);
    circle.ccwSlow3.absErrorMeanEstMapY = mean(errorVecEstMapY);
    circle.ccwSlow3.absErrorMeanEstMapXY =...
        sqrt(circle.ccwSlow3.absErrorMeanEstMapX^2+...
        circle.ccwSlow3.absErrorMeanEstMapY^2);
    circle.ccwSlow3.absErrorStdEstMapX = std(errorVecEstMapX);
    circle.ccwSlow3.absErrorStdEstMapY = std(errorVecEstMapY);
    circle.ccwSlow3.absErrorStdEstMapXY =...
        sqrt(circle.ccwSlow3.absErrorStdEstMapX^2+...
        circle.ccwSlow3.absErrorStdEstMapY^2);
    circle.ccwSlow3.gnssToEstMeanErrorChange =...
        (circle.ccwSlow3.absErrorMeanEstMapXY-...
        circle.ccwSlow3.absErrorMeanGnssMapXY)/...
        circle.ccwSlow3.absErrorMeanGnssMapXY;
    circle.ccwSlow3.gnssToEstStdErrorChange =...
        (circle.ccwSlow3.absErrorStdEstMapXY-...
        circle.ccwSlow3.absErrorStdGnssMapXY)/...
        circle.ccwSlow3.absErrorStdGnssMapXY;

    % CCW Slow 4 pose accuracy
    circle.ccwSlow4.groundTruthTimetable =...
        array2timetable(circle.ccwSlow4.groundTruthData,...
        'RowTimes',circle.ccwSlow4.groundTruthDatetime);        % create timetable for
groundTruth
    circle.ccwSlow4.gnssMapPoseTimetable =...
        array2timetable(circle.ccwSlow4.gnssMapPoseData,...
        'RowTimes',circle.ccwSlow4.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwSlow4.estMapPoseTimetable =...
        array2timetable(circle.ccwSlow4.estMapPoseData,...
        'RowTimes',circle.ccwSlow4.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwSlow4.groundTruthGnssMapErrorTimetable =...
        synchronize(circle.ccwSlow4.gnssMapPoseTimetable,...
        circle.ccwSlow4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
gnssMapPose
    circle.ccwSlow4.groundTruthEstMapErrorTimetable =...
        synchronize(circle.ccwSlow4.estMapPoseTimetable,...
        circle.ccwSlow4.groundTruthTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncGnssX =...
        circle.ccwSlow4.groundTruthGnssMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwSlow4.groundTruthGnssMapErrorTimetable.Var1_2;
    syncGnssY =...
        circle.ccwSlow4.groundTruthGnssMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwSlow4.groundTruthGnssMapErrorTimetable.Var2_2;
    errorVecGnssMapX = abs(syncGnssX-syncGroundTruthX);
    errorVecGnssMapY = abs(syncGnssY-syncGroundTruthY);
    circle.ccwSlow4.absErrorMeanGnssMapX = mean(errorVecGnssMapX);
    circle.ccwSlow4.absErrorMeanGnssMapY = mean(errorVecGnssMapY);
    circle.ccwSlow4.absErrorMeanGnssMapXY =...
        sqrt(circle.ccwSlow4.absErrorMeanGnssMapX^2+...
        circle.ccwSlow4.absErrorMeanGnssMapY^2);
    circle.ccwSlow4.absErrorStdGnssMapX = std(errorVecGnssMapX);
    circle.ccwSlow4.absErrorStdGnssMapY = std(errorVecGnssMapY);
    circle.ccwSlow4.absErrorStdGnssMapXY =...
        sqrt(circle.ccwSlow4.absErrorStdGnssMapX^2+...
        circle.ccwSlow4.absErrorStdGnssMapY^2);
    syncEstX =...
        circle.ccwSlow4.groundTruthEstMapErrorTimetable.Var1_1;
    syncGroundTruthX =...
        circle.ccwSlow4.groundTruthEstMapErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwSlow4.groundTruthEstMapErrorTimetable.Var2_1;
    syncGroundTruthY =...
        circle.ccwSlow4.groundTruthEstMapErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGroundTruthX);
    errorVecEstMapY = abs(syncEstY-syncGroundTruthY);
    circle.ccwSlow4.absErrorMeanEstMapX = mean(errorVecEstMapX);
    circle.ccwSlow4.absErrorMeanEstMapY = mean(errorVecEstMapY);
    circle.ccwSlow4.absErrorMeanEstMapXY =...
        sqrt(circle.ccwSlow4.absErrorMeanEstMapX^2+...
        circle.ccwSlow4.absErrorMeanEstMapY^2);
    circle.ccwSlow4.absErrorStdEstMapX = std(errorVecEstMapX);
    circle.ccwSlow4.absErrorStdEstMapY = std(errorVecEstMapY);
    circle.ccwSlow4.absErrorStdEstMapXY =...
        sqrt(circle.ccwSlow4.absErrorStdEstMapX^2+...
        circle.ccwSlow4.absErrorStdEstMapY^2);
    circle.ccwSlow4.gnssToEstMeanErrorChange =...
        (circle.ccwSlow4.absErrorMeanEstMapXY-...
        circle.ccwSlow4.absErrorMeanGnssMapXY)/...
        circle.ccwSlow4.absErrorMeanGnssMapXY;
    circle.ccwSlow4.gnssToEstStdErrorChange =...
        (circle.ccwSlow4.absErrorStdEstMapXY-...
        circle.ccwSlow4.absErrorStdGnssMapXY)/...
        circle.ccwSlow4.absErrorStdGnssMapXY;

end

function circle = calculateInterpolationError(circle)

    % CCW Fast 1 interpolation accuracy
    circle.ccwFast1.gnssMapPoseTimetable =...
        array2timetable(circle.ccwFast1.gnssMapPoseData,...
        'RowTimes',circle.ccwFast1.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwFast1.estMapPoseTimetable =...
        array2timetable(circle.ccwFast1.estMapPoseData,...
        'RowTimes',circle.ccwFast1.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwFast1.gnssEstInterpErrorTimetable =...
        synchronize(circle.ccwFast1.estMapPoseTimetable,...
        circle.ccwFast1.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        circle.ccwFast1.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        circle.ccwFast1.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwFast1.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        circle.ccwFast1.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    circle.ccwFast1.interpErrorMeanX = mean(errorVecEstMapX);
    circle.ccwFast1.interpErrorMeanY = mean(errorVecEstMapY);
    circle.ccwFast1.interpErrorStdX = std(errorVecEstMapX);
    circle.ccwFast1.interpErrorStdY = std(errorVecEstMapY);
    circle.ccwFast1.interpErrorMeanXY =...
        sqrt(circle.ccwFast1.interpErrorMeanX^2+...
        circle.ccwFast1.interpErrorMeanY^2);
    circle.ccwFast1.interpErrorStdXY =...
        sqrt(circle.ccwFast1.interpErrorStdX^2+...
        circle.ccwFast1.interpErrorStdY^2);
    % CCW Fast 2 interpolation accuracy
    circle.ccwFast2.gnssMapPoseTimetable =...
        array2timetable(circle.ccwFast2.gnssMapPoseData,...
        'RowTimes',circle.ccwFast2.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwFast2.estMapPoseTimetable =...
        array2timetable(circle.ccwFast2.estMapPoseData,...
        'RowTimes',circle.ccwFast2.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwFast2.gnssEstInterpErrorTimetable =...
        synchronize(circle.ccwFast2.estMapPoseTimetable,...
        circle.ccwFast2.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        circle.ccwFast2.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        circle.ccwFast2.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwFast2.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        circle.ccwFast2.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    circle.ccwFast2.interpErrorMeanX = mean(errorVecEstMapX);
    circle.ccwFast2.interpErrorMeanY = mean(errorVecEstMapY);
    circle.ccwFast2.interpErrorStdX = std(errorVecEstMapX);
    circle.ccwFast2.interpErrorStdY = std(errorVecEstMapY);
    circle.ccwFast2.interpErrorMeanXY =...
        sqrt(circle.ccwFast2.interpErrorMeanX^2+...
        circle.ccwFast2.interpErrorMeanY^2);
    circle.ccwFast2.interpErrorStdXY =...
        sqrt(circle.ccwFast2.interpErrorStdX^2+...
        circle.ccwFast2.interpErrorStdY^2);

    % CCW Fast 3 pose accuracy
    circle.ccwFast3.gnssMapPoseTimetable =...
        array2timetable(circle.ccwFast3.gnssMapPoseData,...
        'RowTimes',circle.ccwFast3.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwFast3.estMapPoseTimetable =...
        array2timetable(circle.ccwFast3.estMapPoseData,...
        'RowTimes',circle.ccwFast3.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwFast3.gnssEstInterpErrorTimetable =...
        synchronize(circle.ccwFast3.estMapPoseTimetable,...
        circle.ccwFast3.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        circle.ccwFast3.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        circle.ccwFast3.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwFast3.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        circle.ccwFast3.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    circle.ccwFast3.interpErrorMeanX = mean(errorVecEstMapX);
    circle.ccwFast3.interpErrorMeanY = mean(errorVecEstMapY);
    circle.ccwFast3.interpErrorStdX = std(errorVecEstMapX);
    circle.ccwFast3.interpErrorStdY = std(errorVecEstMapY);
    circle.ccwFast3.interpErrorMeanXY =...
        sqrt(circle.ccwFast3.interpErrorMeanX^2+...
        circle.ccwFast3.interpErrorMeanY^2);
    circle.ccwFast3.interpErrorStdXY =...
        sqrt(circle.ccwFast3.interpErrorStdX^2+...
        circle.ccwFast3.interpErrorStdY^2);

    % CCW Slow 1 interpolation accuracy
    circle.ccwSlow1.gnssMapPoseTimetable =...
        array2timetable(circle.ccwSlow1.gnssMapPoseData,...
        'RowTimes',circle.ccwSlow1.gnssMapPoseDatetime);        % create timetable for 
gnssMapPose
    circle.ccwSlow1.estMapPoseTimetable =...
        array2timetable(circle.ccwSlow1.estMapPoseData,...
        'RowTimes',circle.ccwSlow1.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwSlow1.gnssEstInterpErrorTimetable =...
        synchronize(circle.ccwSlow1.estMapPoseTimetable,...
        circle.ccwSlow1.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and 
estMapPose
    syncEstX =...
        circle.ccwSlow1.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        circle.ccwSlow1.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwSlow1.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        circle.ccwSlow1.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    circle.ccwSlow1.interpErrorMeanX = mean(errorVecEstMapX);
    circle.ccwSlow1.interpErrorMeanY = mean(errorVecEstMapY);
    circle.ccwSlow1.interpErrorStdX = std(errorVecEstMapX);
    circle.ccwSlow1.interpErrorStdY = std(errorVecEstMapY);
    circle.ccwSlow1.interpErrorMeanXY =...
        sqrt(circle.ccwSlow1.interpErrorMeanX^2+...
        circle.ccwSlow1.interpErrorMeanY^2);
    circle.ccwSlow1.interpErrorStdXY =...
        sqrt(circle.ccwSlow1.interpErrorStdX^2+...
        circle.ccwSlow1.interpErrorStdY^2);

    % CCW Slow 2 interpolation accuracy
    circle.ccwSlow2.gnssMapPoseTimetable =...
        array2timetable(circle.ccwSlow2.gnssMapPoseData,...
        'RowTimes',circle.ccwSlow2.gnssMapPoseDatetime);        % create timetable for 
gnssMapPose
    circle.ccwSlow2.estMapPoseTimetable =...
        array2timetable(circle.ccwSlow2.estMapPoseData,...
        'RowTimes',circle.ccwSlow2.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwSlow2.gnssEstInterpErrorTimetable =...
        synchronize(circle.ccwSlow2.estMapPoseTimetable,...
        circle.ccwSlow2.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and 
estMapPose
    syncEstX =...
        circle.ccwSlow2.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        circle.ccwSlow2.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwSlow2.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        circle.ccwSlow2.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    circle.ccwSlow2.interpErrorMeanX = mean(errorVecEstMapX);
    circle.ccwSlow2.interpErrorMeanY = mean(errorVecEstMapY);
    circle.ccwSlow2.interpErrorStdX = std(errorVecEstMapX);
    circle.ccwSlow2.interpErrorStdY = std(errorVecEstMapY);
    circle.ccwSlow2.interpErrorMeanXY =...
        sqrt(circle.ccwSlow2.interpErrorMeanX^2+...
        circle.ccwSlow2.interpErrorMeanY^2);
    circle.ccwSlow2.interpErrorStdXY =...
        sqrt(circle.ccwSlow2.interpErrorStdX^2+...
        circle.ccwSlow2.interpErrorStdY^2);

    % CCW Slow 3 pose accuracy
    circle.ccwSlow3.gnssMapPoseTimetable =...
        array2timetable(circle.ccwSlow3.gnssMapPoseData,...
        'RowTimes',circle.ccwSlow3.gnssMapPoseDatetime);        % create timetable for 
gnssMapPose
    circle.ccwSlow3.estMapPoseTimetable =...
        array2timetable(circle.ccwSlow3.estMapPoseData,...
        'RowTimes',circle.ccwSlow3.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwSlow3.gnssEstInterpErrorTimetable =...
        synchronize(circle.ccwSlow3.estMapPoseTimetable,...
        circle.ccwSlow3.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        circle.ccwSlow3.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        circle.ccwSlow3.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwSlow3.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        circle.ccwSlow3.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    circle.ccwSlow3.interpErrorMeanX = mean(errorVecEstMapX);
    circle.ccwSlow3.interpErrorMeanY = mean(errorVecEstMapY);
    circle.ccwSlow3.interpErrorStdX = std(errorVecEstMapX);
    circle.ccwSlow3.interpErrorStdY = std(errorVecEstMapY);
    circle.ccwSlow3.interpErrorMeanXY =...
        sqrt(circle.ccwSlow3.interpErrorMeanX^2+...
        circle.ccwSlow3.interpErrorMeanY^2);
    circle.ccwSlow3.interpErrorStdXY =...
        sqrt(circle.ccwSlow3.interpErrorStdX^2+...
        circle.ccwSlow3.interpErrorStdY^2);

    % CCW Slow 4 pose accuracy
    circle.ccwSlow4.gnssMapPoseTimetable =...
        array2timetable(circle.ccwSlow4.gnssMapPoseData,...
        'RowTimes',circle.ccwSlow4.gnssMapPoseDatetime);        % create timetable for
gnssMapPose
    circle.ccwSlow4.estMapPoseTimetable =...
        array2timetable(circle.ccwSlow4.estMapPoseData,...
        'RowTimes',circle.ccwSlow4.estMapPoseDatetime);         % create timetable for estMapPose
    circle.ccwSlow4.gnssEstInterpErrorTimetable =...
        synchronize(circle.ccwSlow4.estMapPoseTimetable,...
        circle.ccwSlow4.gnssMapPoseTimetable,'first','linear'); % synchronize groundTruth and
estMapPose
    syncEstX =...
        circle.ccwSlow4.gnssEstInterpErrorTimetable.Var1_1;
    syncGnssX =...
        circle.ccwSlow4.gnssEstInterpErrorTimetable.Var1_2;
    syncEstY =...
        circle.ccwSlow4.gnssEstInterpErrorTimetable.Var2_1;
    syncGnssY =...
        circle.ccwSlow4.gnssEstInterpErrorTimetable.Var2_2;
    errorVecEstMapX = abs(syncEstX-syncGnssX);
    errorVecEstMapY = abs(syncEstY-syncGnssY);
    circle.ccwSlow4.interpErrorMeanX = mean(errorVecEstMapX);
    circle.ccwSlow4.interpErrorMeanY = mean(errorVecEstMapY);
    circle.ccwSlow4.interpErrorStdX = std(errorVecEstMapX);
    circle.ccwSlow4.interpErrorStdY = std(errorVecEstMapY);
    circle.ccwSlow4.interpErrorMeanXY =...
        sqrt(circle.ccwSlow4.interpErrorMeanX^2+...
        circle.ccwSlow4.interpErrorMeanY^2);
    circle.ccwSlow4.interpErrorStdXY =...
        sqrt(circle.ccwSlow4.interpErrorStdX^2+...
        circle.ccwSlow4.interpErrorStdY^2);

    % Save to file
    save('circleCorrectedWithCircleAndError.mat','circle');

end

function plotAllData(circle)

    % Load structure from file
    load('circleCorrectedWithCircle.mat');

    % CCW Slow 1 (CCW Slow Trial 1 in paper)
    plotTrial(circle.ccwSlow1,1,...

'$CCW\hspace{1mm}Circle\hspace{1mm}Slow\hspace{1mm}Trial\hspace{1mm}1\hspace{1mm}(\bar{v}\approx1
.2\hspace{1mm}[m/s])$');

    % CCW Slow 3 (CCW Slow Trial 2 in paper)
    plotTrial(circle.ccwSlow3,4,...

'$CCW\hspace{1mm}Circle\hspace{1mm}Slow\hspace{1mm}Trial\hspace{1mm}2\hspace{1mm}(\bar{v}\approx1
.2\hspace{1mm}[m/s])$');

    % CCW Slow 4 (CCW Slow Trial 3 in paper)
    plotTrial(circle.ccwSlow4,7,...

'$CCW\hspace{1mm}Circle\hspace{1mm}Slow\hspace{1mm}Trial\hspace{1mm}3\hspace{1mm}(\bar{v}\approx1
.3\hspace{1mm}[m/s])$');

    % CCW Fast 1 (CCW Fast Trial 1 in paper)
    plotTrial(circle.ccwFast1,10,...

'$CCW\hspace{1mm}Circle\hspace{1mm}Fast\hspace{1mm}Trial\hspace{1mm}1\hspace{1mm}(\bar{v}\approx2
.6\hspace{1mm}[m/s])$');

    % CCW Fast 2 (CCW Fast Trial 2 in paper)
    plotTrial(circle.ccwFast2,13,...

'$CCW\hspace{1mm}Circle\hspace{1mm}Fast\hspace{1mm}Trial\hspace{1mm}2\hspace{1mm}(\bar{v}\approx2
.6\hspace{1mm}[m/s])$');

    % CCW Fast 3 (CCW Fast Trial 3 in paper)
    plotTrial(circle.ccwFast3,16,...

'$CCW\hspace{1mm}Circle\hspace{1mm}Fast\hspace{1mm}Trial\hspace{1mm}3\hspace{1mm}(\bar{v}\approx2
.6\hspace{1mm}[m/s])$');

end

function plotTrial(circleTrial,figNumStart,trialTitleString)

    % Set font size
    titleFontSize = 32;
    defaultFontSize = 28;
    markerSize = 10;

    % Plot
    figure(figNumStart);
    clf;
    hold on;
    plot(circleTrial.groundTruthData(:,1),...
        circleTrial.groundTruthData(:,2),...
        'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(circleTrial.gnssMapPoseData(:,1),...
        circleTrial.gnssMapPoseData(:,2),...
        'r','LineStyle','-','Marker','o','LineWidth',1,'MarkerSize',markerSize);
    plot(circleTrial.estMapPoseData(:,1),...
        circleTrial.estMapPoseData(:,2),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    axis equal;
    grid on;
    ylabel('$y_m\hspace{1mm}[m]$','Interpreter','latex');
    xlabel('$x_m\hspace{1mm}[m]$','Interpreter','latex');
    legend('$ground\hspace{1mm}truth$',...
        '$GNSS\hspace{1mm}measurement$',...
        '$estimator\hspace{1mm}output$',...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);
    title(trialTitleString,...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
```

```
set(gcf,'Position',[0 0 1920 1280]);

fontSizeMod = 4;
figure(figNumStart+1);
clf;
rx = subplot(3,3,1);
hold on;
plot(circleTrial.gnssRbtPoseDatetime,...
    circleTrial.gnssRbtPoseData(:,1),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/1.5);
plot(circleTrial.estRbtPoseDatetime,...
    circleTrial.estRbtPoseData(:,1),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$x_r\hspace{1mm}[m]$','Interpreter','latex');
xlabel('$time [s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Robot\hspace{1mm}Body\hspace{1mm}x_r\hspace{1mm}position$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
ry = subplot(3,3,4);
hold on;
plot(circleTrial.gnssRbtPoseDatetime,...
    circleTrial.gnssRbtPoseData(:,2),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/1.5);
plot(circleTrial.estRbtPoseDatetime,...
    circleTrial.estRbtPoseData(:,2),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$y_r\hspace{1mm}[m]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Robot\hspace{1mm}Body\hspace{1mm}y_r\hspace{1mm}position$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rp = subplot(3,3,7);
hold on;
plot(circleTrial.imuPoseDatetime,...
    circleTrial.imuPoseData(:,3),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/2);
plot(circleTrial.estRbtPoseDatetime,...
    circleTrial.estRbtPoseData(:,3),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\psi_r\hspace{1mm}[rad]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Robot\hspace{1mm}Body\hspace{1mm}\psi_r\hspace{1mm}position$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rdx = subplot(3,3,2);
hold on;
plot(circleTrial.estRbtTwistDatetime,...
    circleTrial.estRbtTwistData(:,2),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
plot(circleTrial.linVelAbsDatetime,...
    circleTrial.linVelAbsData(:,3),...
    'c','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\dot{x}_r\hspace{1mm}[m/s]$','Interpreter','latex');
xlabel('$time [s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
```

```
title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{x}_r\hspace{1mm}velocity$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rdy = subplot(3,3,5);
hold on;
plot(circleTrial.estRbtTwistDatetime,...
    circleTrial.estRbtTwistData(:,1),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
plot(circleTrial.linVelAbsDatetime,...
    circleTrial.linVelAbsData(:,3),...
    'c','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\dot{y}_r\hspace{1mm}[m/s]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{y}_r\hspace{1mm}velocity$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rdp = subplot(3,3,8);
hold on;
plot(circleTrial.imuTwistDatetime,...
    circleTrial.imuTwistData(:,3),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/2);
plot(circleTrial.estRbtTwistDatetime,...
    circleTrial.estRbtTwistData(:,3),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\dot{\psi}_r\hspace{1mm}[rad/s]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{\psi}_r\hspace{1mm}velocity$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rddx = subplot(3,3,3);
hold on;
plot(circleTrial.imuAccDatetime,...
    circleTrial.imuAccData(:,1),...
    'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
plot(circleTrial.estRbtAccDatetime,...
    circleTrial.estRbtAccData(:,1),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\ddot{x}_r\hspace{1mm}[m/s^2]$','Interpreter','latex');
xlabel('$time [s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Robot\hspace{1mm}Body\hspace{1mm}\ddot{x}_r\hspace{1mm}acceleration$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
rddy = subplot(3,3,6);
hold on;
plot(circleTrial.imuAccDatetime,...
    circleTrial.imuAccData(:,2),...
    'r','LineStyle','none','Marker','o','LineWidth',1,'MarkerSize',markerSize/2);
plot(circleTrial.estRbtAccDatetime,...
    circleTrial.estRbtAccData(:,2),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\ddot{y}_r\hspace{1mm}[m/s^2]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Robot\hspace{1mm}Body\hspace{1mm}\ddot{y}_r\hspace{1mm}acceleration$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
legend('$measurement$',...
    '$estimator\hspace{1mm}output$',...
    'Interpreter','latex');
linkaxes([rx,ry,rp,rdx,rdy,rdp,rddx,rddy],'x');
set(gcf,'Position',[0 0 1920 1280]);

clear rdx rdy rdp;
figure(figNumStart+2);
clf;
mx = subplot(3,3,1);
hold on;
plot(circleTrial.groundTruthDatetime,...
    circleTrial.groundTruthData(:,1),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(circleTrial.gnssMapPoseDatetime,...
    circleTrial.gnssMapPoseData(:,1),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/1.5);
plot(circleTrial.estMapPoseDatetime,...
    circleTrial.estMapPoseData(:,1),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$x_m\hspace{1mm}[m]$','Interpreter','latex');
xlabel('$time [s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Map\hspace{1mm}x_m\hspace{1mm}position$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
legend('$ground\hspace{1mm}truth$',...
    '$measurement$',...
    '$estimator\hspace{1mm}output$',...
    'Interpreter','latex');
my = subplot(3,3,4);
hold on;
plot(circleTrial.groundTruthDatetime,...
    circleTrial.groundTruthData(:,2),...
    'k','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
plot(circleTrial.gnssMapPoseDatetime,...
    circleTrial.gnssMapPoseData(:,2),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/1.5);
plot(circleTrial.estMapPoseDatetime,...
    circleTrial.estMapPoseData(:,2),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$y_m\hspace{1mm}[m]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Map\hspace{1mm}y_m\hspace{1mm}position$',...
    'FontSize',titleFontSize-fontSizeMod,...
    'Interpreter','latex');
mp = subplot(3,3,7);
hold on;
plot(circleTrial.imuPoseDatetime,...
    circleTrial.imuPoseData(:,3),...
    'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/2);
plot(circleTrial.estMapPoseDatetime,...
    circleTrial.estMapPoseData(:,3),...
    'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
hold off;
grid on;
ylabel('$\psi_m\hspace{1mm}[rad]$','Interpreter','latex');
xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
datetick('x','MM:ss','keeplimits','keepticks');
title('$Map\hspace{1mm}\psi_m\hspace{1mm}position$',...
    'FontSize',titleFontSize-fontSizeMod,...
```

```
        'Interpreter','latex');
    rdx = subplot(3,3,2);
    hold on;
    plot(circleTrial.estRbtTwistDatetime,...
        circleTrial.estRbtTwistData(:,2),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    plot(circleTrial.linVelAbsDatetime,...
        circleTrial.linVelAbsData(:,3),...
        'c','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{x}_r\hspace{1mm}[m/s]$','Interpreter','latex');
    xlabel('$time [s]$','Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    datetick('x','MM:ss','keeplimits','keepticks');
    title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{x}_r\hspace{1mm}velocity$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    rdy = subplot(3,3,5);
    hold on;
    plot(circleTrial.estRbtTwistDatetime,...
        circleTrial.estRbtTwistData(:,1),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    plot(circleTrial.linVelAbsDatetime,...
        circleTrial.linVelAbsData(:,3),...
        'c','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{y}_r\hspace{1mm}[m/s]$','Interpreter','latex');
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    datetick('x','MM:ss','keeplimits','keepticks')
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{y}_r\hspace{1mm}velocity$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    rdp = subplot(3,3,8);
    hold on;
    plot(circleTrial.imuTwistDatetime,...
        circleTrial.imuTwistData(:,3),...
        'r','LineStyle','none','Marker','o','LineWidth',2,'MarkerSize',markerSize/2);
    plot(circleTrial.estRbtTwistDatetime,...
        circleTrial.estRbtTwistData(:,3),...
        'b','LineStyle','-','Marker','.','LineWidth',1,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{\psi}_r\hspace{1mm}[rad/s]$','Interpreter','latex');
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize-fontSizeMod);
    datetick('x','MM:ss','keeplimits','keepticks');
    title('$Robot\hspace{1mm}Body\hspace{1mm}\dot{\psi}_r\hspace{1mm}velocity$',...
        'FontSize',titleFontSize-fontSizeMod,...
        'Interpreter','latex');
    linkaxes([mx,my,mp,rdx,rdy,rdp],'x');
    set(gcf,'Position',[0 0 1920 1280]);

end
```

8.5. Post-processing Matlab code for linear velocity step input .csv data

```
function processVelocityBags()

    % Import data runs, set argument to 0 to import from .mat (fast), set
    % to 1 to import from actual .csv (slow)
    vel = importSquareData(0);

    % Manual data repairs
    vel = manualDataRepairs(vel);

    % Zero map frame data
    vel = zeroMapData(vel);

    % Calculate velocity
    vel = velocityCalculations(vel);

    % Plot data
    plotAllVelData(vel);
%     plotVelData(vel);

    % Assign to workspace
    assignin('base','vel',vel);

end

function vel = importSquareData(csvImport)

    % Import from actual .csv files
    if (csvImport == 1)

        % Slow1
        vel.slow1.gnssMapPose = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-est-
gnss-map-pose.csv';
        vel.slow1.gnssRbtPose = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-est-
gnss-rbt-pose.csv';
        vel.slow1.imuPose = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-est-imu-
rbt-pose.csv';
        vel.slow1.imuTwist = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-est-imu-
rbt-twist.csv';
        vel.slow1.imuAcc = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-est-imu-
rbt-acc.csv';
        vel.slow1.estMapPose = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-est-
map-pose.csv';
        vel.slow1.estRbtPose = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-est-
rbt-pose.csv';
        vel.slow1.estRbtTwist = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-est-
rbt-twist.csv';
        vel.slow1.estRbtAcc = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-est-rbt-
acc.csv';
        vel.slow1.estRbtLinVelAbs = 'zone20VelocityTestSlow1/zone20VelocityTestSlow1_estimated-
est-rbt-linVelAbs.csv';
        vel.slow1 = importEstimatorBag(vel.slow1);

        % Slow2
        vel.slow2.gnssMapPose = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-est-
gnss-map-pose.csv';
        vel.slow2.gnssRbtPose = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-est-
gnss-rbt-pose.csv';
        vel.slow2.imuPose = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-est-imu-
rbt-pose.csv';
        vel.slow2.imuTwist = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-est-imu-
rbt-twist.csv';
        vel.slow2.imuAcc = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-est-imu-
rbt-acc.csv';
        vel.slow2.estMapPose = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-est-
map-pose.csv';
        vel.slow2.estRbtPose = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-est-
rbt-pose.csv';
        vel.slow2.estRbtTwist = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-est-
rbt-twist.csv';
        vel.slow2.estRbtAcc = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-est-rbt-
acc.csv';
        vel.slow2.estRbtLinVelAbs = 'zone20VelocityTestSlow2/zone20VelocityTestSlow2_estimated-
est-rbt-linVelAbs.csv';
        vel.slow2 = importEstimatorBag(vel.slow2);

        % Slow3
        vel.slow3.gnssMapPose = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-est-
gnss-map-pose.csv';
        vel.slow3.gnssRbtPose = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-est-
gnss-rbt-pose.csv';
        vel.slow3.imuPose = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-est-imu-
rbt-pose.csv';
        vel.slow3.imuTwist = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-est-imu-
rbt-twist.csv';
        vel.slow3.imuAcc = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-est-imu-
rbt-acc.csv';
        vel.slow3.estMapPose = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-est-
map-pose.csv';
        vel.slow3.estRbtPose = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-est-
rbt-pose.csv';
        vel.slow3.estRbtTwist = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-est-
rbt-twist.csv';
        vel.slow3.estRbtAcc = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-est-rbt-
acc.csv';
        vel.slow3.estRbtLinVelAbs = 'zone20VelocityTestSlow3/zone20VelocityTestSlow3_estimated-
est-rbt-linVelAbs.csv';
        vel.slow3 = importEstimatorBag(vel.slow3);

        % Slow4
        vel.slow4.gnssMapPose = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-est-
gnss-map-pose.csv';
        vel.slow4.gnssRbtPose = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-est-
gnss-rbt-pose.csv';
        vel.slow4.imuPose = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-est-imu-
rbt-pose.csv';
        vel.slow4.imuTwist = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-est-imu-
rbt-twist.csv';
        vel.slow4.imuAcc = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-est-imu-
rbt-acc.csv';
        vel.slow4.estMapPose = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-est-
map-pose.csv';
        vel.slow4.estRbtPose = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-est-
rbt-pose.csv';
        vel.slow4.estRbtTwist = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-est-
rbt-twist.csv';
        vel.slow4.estRbtAcc = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-est-rbt-
acc.csv';
        vel.slow4.estRbtLinVelAbs = 'zone20VelocityTestSlow4/zone20VelocityTestSlow4_estimated-
est-rbt-linVelAbs.csv';
        vel.slow4 = importEstimatorBag(vel.slow4);

        % Slow5
        vel.slow5.gnssMapPose = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-est-
gnss-map-pose.csv';
        vel.slow5.gnssRbtPose = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-est-
gnss-rbt-pose.csv';
        vel.slow5.imuPose = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-est-imu-
rbt-pose.csv';
        vel.slow5.imuTwist = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-est-imu-
rbt-twist.csv';
        vel.slow5.imuAcc = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-est-imu-
rbt-acc.csv';
        vel.slow5.estMapPose = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-est-
map-pose.csv';
        vel.slow5.estRbtPose = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-est-
rbt-pose.csv';
        vel.slow5.estRbtTwist = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-est-
rbt-twist.csv';
        vel.slow5.estRbtAcc = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-est-rbt-
acc.csv';
        vel.slow5.estRbtLinVelAbs = 'zone20VelocityTestSlow5/zone20VelocityTestSlow5_estimated-
est-rbt-linVelAbs.csv';
        vel.slow5 = importEstimatorBag(vel.slow5);

        % Medium1
        vel.medium1.gnssMapPose = 'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-
est-gnss-map-pose.csv';
        vel.medium1.gnssRbtPose = 'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-
est-gnss-rbt-pose.csv';
        vel.medium1.imuPose = 'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-est-
imu-rbt-pose.csv';
        vel.medium1.imuTwist = 'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-
est-imu-rbt-twist.csv';
        vel.medium1.imuAcc = 'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-est-
imu-rbt-acc.csv';
        vel.medium1.estMapPose = 'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-
est-map-pose.csv';
        vel.medium1.estRbtPose = 'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-
est-rbt-pose.csv';
        vel.medium1.estRbtTwist = 'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-
est-rbt-twist.csv';
        vel.medium1.estRbtAcc = 'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-
est-rbt-acc.csv';
        vel.medium1.estRbtLinVelAbs =
'zone20VelocityTestMedium1/zone20VelocityTestMedium1_estimated-est-rbt-linVelAbs.csv';
        vel.medium1 = importEstimatorBag(vel.medium1);

        % Medium2
        vel.medium2.gnssMapPose = 'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-
est-gnss-map-pose.csv';
        vel.medium2.gnssRbtPose = 'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-
est-gnss-rbt-pose.csv';
        vel.medium2.imuPose = 'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-est-
imu-rbt-pose.csv';
        vel.medium2.imuTwist = 'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-
est-imu-rbt-twist.csv';
        vel.medium2.imuAcc = 'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-est-
imu-rbt-acc.csv';
        vel.medium2.estMapPose = 'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-
est-map-pose.csv';
        vel.medium2.estRbtPose = 'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-
est-rbt-pose.csv';
        vel.medium2.estRbtTwist = 'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-
est-rbt-twist.csv';
        vel.medium2.estRbtAcc = 'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-
est-rbt-acc.csv';
        vel.medium2.estRbtLinVelAbs =
'zone20VelocityTestMedium2/zone20VelocityTestMedium2_estimated-est-rbt-linVelAbs.csv';
        vel.medium2 = importEstimatorBag(vel.medium2);

        % Medium3
        vel.medium3.gnssMapPose = 'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-
est-gnss-map-pose.csv';
        vel.medium3.gnssRbtPose = 'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-
est-gnss-rbt-pose.csv';
        vel.medium3.imuPose = 'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-est-
imu-rbt-pose.csv';
        vel.medium3.imuTwist = 'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-
est-imu-rbt-twist.csv';
        vel.medium3.imuAcc = 'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-est-
imu-rbt-acc.csv';
        vel.medium3.estMapPose = 'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-
est-map-pose.csv';
        vel.medium3.estRbtPose = 'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-
est-rbt-pose.csv';
        vel.medium3.estRbtTwist = 'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-
est-rbt-twist.csv';
        vel.medium3.estRbtAcc = 'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-
est-rbt-acc.csv';
        vel.medium3.estRbtLinVelAbs =
'zone20VelocityTestMedium3/zone20VelocityTestMedium3_estimated-est-rbt-linVelAbs.csv';
        vel.medium3 = importEstimatorBag(vel.medium3);

        % Medium4
        vel.medium4.gnssMapPose = 'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-
est-gnss-map-pose.csv';
        vel.medium4.gnssRbtPose = 'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-
est-gnss-rbt-pose.csv';
        vel.medium4.imuPose = 'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-est-
imu-rbt-pose.csv';
        vel.medium4.imuTwist = 'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-
est-imu-rbt-twist.csv';
        vel.medium4.imuAcc = 'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-est-
imu-rbt-acc.csv';
        vel.medium4.estMapPose = 'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-
est-map-pose.csv';
        vel.medium4.estRbtPose = 'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-
est-rbt-pose.csv';
        vel.medium4.estRbtTwist = 'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-
est-rbt-twist.csv';
        vel.medium4.estRbtAcc = 'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-
est-rbt-acc.csv';
        vel.medium4.estRbtLinVelAbs =
'zone20VelocityTestMedium4/zone20VelocityTestMedium4_estimated-est-rbt-linVelAbs.csv';
        vel.medium4 = importEstimatorBag(vel.medium4);

        % Medium5
        vel.medium5.gnssMapPose = 'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-
est-gnss-map-pose.csv';
        vel.medium5.gnssRbtPose = 'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-
est-gnss-rbt-pose.csv';
        vel.medium5.imuPose = 'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-est-
imu-rbt-pose.csv';
        vel.medium5.imuTwist = 'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-
est-imu-rbt-twist.csv';
        vel.medium5.imuAcc = 'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-est-
imu-rbt-acc.csv';
        vel.medium5.estMapPose = 'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-
est-map-pose.csv';
        vel.medium5.estRbtPose = 'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-
est-rbt-pose.csv';
        vel.medium5.estRbtTwist = 'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-
est-rbt-twist.csv';
        vel.medium5.estRbtAcc = 'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-
est-rbt-acc.csv';
        vel.medium5.estRbtLinVelAbs =
'zone20VelocityTestMedium5/zone20VelocityTestMedium5_estimated-est-rbt-linVelAbs.csv';
        vel.medium5 = importEstimatorBag(vel.medium5);

        % Fast1
        vel.fast1.gnssMapPose = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-est-
gnss-map-pose.csv';
        vel.fast1.gnssRbtPose = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-est-
gnss-rbt-pose.csv';
        vel.fast1.imuPose = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-est-imu-
rbt-pose.csv';
        vel.fast1.imuTwist = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-est-imu-
rbt-twist.csv';
        vel.fast1.imuAcc = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-est-imu-
rbt-acc.csv';
        vel.fast1.estMapPose = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-est-
map-pose.csv';
        vel.fast1.estRbtPose = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-est-
rbt-pose.csv';
        vel.fast1.estRbtTwist = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-est-
rbt-twist.csv';
        vel.fast1.estRbtAcc = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-est-rbt-
acc.csv';
        vel.fast1.estRbtLinVelAbs = 'zone20VelocityTestFast1/zone20VelocityTestFast1_estimated-
est-rbt-linVelAbs.csv';
        vel.fast1 = importEstimatorBag(vel.fast1);

        % Fast2
        vel.fast2.gnssMapPose = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-est-
gnss-map-pose.csv';
        vel.fast2.gnssRbtPose = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-est-
gnss-rbt-pose.csv';
        vel.fast2.imuPose = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-est-imu-
rbt-pose.csv';
        vel.fast2.imuTwist = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-est-imu-
rbt-twist.csv';
        vel.fast2.imuAcc = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-est-imu-
rbt-acc.csv';
        vel.fast2.estMapPose = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-est-
map-pose.csv';
        vel.fast2.estRbtPose = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-est-
rbt-pose.csv';
        vel.fast2.estRbtTwist = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-est-
rbt-twist.csv';
        vel.fast2.estRbtAcc = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-est-rbt-
acc.csv';
        vel.fast2.estRbtLinVelAbs = 'zone20VelocityTestFast2/zone20VelocityTestFast2_estimated-
est-rbt-linVelAbs.csv';
        vel.fast2 = importEstimatorBag(vel.fast2);

        % Fast3
        vel.fast3.gnssMapPose = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-est-
gnss-map-pose.csv';
        vel.fast3.gnssRbtPose = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-est-
gnss-rbt-pose.csv';
        vel.fast3.imuPose = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-est-imu-
rbt-pose.csv';
        vel.fast3.imuTwist = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-est-imu-
rbt-twist.csv';
        vel.fast3.imuAcc = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-est-imu-
rbt-acc.csv';
        vel.fast3.estMapPose = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-est-
map-pose.csv';
        vel.fast3.estRbtPose = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-est-
rbt-pose.csv';
        vel.fast3.estRbtTwist = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-est-
rbt-twist.csv';
        vel.fast3.estRbtAcc = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-est-rbt-
acc.csv';
        vel.fast3.estRbtLinVelAbs = 'zone20VelocityTestFast3/zone20VelocityTestFast3_estimated-
est-rbt-linVelAbs.csv';
        vel.fast3 = importEstimatorBag(vel.fast3);

        % Fast4
        vel.fast4.gnssMapPose = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-est-
gnss-map-pose.csv';
        vel.fast4.gnssRbtPose = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-est-
gnss-rbt-pose.csv';
        vel.fast4.imuPose = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-est-imu-
rbt-pose.csv';
        vel.fast4.imuTwist = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-est-imu-
rbt-twist.csv';
        vel.fast4.imuAcc = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-est-imu-
rbt-acc.csv';
        vel.fast4.estMapPose = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-est-
map-pose.csv';
        vel.fast4.estRbtPose = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-est-
rbt-pose.csv';
        vel.fast4.estRbtTwist = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-est-
rbt-twist.csv';
        vel.fast4.estRbtAcc = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-est-rbt-
acc.csv';
        vel.fast4.estRbtLinVelAbs = 'zone20VelocityTestFast4/zone20VelocityTestFast4_estimated-
est-rbt-linVelAbs.csv';
        vel.fast4 = importEstimatorBag(vel.fast4);

        % Fast5
        vel.fast5.gnssMapPose = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-est-
gnss-map-pose.csv';
        vel.fast5.gnssRbtPose = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-est-
gnss-rbt-pose.csv';
        vel.fast5.imuPose = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-est-imu-
rbt-pose.csv';
        vel.fast5.imuTwist = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-est-imu-
rbt-twist.csv';
        vel.fast5.imuAcc = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-est-imu-
rbt-acc.csv';
        vel.fast5.estMapPose = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-est-
map-pose.csv';
        vel.fast5.estRbtPose = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-est-
rbt-pose.csv';
        vel.fast5.estRbtTwist = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-est-
rbt-twist.csv';
        vel.fast5.estRbtAcc = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-est-rbt-
acc.csv';
        vel.fast5.estRbtLinVelAbs = 'zone20VelocityTestFast5/zone20VelocityTestFast5_estimated-
est-rbt-linVelAbs.csv';
        vel.fast5 = importEstimatorBag(vel.fast5);

        % Save to file
        save('vel.mat','vel');

    % Import from .mat file
    else

        load('vel.mat');

    end

end

function trial = importEstimatorBag(trial)

    % Sensor refresh rate for approximate time vectors
    gnssRR = 1;     % GNSS refresh rate in [Hz]
    imuRR = 20;     % IMU refresh rate in [Hz]
    estRR = 40;     % estimator refresh rate in [Hz]

    % GNSS map pose
    string = readmatrix(trial.gnssMapPose,...
        'Delimiter',',','OutputType','string');     % read in GNSS map pose as a string (for time
data)
    num = readmatrix(trial.gnssMapPose,...
        'Delimiter',',');                           % read in GNSS map pose as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/gnssRR:(row-2)*(1/gnssRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.gnssMapPoseDatetime = dt;
    trial.gnssMapPoseData = num(:,2:4);

    % GNSS robot pose
    string = readmatrix(trial.gnssRbtPose,...
        'Delimiter',',','OutputType','string');      % read in GNSS robot pose as a string (for
time data)
    num = readmatrix(trial.gnssRbtPose,...
        'Delimiter',',');                            % read in GNSS robot pose as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/gnssRR:(row-2)*(1/gnssRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.gnssRbtPoseDatetime = dt;
    trial.gnssRbtPoseData = num(:,2:4);

    % IMU pose
    string = readmatrix(trial.imuPose,...
        'Delimiter',',','OutputType','string');      % read in IMU pose as a string (for time
data)
    num = readmatrix(trial.imuPose,...
        'Delimiter',',');                            % read in IMU pose as doubles (for actual
data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/imuRR:(row-2)*(1/imuRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.imuPoseDatetime = dt;
    trial.imuPoseData = num(:,2:4);

    % IMU twist
    string = readmatrix(trial.imuTwist,...
        'Delimiter',',','OutputType','string');      % read in IMU twist as a string (for time
data)
    num = readmatrix(trial.imuTwist,...
        'Delimiter',',');                            % read in IMU twist as doubles (for actual
data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/imuRR:(row-2)*(1/imuRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.imuTwistDatetime = dt;
    trial.imuTwistData = num(:,2:4);

    % IMU acceleration
    string = readmatrix(trial.imuAcc,...
        'Delimiter',',','OutputType','string');      % read in IMU acceleration as a string (for
time data)
    num = readmatrix(trial.imuAcc,...
        'Delimiter',',');                            % read in IMU acceleration as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/imuRR:(row-2)*(1/imuRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.imuAccDatetime = dt;
    trial.imuAccData = num(:,2:4);

    % Estimator map pose
    string = readmatrix(trial.estMapPose,...
        'Delimiter',',','OutputType','string');      % read in estimator map pose as a string (for
time data)
    num = readmatrix(trial.estMapPose,...
        'Delimiter',',');                            % read in estimator map pose as doubles (for
actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estMapPoseDatetime = dt;
    trial.estMapPoseData = num(:,2:4);

    % Estimator robot pose
    string = readmatrix(trial.estRbtPose,...
        'Delimiter',',','OutputType','string');      % read in estimator robot pose as a string
(for time data)
    num = readmatrix(trial.estRbtPose,...
        'Delimiter',',');                            % read in estimator robot pose as doubles
(for actual data)
```

```
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estRbtPoseDatetime = dt;
    trial.estRbtPoseData = num(:,2:4);

    % Estimator robot twist
    string = readmatrix(trial.estRbtTwist,...
        'Delimiter',',','OutputType','string');     % read in estimator robot twist as a string
(for time data)
    num = readmatrix(trial.estRbtTwist,...
        'Delimiter',',');                           % read in estimator robot twist as doubles
(for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estRbtTwistDatetime = dt;
    trial.estRbtTwistData = num(:,2:4);

    % Estimator robot acceleration
    string = readmatrix(trial.estRbtAcc,...
        'Delimiter',',','OutputType','string');     % read in estimator robot acceleration as a
string (for time data)
    num = readmatrix(trial.estRbtAcc,...
        'Delimiter',',');                           % read in estimator robot acceleration as
doubles (for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.estRbtAccDatetime = dt;
    trial.estRbtAccData = num(:,2:4);

    % Estimator robot absolute linear velocity vector
    string = readmatrix(trial.estRbtLinVelAbs,...
        'Delimiter',',','OutputType','string');     % read in estimator robot absolute linear
velocity vector as a string (for time data)
    num = readmatrix(trial.estRbtLinVelAbs,...
        'Delimiter',',');                           % read in estimator robot absolute linear
velocity vector as doubles (for actual data)
    startTime = char(string(1,1));
    secondTime = char(string(2,1));
    startSec = str2num(startTime(1,18:26));
    secondSec = str2num(secondTime(1,18:26));
    [row,~] = size(num);
    t1 = secondSec-startSec;
    timeVec = (0:1/estRR:(row-2)*(1/estRR))'+t1;
    timeVec = [0;timeVec];
    Y = (ones(size(timeVec))*2019);
    M = (ones(size(timeVec))*1);
    D = (ones(size(timeVec))*1);
    H = (ones(size(timeVec))*0);
    MI = (ones(size(timeVec))*0);
    S = floor(timeVec);
    MS = (timeVec-floor(timeVec))*1E3;
    dt = datetime(Y,M,D,H,MI,S,MS,...
        'Format','yyyy-MM-dd HH:mm:ss.SSS');
    trial.linVelAbsDatetime = dt;
    trial.linVelAbsData = num(:,2:4);

end

function vel = manualDataRepairs(vel)

    % slow1
    gnss0 = 1;
    imu0 = 0;
    est0 = 270;
    estEndRem = 24;
    vel.slow1.gnssMapPoseDatetime(1:gnss0,:) = [];
    vel.slow1.gnssMapPoseData(1:gnss0,:) = [];
    vel.slow1.gnssRbtPoseDatetime(1:gnss0,:) = [];
    vel.slow1.gnssRbtPoseData(1:gnss0,:) = [];
    vel.slow1.imuPoseDatetime(1:imu0,:) = [];
    vel.slow1.imuPoseData(1:imu0,:) = [];
    vel.slow1.imuTwistDatetime(1:imu0,:) = [];
    vel.slow1.imuTwistData(1:imu0,:) = [];
    vel.slow1.imuAccDatetime(1:imu0,:) = [];
    vel.slow1.imuAccData(1:imu0,:) = [];
    vel.slow1.estMapPoseDatetime(1:est0,:) = [];
    vel.slow1.estMapPoseData(1:est0,:) = [];
    vel.slow1.estRbtPoseDatetime(1:est0,:) = [];
    vel.slow1.estRbtPoseData(1:est0,:) = [];
    vel.slow1.estRbtTwistDatetime(1:est0,:) = [];
    vel.slow1.estRbtTwistData(1:est0,:) = [];
    vel.slow1.estRbtAccDatetime(1:est0,:) = [];
    vel.slow1.estRbtAccData(1:est0,:) = [];
    vel.slow1.linVelAbsDatetime(1:est0,:) = [];
    vel.slow1.linVelAbsData(1:est0,:) = [];
    vel.slow1.estMapPoseDatetime(end-estEndRem:end,:) = [];
    vel.slow1.estMapPoseData(end-estEndRem:end,:) = [];
    vel.slow1.estRbtPoseDatetime(end-estEndRem:end,:) = [];
    vel.slow1.estRbtPoseData(end-estEndRem:end,:) = [];
    vel.slow1.estRbtTwistDatetime(end-estEndRem:end,:) = [];
    vel.slow1.estRbtTwistData(end-estEndRem:end,:) = [];
    vel.slow1.estRbtAccDatetime(end-estEndRem:end,:) = [];
    vel.slow1.estRbtAccData(end-estEndRem:end,:) = [];
    vel.slow1.linVelAbsDatetime(end-estEndRem:end,:) = [];
    vel.slow1.linVelAbsData(end-estEndRem:end,:) = [];
```

```
% slow2
gnss0 = 1;
imu0 = 0;
est0 = 183;
estEndRem = 29;
vel.slow2.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.slow2.gnssMapPoseData(1:gnss0,:) = [];
vel.slow2.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.slow2.gnssRbtPoseData(1:gnss0,:) = [];
vel.slow2.imuPoseDatetime(1:imu0,:) = [];
vel.slow2.imuPoseData(1:imu0,:) = [];
vel.slow2.imuTwistDatetime(1:imu0,:) = [];
vel.slow2.imuTwistData(1:imu0,:) = [];
vel.slow2.imuAccDatetime(1:imu0,:) = [];
vel.slow2.imuAccData(1:imu0,:) = [];
vel.slow2.estMapPoseDatetime(1:est0,:) = [];
vel.slow2.estMapPoseData(1:est0,:) = [];
vel.slow2.estRbtPoseDatetime(1:est0,:) = [];
vel.slow2.estRbtPoseData(1:est0,:) = [];
vel.slow2.estRbtTwistDatetime(1:est0,:) = [];
vel.slow2.estRbtTwistData(1:est0,:) = [];
vel.slow2.estRbtAccDatetime(1:est0,:) = [];
vel.slow2.estRbtAccData(1:est0,:) = [];
vel.slow2.linVelAbsDatetime(1:est0,:) = [];
vel.slow2.linVelAbsData(1:est0,:) = [];
vel.slow2.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.slow2.estMapPoseData(end-estEndRem:end,:) = [];
vel.slow2.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.slow2.estRbtPoseData(end-estEndRem:end,:) = [];
vel.slow2.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.slow2.estRbtTwistData(end-estEndRem:end,:) = [];
vel.slow2.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.slow2.estRbtAccData(end-estEndRem:end,:) = [];
vel.slow2.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.slow2.linVelAbsData(end-estEndRem:end,:) = [];

% slow3
gnss0 = 0;
imu0 = 0;
est0 = 195;
estEndRem = 28;
vel.slow3.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.slow3.gnssMapPoseData(1:gnss0,:) = [];
vel.slow3.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.slow3.gnssRbtPoseData(1:gnss0,:) = [];
vel.slow3.imuPoseDatetime(1:imu0,:) = [];
vel.slow3.imuPoseData(1:imu0,:) = [];
vel.slow3.imuTwistDatetime(1:imu0,:) = [];
vel.slow3.imuTwistData(1:imu0,:) = [];
vel.slow3.imuAccDatetime(1:imu0,:) = [];
vel.slow3.imuAccData(1:imu0,:) = [];
vel.slow3.estMapPoseDatetime(1:est0,:) = [];
vel.slow3.estMapPoseData(1:est0,:) = [];
vel.slow3.estRbtPoseDatetime(1:est0,:) = [];
vel.slow3.estRbtPoseData(1:est0,:) = [];
vel.slow3.estRbtTwistDatetime(1:est0,:) = [];
vel.slow3.estRbtTwistData(1:est0,:) = [];
vel.slow3.estRbtAccDatetime(1:est0,:) = [];
vel.slow3.estRbtAccData(1:est0,:) = [];
vel.slow3.linVelAbsDatetime(1:est0,:) = [];
vel.slow3.linVelAbsData(1:est0,:) = [];
vel.slow3.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.slow3.estMapPoseData(end-estEndRem:end,:) = [];
vel.slow3.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.slow3.estRbtPoseData(end-estEndRem:end,:) = [];
vel.slow3.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.slow3.estRbtTwistData(end-estEndRem:end,:) = [];
vel.slow3.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.slow3.estRbtAccData(end-estEndRem:end,:) = [];
vel.slow3.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.slow3.linVelAbsData(end-estEndRem:end,:) = [];

% slow4
gnss0 = 0;
imu0 = 0;
est0 = 246;
estEndRem = 25;
vel.slow4.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.slow4.gnssMapPoseData(1:gnss0,:) = [];
vel.slow4.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.slow4.gnssRbtPoseData(1:gnss0,:) = [];
vel.slow4.imuPoseDatetime(1:imu0,:) = [];
vel.slow4.imuPoseData(1:imu0,:) = [];
vel.slow4.imuTwistDatetime(1:imu0,:) = [];
vel.slow4.imuTwistData(1:imu0,:) = [];
vel.slow4.imuAccDatetime(1:imu0,:) = [];
vel.slow4.imuAccData(1:imu0,:) = [];
vel.slow4.estMapPoseDatetime(1:est0,:) = [];
vel.slow4.estMapPoseData(1:est0,:) = [];
vel.slow4.estRbtPoseDatetime(1:est0,:) = [];
vel.slow4.estRbtPoseData(1:est0,:) = [];
vel.slow4.estRbtTwistDatetime(1:est0,:) = [];
vel.slow4.estRbtTwistData(1:est0,:) = [];
vel.slow4.estRbtAccDatetime(1:est0,:) = [];
vel.slow4.estRbtAccData(1:est0,:) = [];
vel.slow4.linVelAbsDatetime(1:est0,:) = [];
vel.slow4.linVelAbsData(1:est0,:) = [];
vel.slow4.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.slow4.estMapPoseData(end-estEndRem:end,:) = [];
vel.slow4.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.slow4.estRbtPoseData(end-estEndRem:end,:) = [];
vel.slow4.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.slow4.estRbtTwistData(end-estEndRem:end,:) = [];
vel.slow4.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.slow4.estRbtAccData(end-estEndRem:end,:) = [];
vel.slow4.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.slow4.linVelAbsData(end-estEndRem:end,:) = [];

% slow5
gnss0 = 1;
imu0 = 0;
est0 = 260;
estEndRem = 24;
vel.slow5.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.slow5.gnssMapPoseData(1:gnss0,:) = [];
vel.slow5.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.slow5.gnssRbtPoseData(1:gnss0,:) = [];
vel.slow5.imuPoseDatetime(1:imu0,:) = [];
vel.slow5.imuPoseData(1:imu0,:) = [];
vel.slow5.imuTwistDatetime(1:imu0,:) = [];
vel.slow5.imuTwistData(1:imu0,:) = [];
vel.slow5.imuAccDatetime(1:imu0,:) = [];
vel.slow5.imuAccData(1:imu0,:) = [];
vel.slow5.estMapPoseDatetime(1:est0,:) = [];
vel.slow5.estMapPoseData(1:est0,:) = [];
vel.slow5.estRbtPoseDatetime(1:est0,:) = [];
vel.slow5.estRbtPoseData(1:est0,:) = [];
vel.slow5.estRbtTwistDatetime(1:est0,:) = [];
vel.slow5.estRbtTwistData(1:est0,:) = [];
vel.slow5.estRbtAccDatetime(1:est0,:) = [];
vel.slow5.estRbtAccData(1:est0,:) = [];
vel.slow5.linVelAbsDatetime(1:est0,:) = [];
vel.slow5.linVelAbsData(1:est0,:) = [];
vel.slow5.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.slow5.estMapPoseData(end-estEndRem:end,:) = [];
vel.slow5.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.slow5.estRbtPoseData(end-estEndRem:end,:) = [];
vel.slow5.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.slow5.estRbtTwistData(end-estEndRem:end,:) = [];
vel.slow5.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.slow5.estRbtAccData(end-estEndRem:end,:) = [];
vel.slow5.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.slow5.linVelAbsData(end-estEndRem:end,:) = [];

% medium1
gnss0 = 1;
imu0 = 0;
est0 = 205;
estEndRem = 58;
vel.medium1.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.medium1.gnssMapPoseData(1:gnss0,:) = [];
vel.medium1.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.medium1.gnssRbtPoseData(1:gnss0,:) = [];
vel.medium1.imuPoseDatetime(1:imu0,:) = [];
vel.medium1.imuPoseData(1:imu0,:) = [];
vel.medium1.imuTwistDatetime(1:imu0,:) = [];
vel.medium1.imuTwistData(1:imu0,:) = [];
vel.medium1.imuAccDatetime(1:imu0,:) = [];
vel.medium1.imuAccData(1:imu0,:) = [];
vel.medium1.estMapPoseDatetime(1:est0,:) = [];
vel.medium1.estMapPoseData(1:est0,:) = [];
vel.medium1.estRbtPoseDatetime(1:est0,:) = [];
vel.medium1.estRbtPoseData(1:est0,:) = [];
vel.medium1.estRbtTwistDatetime(1:est0,:) = [];
vel.medium1.estRbtTwistData(1:est0,:) = [];
vel.medium1.estRbtAccDatetime(1:est0,:) = [];
vel.medium1.estRbtAccData(1:est0,:) = [];
vel.medium1.linVelAbsDatetime(1:est0,:) = [];
vel.medium1.linVelAbsData(1:est0,:) = [];
vel.medium1.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.medium1.estMapPoseData(end-estEndRem:end,:) = [];
vel.medium1.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.medium1.estRbtPoseData(end-estEndRem:end,:) = [];
vel.medium1.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.medium1.estRbtTwistData(end-estEndRem:end,:) = [];
vel.medium1.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.medium1.estRbtAccData(end-estEndRem:end,:) = [];
vel.medium1.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.medium1.linVelAbsData(end-estEndRem:end,:) = [];

% medium2
gnss0 = 0;
imu0 = 0;
est0 = 235;
estEndRem = 31;
vel.medium2.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.medium2.gnssMapPoseData(1:gnss0,:) = [];
vel.medium2.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.medium2.gnssRbtPoseData(1:gnss0,:) = [];
vel.medium2.imuPoseDatetime(1:imu0,:) = [];
vel.medium2.imuPoseData(1:imu0,:) = [];
vel.medium2.imuTwistDatetime(1:imu0,:) = [];
vel.medium2.imuTwistData(1:imu0,:) = [];
vel.medium2.imuAccDatetime(1:imu0,:) = [];
vel.medium2.imuAccData(1:imu0,:) = [];
vel.medium2.estMapPoseDatetime(1:est0,:) = [];
vel.medium2.estMapPoseData(1:est0,:) = [];
vel.medium2.estRbtPoseDatetime(1:est0,:) = [];
vel.medium2.estRbtPoseData(1:est0,:) = [];
vel.medium2.estRbtTwistDatetime(1:est0,:) = [];
vel.medium2.estRbtTwistData(1:est0,:) = [];
vel.medium2.estRbtAccDatetime(1:est0,:) = [];
vel.medium2.estRbtAccData(1:est0,:) = [];
vel.medium2.linVelAbsDatetime(1:est0,:) = [];
vel.medium2.linVelAbsData(1:est0,:) = [];
vel.medium2.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.medium2.estMapPoseData(end-estEndRem:end,:) = [];
vel.medium2.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.medium2.estRbtPoseData(end-estEndRem:end,:) = [];
vel.medium2.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.medium2.estRbtTwistData(end-estEndRem:end,:) = [];
vel.medium2.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.medium2.estRbtAccData(end-estEndRem:end,:) = [];
vel.medium2.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.medium2.linVelAbsData(end-estEndRem:end,:) = [];

% medium3
gnss0 = 1;
imu0 = 0;
est0 = 205;
estEndRem = 49;
vel.medium3.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.medium3.gnssMapPoseData(1:gnss0,:) = [];
vel.medium3.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.medium3.gnssRbtPoseData(1:gnss0,:) = [];
vel.medium3.imuPoseDatetime(1:imu0,:) = [];
vel.medium3.imuPoseData(1:imu0,:) = [];
vel.medium3.imuTwistDatetime(1:imu0,:) = [];
vel.medium3.imuTwistData(1:imu0,:) = [];
vel.medium3.imuAccDatetime(1:imu0,:) = [];
vel.medium3.imuAccData(1:imu0,:) = [];
vel.medium3.estMapPoseDatetime(1:est0,:) = [];
vel.medium3.estMapPoseData(1:est0,:) = [];
vel.medium3.estRbtPoseDatetime(1:est0,:) = [];
vel.medium3.estRbtPoseData(1:est0,:) = [];
vel.medium3.estRbtTwistDatetime(1:est0,:) = [];
vel.medium3.estRbtTwistData(1:est0,:) = [];
vel.medium3.estRbtAccDatetime(1:est0,:) = [];
vel.medium3.estRbtAccData(1:est0,:) = [];
vel.medium3.linVelAbsDatetime(1:est0,:) = [];
vel.medium3.linVelAbsData(1:est0,:) = [];
vel.medium3.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.medium3.estMapPoseData(end-estEndRem:end,:) = [];
vel.medium3.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.medium3.estRbtPoseData(end-estEndRem:end,:) = [];
vel.medium3.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.medium3.estRbtTwistData(end-estEndRem:end,:) = [];
vel.medium3.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.medium3.estRbtAccData(end-estEndRem:end,:) = [];
vel.medium3.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.medium3.linVelAbsData(end-estEndRem:end,:) = [];

% medium4
gnss0 = 0;
imu0 = 0;
est0 = 237;
estEndRem = 57;
vel.medium4.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.medium4.gnssMapPoseData(1:gnss0,:) = [];
vel.medium4.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.medium4.gnssRbtPoseData(1:gnss0,:) = [];
vel.medium4.imuPoseDatetime(1:imu0,:) = [];
vel.medium4.imuPoseData(1:imu0,:) = [];
vel.medium4.imuTwistDatetime(1:imu0,:) = [];
vel.medium4.imuTwistData(1:imu0,:) = [];
vel.medium4.imuAccDatetime(1:imu0,:) = [];
vel.medium4.imuAccData(1:imu0,:) = [];
vel.medium4.estMapPoseDatetime(1:est0,:) = [];
vel.medium4.estMapPoseData(1:est0,:) = [];
vel.medium4.estRbtPoseDatetime(1:est0,:) = [];
vel.medium4.estRbtPoseData(1:est0,:) = [];
vel.medium4.estRbtTwistDatetime(1:est0,:) = [];
vel.medium4.estRbtTwistData(1:est0,:) = [];
vel.medium4.estRbtAccDatetime(1:est0,:) = [];
vel.medium4.estRbtAccData(1:est0,:) = [];
vel.medium4.linVelAbsDatetime(1:est0,:) = [];
vel.medium4.linVelAbsData(1:est0,:) = [];
vel.medium4.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.medium4.estMapPoseData(end-estEndRem:end,:) = [];
vel.medium4.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.medium4.estRbtPoseData(end-estEndRem:end,:) = [];
vel.medium4.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.medium4.estRbtTwistData(end-estEndRem:end,:) = [];
vel.medium4.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.medium4.estRbtAccData(end-estEndRem:end,:) = [];
vel.medium4.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.medium4.linVelAbsData(end-estEndRem:end,:) = [];

% medium5
gnss0 = 1;
imu0 = 0;
est0 = 215;
estEndRem = 50;
vel.medium5.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.medium5.gnssMapPoseData(1:gnss0,:) = [];
vel.medium5.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.medium5.gnssRbtPoseData(1:gnss0,:) = [];
vel.medium5.imuPoseDatetime(1:imu0,:) = [];
vel.medium5.imuPoseData(1:imu0,:) = [];
vel.medium5.imuTwistDatetime(1:imu0,:) = [];
vel.medium5.imuTwistData(1:imu0,:) = [];
vel.medium5.imuAccDatetime(1:imu0,:) = [];
vel.medium5.imuAccData(1:imu0,:) = [];
vel.medium5.estMapPoseDatetime(1:est0,:) = [];
vel.medium5.estMapPoseData(1:est0,:) = [];
vel.medium5.estRbtPoseDatetime(1:est0,:) = [];
vel.medium5.estRbtPoseData(1:est0,:) = [];
vel.medium5.estRbtTwistDatetime(1:est0,:) = [];
vel.medium5.estRbtTwistData(1:est0,:) = [];
vel.medium5.estRbtAccDatetime(1:est0,:) = [];
vel.medium5.estRbtAccData(1:est0,:) = [];
vel.medium5.linVelAbsDatetime(1:est0,:) = [];
vel.medium5.linVelAbsData(1:est0,:) = [];
vel.medium5.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.medium5.estMapPoseData(end-estEndRem:end,:) = [];
vel.medium5.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.medium5.estRbtPoseData(end-estEndRem:end,:) = [];
vel.medium5.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.medium5.estRbtTwistData(end-estEndRem:end,:) = [];
vel.medium5.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.medium5.estRbtAccData(end-estEndRem:end,:) = [];
vel.medium5.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.medium5.linVelAbsData(end-estEndRem:end,:) = [];

% fast1
gnss0 = 0;
imu0 = 0;
est0 = 126;
estEndRem = 30;
vel.fast1.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.fast1.gnssMapPoseData(1:gnss0,:) = [];
vel.fast1.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.fast1.gnssRbtPoseData(1:gnss0,:) = [];
vel.fast1.imuPoseDatetime(1:imu0,:) = [];
vel.fast1.imuPoseData(1:imu0,:) = [];
vel.fast1.imuTwistDatetime(1:imu0,:) = [];
vel.fast1.imuTwistData(1:imu0,:) = [];
vel.fast1.imuAccDatetime(1:imu0,:) = [];
vel.fast1.imuAccData(1:imu0,:) = [];
vel.fast1.estMapPoseDatetime(1:est0,:) = [];
vel.fast1.estMapPoseData(1:est0,:) = [];
vel.fast1.estRbtPoseDatetime(1:est0,:) = [];
vel.fast1.estRbtPoseData(1:est0,:) = [];
vel.fast1.estRbtTwistDatetime(1:est0,:) = [];
vel.fast1.estRbtTwistData(1:est0,:) = [];
vel.fast1.estRbtAccDatetime(1:est0,:) = [];
vel.fast1.estRbtAccData(1:est0,:) = [];
vel.fast1.linVelAbsDatetime(1:est0,:) = [];
vel.fast1.linVelAbsData(1:est0,:) = [];
vel.fast1.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.fast1.estMapPoseData(end-estEndRem:end,:) = [];
vel.fast1.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.fast1.estRbtPoseData(end-estEndRem:end,:) = [];
vel.fast1.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.fast1.estRbtTwistData(end-estEndRem:end,:) = [];
vel.fast1.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.fast1.estRbtAccData(end-estEndRem:end,:) = [];
vel.fast1.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.fast1.linVelAbsData(end-estEndRem:end,:) = [];

% fast2
gnss0 = 2;
imu0 = 0;
est0 = 140;
estEndRem = 54;
vel.fast2.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.fast2.gnssMapPoseData(1:gnss0,:) = [];
vel.fast2.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.fast2.gnssRbtPoseData(1:gnss0,:) = [];
vel.fast2.imuPoseDatetime(1:imu0,:) = [];
vel.fast2.imuPoseData(1:imu0,:) = [];
vel.fast2.imuTwistDatetime(1:imu0,:) = [];
vel.fast2.imuTwistData(1:imu0,:) = [];
vel.fast2.imuAccDatetime(1:imu0,:) = [];
vel.fast2.imuAccData(1:imu0,:) = [];
vel.fast2.estMapPoseDatetime(1:est0,:) = [];
vel.fast2.estMapPoseData(1:est0,:) = [];
vel.fast2.estRbtPoseDatetime(1:est0,:) = [];
vel.fast2.estRbtPoseData(1:est0,:) = [];
vel.fast2.estRbtTwistDatetime(1:est0,:) = [];
vel.fast2.estRbtTwistData(1:est0,:) = [];
vel.fast2.estRbtAccDatetime(1:est0,:) = [];
vel.fast2.estRbtAccData(1:est0,:) = [];
vel.fast2.linVelAbsDatetime(1:est0,:) = [];
vel.fast2.linVelAbsData(1:est0,:) = [];
vel.fast2.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.fast2.estMapPoseData(end-estEndRem:end,:) = [];
vel.fast2.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.fast2.estRbtPoseData(end-estEndRem:end,:) = [];
vel.fast2.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.fast2.estRbtTwistData(end-estEndRem:end,:) = [];
vel.fast2.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.fast2.estRbtAccData(end-estEndRem:end,:) = [];
vel.fast2.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.fast2.linVelAbsData(end-estEndRem:end,:) = [];

% fast3
gnss0 = 0;
imu0 = 0;
est0 = 171;
estEndRem = 35;
vel.fast3.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.fast3.gnssMapPoseData(1:gnss0,:) = [];
vel.fast3.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.fast3.gnssRbtPoseData(1:gnss0,:) = [];
vel.fast3.imuPoseDatetime(1:imu0,:) = [];
vel.fast3.imuPoseData(1:imu0,:) = [];
vel.fast3.imuTwistDatetime(1:imu0,:) = [];
vel.fast3.imuTwistData(1:imu0,:) = [];
vel.fast3.imuAccDatetime(1:imu0,:) = [];
vel.fast3.imuAccData(1:imu0,:) = [];
vel.fast3.estMapPoseDatetime(1:est0,:) = [];
vel.fast3.estMapPoseData(1:est0,:) = [];
vel.fast3.estRbtPoseDatetime(1:est0,:) = [];
vel.fast3.estRbtPoseData(1:est0,:) = [];
vel.fast3.estRbtTwistDatetime(1:est0,:) = [];
vel.fast3.estRbtTwistData(1:est0,:) = [];
vel.fast3.estRbtAccDatetime(1:est0,:) = [];
vel.fast3.estRbtAccData(1:est0,:) = [];
vel.fast3.linVelAbsDatetime(1:est0,:) = [];
vel.fast3.linVelAbsData(1:est0,:) = [];
vel.fast3.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.fast3.estMapPoseData(end-estEndRem:end,:) = [];
vel.fast3.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.fast3.estRbtPoseData(end-estEndRem:end,:) = [];
vel.fast3.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.fast3.estRbtTwistData(end-estEndRem:end,:) = [];
vel.fast3.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.fast3.estRbtAccData(end-estEndRem:end,:) = [];
vel.fast3.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.fast3.linVelAbsData(end-estEndRem:end,:) = [];

% fast4
gnss0 = 0;
imu0 = 0;
est0 = 162;
estEndRem = 30;
vel.fast4.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.fast4.gnssMapPoseData(1:gnss0,:) = [];
vel.fast4.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.fast4.gnssRbtPoseData(1:gnss0,:) = [];
vel.fast4.imuPoseDatetime(1:imu0,:) = [];
vel.fast4.imuPoseData(1:imu0,:) = [];
vel.fast4.imuTwistDatetime(1:imu0,:) = [];
vel.fast4.imuTwistData(1:imu0,:) = [];
vel.fast4.imuAccDatetime(1:imu0,:) = [];
vel.fast4.imuAccData(1:imu0,:) = [];
vel.fast4.estMapPoseDatetime(1:est0,:) = [];
vel.fast4.estMapPoseData(1:est0,:) = [];
vel.fast4.estRbtPoseDatetime(1:est0,:) = [];
vel.fast4.estRbtPoseData(1:est0,:) = [];
vel.fast4.estRbtTwistDatetime(1:est0,:) = [];
vel.fast4.estRbtTwistData(1:est0,:) = [];
vel.fast4.estRbtAccDatetime(1:est0,:) = [];
vel.fast4.estRbtAccData(1:est0,:) = [];
vel.fast4.linVelAbsDatetime(1:est0,:) = [];
vel.fast4.linVelAbsData(1:est0,:) = [];
vel.fast4.estMapPoseDatetime(end-estEndRem:end,:) = [];
vel.fast4.estMapPoseData(end-estEndRem:end,:) = [];
vel.fast4.estRbtPoseDatetime(end-estEndRem:end,:) = [];
vel.fast4.estRbtPoseData(end-estEndRem:end,:) = [];
vel.fast4.estRbtTwistDatetime(end-estEndRem:end,:) = [];
vel.fast4.estRbtTwistData(end-estEndRem:end,:) = [];
vel.fast4.estRbtAccDatetime(end-estEndRem:end,:) = [];
vel.fast4.estRbtAccData(end-estEndRem:end,:) = [];
vel.fast4.linVelAbsDatetime(end-estEndRem:end,:) = [];
vel.fast4.linVelAbsData(end-estEndRem:end,:) = [];

% fast5
gnss0 = 0;
imu0 = 0;
est0 = 148;
estEndRem = 23;
vel.fast5.gnssMapPoseDatetime(1:gnss0,:) = [];
vel.fast5.gnssMapPoseData(1:gnss0,:) = [];
vel.fast5.gnssRbtPoseDatetime(1:gnss0,:) = [];
vel.fast5.gnssRbtPoseData(1:gnss0,:) = [];
vel.fast5.imuPoseDatetime(1:imu0,:) = [];
vel.fast5.imuPoseData(1:imu0,:) = [];
vel.fast5.imuTwistDatetime(1:imu0,:) = [];
vel.fast5.imuTwistData(1:imu0,:) = [];
vel.fast5.imuAccDatetime(1:imu0,:) = [];
vel.fast5.imuAccData(1:imu0,:) = [];
vel.fast5.estMapPoseDatetime(1:est0,:) = [];
vel.fast5.estMapPoseData(1:est0,:) = [];
vel.fast5.estRbtPoseDatetime(1:est0,:) = [];
vel.fast5.estRbtPoseData(1:est0,:) = [];
vel.fast5.estRbtTwistDatetime(1:est0,:) = [];
vel.fast5.estRbtTwistData(1:est0,:) = [];
```

```
    vel.fast5.estRbtAccDatetime(1:est0,:) = [];
    vel.fast5.estRbtAccData(1:est0,:) = [];
    vel.fast5.linVelAbsDatetime(1:est0,:) = [];
    vel.fast5.linVelAbsData(1:est0,:) = [];
    vel.fast5.estMapPoseDatetime(end-estEndRem:end,:) = [];
    vel.fast5.estMapPoseData(end-estEndRem:end,:) = [];
    vel.fast5.estRbtPoseDatetime(end-estEndRem:end,:) = [];
    vel.fast5.estRbtPoseData(end-estEndRem:end,:) = [];
    vel.fast5.estRbtTwistDatetime(end-estEndRem:end,:) = [];
    vel.fast5.estRbtTwistData(end-estEndRem:end,:) = [];
    vel.fast5.estRbtAccDatetime(end-estEndRem:end,:) = [];
    vel.fast5.estRbtAccData(end-estEndRem:end,:) = [];
    vel.fast5.linVelAbsDatetime(end-estEndRem:end,:) = [];
    vel.fast5.linVelAbsData(end-estEndRem:end,:) = [];


end

function vel = zeroMapData(vel)

    % slow1
    mapZero = vel.slow1.gnssMapPoseData(1,:);
    vel.slow1.gnssMapPoseData =...
        vel.slow1.gnssMapPoseData-mapZero;
    vel.slow1.estMapPoseData =...
        vel.slow1.estMapPoseData-mapZero;
    rbtZero = vel.slow1.gnssRbtPoseData(1,:);
    vel.slow1.gnssRbtPoseData =...
        vel.slow1.gnssRbtPoseData-rbtZero;
    vel.slow1.estRbtPoseData =...
        vel.slow1.estRbtPoseData-rbtZero;

    % slow2
    mapZero = vel.slow2.gnssMapPoseData(1,:);
    vel.slow2.gnssMapPoseData =...
        vel.slow2.gnssMapPoseData-mapZero;
    vel.slow2.estMapPoseData =...
        vel.slow2.estMapPoseData-mapZero;
    rbtZero = vel.slow2.gnssRbtPoseData(1,:);
    vel.slow2.gnssRbtPoseData =...
        vel.slow2.gnssRbtPoseData-rbtZero;
    vel.slow2.estRbtPoseData =...
        vel.slow2.estRbtPoseData-rbtZero;

    % slow3
    mapZero = vel.slow3.gnssMapPoseData(1,:);
    vel.slow3.gnssMapPoseData =...
        vel.slow3.gnssMapPoseData-mapZero;
    vel.slow3.estMapPoseData =...
        vel.slow3.estMapPoseData-mapZero;
    rbtZero = vel.slow3.gnssRbtPoseData(1,:);
    vel.slow3.gnssRbtPoseData =...
        vel.slow3.gnssRbtPoseData-rbtZero;
    vel.slow3.estRbtPoseData =...
        vel.slow3.estRbtPoseData-rbtZero;

    % slow4
    mapZero = vel.slow4.gnssMapPoseData(1,:);
    vel.slow4.gnssMapPoseData =...
        vel.slow4.gnssMapPoseData-mapZero;
    vel.slow4.estMapPoseData =...
        vel.slow4.estMapPoseData-mapZero;
    rbtZero = vel.slow4.gnssRbtPoseData(1,:);
    vel.slow4.gnssRbtPoseData =...
        vel.slow4.gnssRbtPoseData-rbtZero;
    vel.slow4.estRbtPoseData =...
        vel.slow4.estRbtPoseData-rbtZero;

    % slow5
    mapZero = vel.slow5.gnssMapPoseData(1,:);
```

```
vel.slow5.gnssMapPoseData =...
    vel.slow5.gnssMapPoseData-mapZero;
vel.slow5.estMapPoseData =...
    vel.slow5.estMapPoseData-mapZero;
rbtZero = vel.slow5.gnssRbtPoseData(1,:);
vel.slow5.gnssRbtPoseData =...
    vel.slow5.gnssRbtPoseData-rbtZero;
vel.slow5.estRbtPoseData =...
    vel.slow5.estRbtPoseData-rbtZero;

% medium1
mapZero = vel.medium1.gnssMapPoseData(1,:);
vel.medium1.gnssMapPoseData =...
    vel.medium1.gnssMapPoseData-mapZero;
vel.medium1.estMapPoseData =...
    vel.medium1.estMapPoseData-mapZero;
rbtZero = vel.medium1.gnssRbtPoseData(1,:);
vel.medium1.gnssRbtPoseData =...
    vel.medium1.gnssRbtPoseData-rbtZero;
vel.medium1.estRbtPoseData =...
    vel.medium1.estRbtPoseData-rbtZero;

% medium2
mapZero = vel.medium2.gnssMapPoseData(1,:);
vel.medium2.gnssMapPoseData =...
    vel.medium2.gnssMapPoseData-mapZero;
vel.medium2.estMapPoseData =...
    vel.medium2.estMapPoseData-mapZero;
rbtZero = vel.medium2.gnssRbtPoseData(1,:);
vel.medium2.gnssRbtPoseData =...
    vel.medium2.gnssRbtPoseData-rbtZero;
vel.medium2.estRbtPoseData =...
    vel.medium2.estRbtPoseData-rbtZero;

% medium3
mapZero = vel.medium3.gnssMapPoseData(1,:);
vel.medium3.gnssMapPoseData =...
    vel.medium3.gnssMapPoseData-mapZero;
vel.medium3.estMapPoseData =...
    vel.medium3.estMapPoseData-mapZero;
rbtZero = vel.medium3.gnssRbtPoseData(1,:);
vel.medium3.gnssRbtPoseData =...
    vel.medium3.gnssRbtPoseData-rbtZero;
vel.medium3.estRbtPoseData =...
    vel.medium3.estRbtPoseData-rbtZero;

% medium4
mapZero = vel.medium4.gnssMapPoseData(1,:);
vel.medium4.gnssMapPoseData =...
    vel.medium4.gnssMapPoseData-mapZero;
vel.medium4.estMapPoseData =...
    vel.medium4.estMapPoseData-mapZero;
rbtZero = vel.medium4.gnssRbtPoseData(1,:);
vel.medium4.gnssRbtPoseData =...
    vel.medium4.gnssRbtPoseData-rbtZero;
vel.medium4.estRbtPoseData =...
    vel.medium4.estRbtPoseData-rbtZero;

% medium5
mapZero = vel.medium5.gnssMapPoseData(1,:);
vel.medium5.gnssMapPoseData =...
    vel.medium5.gnssMapPoseData-mapZero;
vel.medium5.estMapPoseData =...
    vel.medium5.estMapPoseData-mapZero;
rbtZero = vel.medium5.gnssRbtPoseData(1,:);
vel.medium5.gnssRbtPoseData =...
    vel.medium5.gnssRbtPoseData-rbtZero;
vel.medium5.estRbtPoseData =...
    vel.medium5.estRbtPoseData-rbtZero;
    % fast1
    mapZero = vel.fast1.gnssMapPoseData(1,:);
    vel.fast1.gnssMapPoseData =...
        vel.fast1.gnssMapPoseData-mapZero;
    vel.fast1.estMapPoseData =...
        vel.fast1.estMapPoseData-mapZero;
    rbtZero = vel.fast1.gnssRbtPoseData(1,:);
    vel.fast1.gnssRbtPoseData =...
        vel.fast1.gnssRbtPoseData-rbtZero;
    vel.fast1.estRbtPoseData =...
        vel.fast1.estRbtPoseData-rbtZero;

    % fast2
    mapZero = vel.fast2.gnssMapPoseData(1,:);
    vel.fast2.gnssMapPoseData =...
        vel.fast2.gnssMapPoseData-mapZero;
    vel.fast2.estMapPoseData =...
        vel.fast2.estMapPoseData-mapZero;
    rbtZero = vel.fast2.gnssRbtPoseData(1,:);
    vel.fast2.gnssRbtPoseData =...
        vel.fast2.gnssRbtPoseData-rbtZero;
    vel.fast2.estRbtPoseData =...
        vel.fast2.estRbtPoseData-rbtZero;

    % fast3
    mapZero = vel.fast3.gnssMapPoseData(1,:);
    vel.fast3.gnssMapPoseData =...
        vel.fast3.gnssMapPoseData-mapZero;
    vel.fast3.estMapPoseData =...
        vel.fast3.estMapPoseData-mapZero;
    rbtZero = vel.fast3.gnssRbtPoseData(1,:);
    vel.fast3.gnssRbtPoseData =...
        vel.fast3.gnssRbtPoseData-rbtZero;
    vel.fast3.estRbtPoseData =...
        vel.fast3.estRbtPoseData-rbtZero;

    % fast4
    mapZero = vel.fast4.gnssMapPoseData(1,:);
    vel.fast4.gnssMapPoseData =...
        vel.fast4.gnssMapPoseData-mapZero;
    vel.fast4.estMapPoseData =...
        vel.fast4.estMapPoseData-mapZero;
    rbtZero = vel.fast4.gnssRbtPoseData(1,:);
    vel.fast4.gnssRbtPoseData =...
        vel.fast4.gnssRbtPoseData-rbtZero;
    vel.fast4.estRbtPoseData =...
        vel.fast4.estRbtPoseData-rbtZero;

    % fast5
    mapZero = vel.fast5.gnssMapPoseData(1,:);
    vel.fast5.gnssMapPoseData =...
        vel.fast5.gnssMapPoseData-mapZero;
    vel.fast5.estMapPoseData =...
        vel.fast5.estMapPoseData-mapZero;
    rbtZero = vel.fast5.gnssRbtPoseData(1,:);
    vel.fast5.gnssRbtPoseData =...
        vel.fast5.gnssRbtPoseData-rbtZero;
    vel.fast5.estRbtPoseData =...
        vel.fast5.estRbtPoseData-rbtZero;

    % Save to file
    save('velCorrected.mat','vel');

end

function vel = velocityCalculations(vel)

    % Constants
    distanceMoved = 32.1;    % distance moved [m]
    estDt = 1/40;            % estimator time step [s]
```

```
% slow1
vel.slow1.runTime = 21.9;                    % moving time (manually measured) [s]
vel.slow1.meanVel = ...
    distanceMoved/vel.slow1.runTime;         % mean velocity for the run [m/s]
[row,~] = size(vel.slow1.linVelAbsData);
endTime = (row-1)*estDt;
vel.slow1.velTimeVec = (0:estDt:endTime)';  % appropriate time vector for data [s]
y = vel.slow1.linVelAbsData(:,3);
tt = vel.slow1.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);             % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.slow1.meanVel]);                % fit data to first order step response
vel.slow1.tau = coefFit(1);                  % fitted time constant
vel.slow1.vss = coefFit(2);                  % fitted steady state value
vel.slow1.velError =...
    (vel.slow1.vss-vel.slow1.meanVel)/...
    vel.slow1.meanVel;                       % velocity estimator error
vel.slow1.velErrorM =...
    (vel.slow1.meanVel-vel.slow1.vss);       % velocity estimator error

% slow2
vel.slow2.runTime = 21.5;                    % moving time (manually measured) [s]
vel.slow2.meanVel = ...
    distanceMoved/vel.slow2.runTime;         % mean velocity for the run [m/s]
[row,~] = size(vel.slow2.linVelAbsData);
endTime = (row-1)*estDt;
vel.slow2.velTimeVec = (0:estDt:endTime)';  % appropriate time vector for data [s]
y = vel.slow2.linVelAbsData(:,3);
tt = vel.slow2.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);             % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.slow2.meanVel]);                % fit data to first order step response
vel.slow2.tau = coefFit(1);                  % fitted time constant
vel.slow2.vss = coefFit(2);                  % fitted steady state value
vel.slow2.velError =...
    (vel.slow2.vss-vel.slow2.meanVel)/...
    vel.slow2.meanVel;                       % velocity estimator error
vel.slow2.velErrorM =...
    (vel.slow2.meanVel-vel.slow2.vss);       % velocity estimator error

% slow3
vel.slow3.runTime = 22.4;                    % moving time (manually measured) [s]
vel.slow3.meanVel = ...
    distanceMoved/vel.slow3.runTime;         % mean velocity for the run [m/s]
[row,~] = size(vel.slow3.linVelAbsData);
endTime = (row-1)*estDt;
vel.slow3.velTimeVec = (0:estDt:endTime)';  % appropriate time vector for data [s]
y = vel.slow3.linVelAbsData(:,3);
tt = vel.slow3.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);             % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.slow3.meanVel]);                % fit data to first order step response
vel.slow3.tau = coefFit(1);                  % fitted time constant
vel.slow3.vss = coefFit(2);                  % fitted steady state value
vel.slow3.velError =...
    (vel.slow3.vss-vel.slow3.meanVel)/...
    vel.slow3.meanVel;                       % velocity estimator error
vel.slow3.velErrorM =...
    (vel.slow3.meanVel-vel.slow3.vss);       % velocity estimator error

% slow4
vel.slow4.runTime = 22.0;                    % moving time (manually measured) [s]
vel.slow4.meanVel = ...
    distanceMoved/vel.slow4.runTime;         % mean velocity for the run [m/s]
[row,~] = size(vel.slow4.linVelAbsData);
endTime = (row-1)*estDt;
```

```
vel.slow4.velTimeVec = (0:estDt:endTime)';  % appropriate time vector for data [s]
y = vel.slow4.linVelAbsData(:,3);
tt = vel.slow4.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);            % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.slow4.meanVel]);               % fit data to first order step response
vel.slow4.tau = coefFit(1);                 % fitted time constant
vel.slow4.vss = coefFit(2);                 % fitted steady state value
vel.slow4.velError =...
    (vel.slow4.vss-vel.slow4.meanVel)/...
    vel.slow4.meanVel;                      % velocity estimator error
vel.slow4.velErrorM =...
    (vel.slow4.meanVel-vel.slow4.vss);      % velocity estimator error

% slow5
vel.slow5.runTime = 22.1;                   % moving time (manually measured) [s]
vel.slow5.meanVel = ...
    distanceMoved/vel.slow5.runTime;        % mean velocity for the run [m/s]
[row,~] = size(vel.slow5.linVelAbsData);
endTime = (row-1)*estDt;
vel.slow5.velTimeVec = (0:estDt:endTime)';  % appropriate time vector for data [s]
y = vel.slow5.linVelAbsData(:,3);
tt = vel.slow5.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);            % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.slow5.meanVel]);               % fit data to first order step response
vel.slow5.tau = coefFit(1);                 % fitted time constant
vel.slow5.vss = coefFit(2);                 % fitted steady state value
vel.slow5.velError =...
    (vel.slow5.vss-vel.slow5.meanVel)/...
    vel.slow5.meanVel;                      % velocity estimator error
vel.slow5.velErrorM =...
    (vel.slow5.meanVel-vel.slow5.vss);      % velocity estimator error

% medium1
vel.medium1.runTime = 11.1;                     % moving time (manually measured) [s]
vel.medium1.meanVel = ...
    distanceMoved/vel.medium1.runTime;          % mean velocity for the run [m/s]
[row,~] = size(vel.medium1.linVelAbsData);
endTime = (row-1)*estDt;
vel.medium1.velTimeVec = (0:estDt:endTime)';    % appropriate time vector for data [s]
y = vel.medium1.linVelAbsData(:,3);
tt = vel.medium1.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.medium1.meanVel]);                 % fit data to first order step response
vel.medium1.tau = coefFit(1);                   % fitted time constant
vel.medium1.vss = coefFit(2);                   % fitted steady state value
vel.medium1.velError =...
    (vel.medium1.vss-vel.medium1.meanVel)/...
    vel.medium1.meanVel;                        % velocity estimator error
vel.medium1.velErrorM =...
    (vel.medium1.meanVel-vel.medium1.vss);      % velocity estimator error

% medium2
vel.medium2.runTime = 11.1;                     % moving time (manually measured) [s]
vel.medium2.meanVel = ...
    distanceMoved/vel.medium2.runTime;          % mean velocity for the run [m/s]
[row,~] = size(vel.medium2.linVelAbsData);
endTime = (row-1)*estDt;
vel.medium2.velTimeVec = (0:estDt:endTime)';    % appropriate time vector for data [s]
y = vel.medium2.linVelAbsData(:,3);
tt = vel.medium2.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.medium2.meanVel]);                 % fit data to first order step response
```

```
vel.medium2.tau = coefFit(1);                   % fitted time constant
vel.medium2.vss = coefFit(2);                   % fitted steady state value
vel.medium2.velError =...
    (vel.medium2.vss-vel.medium2.meanVel)/...
    vel.medium2.meanVel;                        % velocity estimator error
vel.medium2.velErrorM =...
    (vel.medium2.meanVel-vel.medium2.vss);      % velocity estimator error

% medium3
vel.medium3.runTime = 10.7;                     % moving time (manually measured) [s]
vel.medium3.meanVel = ...
    distanceMoved/vel.medium3.runTime;          % mean velocity for the run [m/s]
[row,~] = size(vel.medium3.linVelAbsData);
endTime = (row-1)*estDt;
vel.medium3.velTimeVec = (0:estDt:endTime)';    % appropriate time vector for data [s]
y = vel.medium3.linVelAbsData(:,3);
tt = vel.medium3.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.medium3.meanVel]);                 % fit data to first order step response
vel.medium3.tau = coefFit(1);                   % fitted time constant
vel.medium3.vss = coefFit(2);                   % fitted steady state value
vel.medium3.velError =...
    (vel.medium3.vss-vel.medium3.meanVel)/...
    vel.medium3.meanVel;                        % velocity estimator error
vel.medium3.velErrorM =...
    (vel.medium3.meanVel-vel.medium3.vss);      % velocity estimator error

% medium4
vel.medium4.runTime = 10.7;                     % moving time (manually measured) [s]
vel.medium4.meanVel = ...
    distanceMoved/vel.medium4.runTime;          % mean velocity for the run [m/s]
[row,~] = size(vel.medium4.linVelAbsData);
endTime = (row-1)*estDt;
vel.medium4.velTimeVec = (0:estDt:endTime)';    % appropriate time vector for data [s]
y = vel.medium4.linVelAbsData(:,3);
tt = vel.medium4.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.medium4.meanVel]);                 % fit data to first order step response
vel.medium4.tau = coefFit(1);                   % fitted time constant
vel.medium4.vss = coefFit(2);                   % fitted steady state value
vel.medium4.velError =...
    (vel.medium4.vss-vel.medium4.meanVel)/...
    vel.medium4.meanVel;                        % velocity estimator error
vel.medium4.velErrorM =...
    (vel.medium4.meanVel-vel.medium4.vss);      % velocity estimator error

% medium5
vel.medium5.runTime = 11.1;                     % moving time (manually measured) [s]
vel.medium5.meanVel = ...
    distanceMoved/vel.medium5.runTime;          % mean velocity for the run [m/s]
[row,~] = size(vel.medium5.linVelAbsData);
endTime = (row-1)*estDt;
vel.medium5.velTimeVec = (0:estDt:endTime)';    % appropriate time vector for data [s]
y = vel.medium5.linVelAbsData(:,3);
tt = vel.medium5.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.medium5.meanVel]);                 % fit data to first order step response
vel.medium5.tau = coefFit(1);                   % fitted time constant
vel.medium5.vss = coefFit(2);                   % fitted steady state value
vel.medium5.velError =...
    (vel.medium5.vss-vel.medium5.meanVel)/...
    vel.medium5.meanVel;                        % velocity estimator error
vel.medium5.velErrorM =...
    (vel.medium5.meanVel-vel.medium5.vss);      % velocity estimator error
```

```
% fast1
vel.fast1.runTime = 7.8;                        % moving time (manually measured) [s]
vel.fast1.meanVel = ...
    distanceMoved/vel.fast1.runTime;            % mean velocity for the run [m/s]
[row,~] = size(vel.fast1.linVelAbsData);
endTime = (row-1)*estDt;
vel.fast1.velTimeVec = (0:estDt:endTime)';      % appropriate time vector for data [s]
y = vel.fast1.linVelAbsData(:,3);
tt = vel.fast1.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.fast1.meanVel]);                   % fit data to first order step response
vel.fast1.tau = coefFit(1);                     % fitted time constant
vel.fast1.vss = coefFit(2);                     % fitted steady state value
vel.fast1.velError =...
    (vel.fast1.vss-vel.fast1.meanVel)/...
    vel.fast1.meanVel;                          % velocity estimator error
vel.fast1.velErrorM =...
    (vel.fast1.meanVel-vel.fast1.vss);      % velocity estimator error

% fast2
vel.fast2.runTime = 8.2;                        % moving time (manually measured) [s]
vel.fast2.meanVel = ...
    distanceMoved/vel.fast2.runTime;            % mean velocity for the run [m/s]
[row,~] = size(vel.fast2.linVelAbsData);
endTime = (row-1)*estDt;
vel.fast2.velTimeVec = (0:estDt:endTime)';      % appropriate time vector for data [s]
y = vel.fast2.linVelAbsData(:,3);
tt = vel.fast2.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.fast2.meanVel]);                   % fit data to first order step response
vel.fast2.tau = coefFit(1);                     % fitted time constant
vel.fast2.vss = coefFit(2);                     % fitted steady state value
vel.fast2.velError =...
    (vel.fast2.vss-vel.fast2.meanVel)/...
    vel.fast2.meanVel;                          % velocity estimator error
vel.fast2.velErrorM =...
    (vel.fast2.meanVel-vel.fast2.vss);      % velocity estimator error

% fast3
vel.fast3.runTime = 8.1;                        % moving time (manually measured) [s]
vel.fast3.meanVel = ...
    distanceMoved/vel.fast3.runTime;            % mean velocity for the run [m/s]
[row,~] = size(vel.fast3.linVelAbsData);
endTime = (row-1)*estDt;
vel.fast3.velTimeVec = (0:estDt:endTime)';      % appropriate time vector for data [s]
y = vel.fast3.linVelAbsData(:,3);
tt = vel.fast3.velTimeVec;
error = @(coef) sum((coef(2)*...
    (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
coefFit = fminsearch(error,...
    [1.5,vel.fast3.meanVel]);                   % fit data to first order step response
vel.fast3.tau = coefFit(1);                     % fitted time constant
vel.fast3.vss = coefFit(2);                     % fitted steady state value
vel.fast3.velError =...
    (vel.fast3.vss-vel.fast3.meanVel)/...
    vel.fast3.meanVel;                          % velocity estimator error
vel.fast3.velErrorM =...
    (vel.fast3.meanVel-vel.fast3.vss);      % velocity estimator error

% fast4
vel.fast4.runTime = 8.3;                        % moving time (manually measured) [s]
vel.fast4.meanVel = ...
    distanceMoved/vel.fast4.runTime;            % mean velocity for the run [m/s]
[row,~] = size(vel.fast4.linVelAbsData);
endTime = (row-1)*estDt;
```

```
    vel.fast4.velTimeVec = (0:estDt:endTime)';      % appropriate time vector for data [s]
    y = vel.fast4.linVelAbsData(:,3);
    tt = vel.fast4.velTimeVec;
    error = @(coef) sum((coef(2)*...
        (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
    coefFit = fminsearch(error,...
        [1.5,vel.fast4.meanVel]);                   % fit data to first order step response
    vel.fast4.tau = coefFit(1);                     % fitted time constant
    vel.fast4.vss = coefFit(2);                     % fitted steady state value
    vel.fast4.velError =...
        (vel.fast4.vss-vel.fast4.meanVel)/...
        vel.fast4.meanVel;                          % velocity estimator error
    vel.fast4.velErrorM =...
        (vel.fast4.meanVel-vel.fast4.vss);      % velocity estimator error

    % fast5
    vel.fast5.runTime = 7.6;                        % moving time (manually measured) [s]
    vel.fast5.meanVel = ...
        distanceMoved/vel.fast5.runTime;            % mean velocity for the run [m/s]
    [row,~] = size(vel.fast5.linVelAbsData);
    endTime = (row-1)*estDt;
    vel.fast5.velTimeVec = (0:estDt:endTime)';      % appropriate time vector for data [s]
    y = vel.fast5.linVelAbsData(:,3);
    tt = vel.fast5.velTimeVec;
    error = @(coef) sum((coef(2)*...
        (1-exp(-tt/coef(1)))-y).^2);                % first order step response error function
    coefFit = fminsearch(error,...
        [1.5,vel.fast5.meanVel]);                   % fit data to first order step response
    vel.fast5.tau = coefFit(1);                     % fitted time constant
    vel.fast5.vss = coefFit(2);                     % fitted steady state value
    vel.fast5.velError =...
        (vel.fast5.vss-vel.fast5.meanVel)/...
        vel.fast5.meanVel;                          % velocity estimator error
    vel.fast5.velErrorM =...
        (vel.fast5.meanVel-vel.fast5.vss);      % velocity estimator error

end

function plotAllVelData(vel)

    % Set font size
    titleFontSize = 32;
    defaultFontSize = 28;
    markerSize = 10;

    % Fitted curves
    vel.slow1.fitted = vel.slow1.vss*...
        (1-exp(-vel.slow1.velTimeVec/vel.slow1.tau));
    vel.slow2.fitted = vel.slow2.vss*...
        (1-exp(-vel.slow2.velTimeVec/vel.slow2.tau));
    vel.slow3.fitted = vel.slow3.vss*...
        (1-exp(-vel.slow3.velTimeVec/vel.slow3.tau));
    vel.slow4.fitted = vel.slow4.vss*...
        (1-exp(-vel.slow4.velTimeVec/vel.slow4.tau));
    vel.slow5.fitted = vel.slow5.vss*...
        (1-exp(-vel.slow5.velTimeVec/vel.slow5.tau));
    vel.medium1.fitted = vel.medium1.vss*...
        (1-exp(-vel.medium1.velTimeVec/vel.medium1.tau));
    vel.medium2.fitted = vel.medium2.vss*...
        (1-exp(-vel.medium2.velTimeVec/vel.medium2.tau));
    vel.medium3.fitted = vel.medium3.vss*...
        (1-exp(-vel.medium3.velTimeVec/vel.medium3.tau));
    vel.medium4.fitted = vel.medium4.vss*...
        (1-exp(-vel.medium4.velTimeVec/vel.medium4.tau));
    vel.medium5.fitted = vel.medium5.vss*...
        (1-exp(-vel.medium5.velTimeVec/vel.medium5.tau));
    vel.fast1.fitted = vel.fast1.vss*...
        (1-exp(-vel.fast1.velTimeVec/vel.fast1.tau));
    vel.fast2.fitted = vel.fast2.vss*...
        (1-exp(-vel.fast2.velTimeVec/vel.fast2.tau));
    vel.fast3.fitted = vel.fast3.vss*...
        (1-exp(-vel.fast3.velTimeVec/vel.fast3.tau));
    vel.fast4.fitted = vel.fast4.vss*...
        (1-exp(-vel.fast4.velTimeVec/vel.fast4.tau));
    vel.fast5.fitted = vel.fast5.vss*...
        (1-exp(-vel.fast5.velTimeVec/vel.fast5.tau));

    figure(1);
    clf;
    hold on;
    plot(vel.slow1.velTimeVec,...
        vel.slow1.linVelAbsData(:,3),...
        'b','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.slow1.velTimeVec,...
        vel.slow1.fitted,...
        'b','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.slow2.velTimeVec,...
        vel.slow2.linVelAbsData(:,3),...
        'r','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.slow2.velTimeVec,...
        vel.slow2.fitted,...
        'r','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.slow3.velTimeVec,...
        vel.slow3.linVelAbsData(:,3),...
        'g','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.slow3.velTimeVec,...
        vel.slow3.fitted,...
        'g','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.slow4.velTimeVec,...
        vel.slow4.linVelAbsData(:,3),...
        'c','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.slow4.velTimeVec,...
        vel.slow4.fitted,...
        'c','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.slow5.velTimeVec,...
        vel.slow5.linVelAbsData(:,3),...
        'm','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.slow5.velTimeVec,...
        vel.slow5.fitted,...
        'm','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{x}\hspace{1mm}[m/s]$','Interpreter','latex');
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    legend(...

['$trial\hspace{1mm}1\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.slow1.meanVel),
')$'],...

['$trial\hspace{1mm}1\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.slow1.vss),',
\tau=',sprintf('%1.1f',vel.slow1.tau),')$'],...

['$trial\hspace{1mm}2\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.slow2.meanVel),
')$'],...

['$trial\hspace{1mm}2\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.slow2.vss),',
\tau=',sprintf('%1.1f',vel.slow2.tau),')$'],...

['$trial\hspace{1mm}3\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.slow3.meanVel),
')$'],...

['$trial\hspace{1mm}3\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.slow3.vss),',
\tau=',sprintf('%1.1f',vel.slow3.tau),')$'],...

['$trial\hspace{1mm}4\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.slow4.meanVel),
')$'],...

['$trial\hspace{1mm}4\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.slow4.vss),',
\tau=',sprintf('%1.1f',vel.slow4.tau),')$'],...
```

```
['$trial\hspace{1mm}5\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.slow5.meanVel),
')$'],...

['$trial\hspace{1mm}5\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.slow5.vss),',
\tau=',sprintf('%1.1f',vel.slow5.tau),')$'],...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);

title('$Slow\hspace{1mm}Velocity\hspace{1mm}First\hspace{1mm}Order\hspace{1mm}Step\hspace{1mm}Res
ponse\hspace{1mm}Curves\hspace{1mm}(\bar{v}\approx1.4\hspace{1mm}to\hspace{1mm}1.5\hspace{1mm}[m/
s])$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    xlim([0,22.5]);
    set(gcf,'Position',[0 0 1920 1280]);

    figure(2);
    clf;
    hold on;
    plot(vel.medium1.velTimeVec,...
        vel.medium1.linVelAbsData(:,3),...
        'b','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.medium1.velTimeVec,...
        vel.medium1.fitted,...
        'b','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.medium2.velTimeVec,...
        vel.medium2.linVelAbsData(:,3),...
        'r','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.medium2.velTimeVec,...
        vel.medium2.fitted,...
        'r','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.medium3.velTimeVec,...
        vel.medium3.linVelAbsData(:,3),...
        'g','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.medium3.velTimeVec,...
        vel.medium3.fitted,...
        'g','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.medium4.velTimeVec,...
        vel.medium4.linVelAbsData(:,3),...
        'c','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.medium4.velTimeVec,...
        vel.medium4.fitted,...
        'c','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.medium5.velTimeVec,...
        vel.medium5.linVelAbsData(:,3),...
        'm','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.medium5.velTimeVec,...
        vel.medium5.fitted,...
        'm','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{x}\hspace{1mm}[m/s]$','Interpreter','latex');
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    legend(...

['$trial\hspace{1mm}1\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.medium1.meanVel
),')$'],...

['$trial\hspace{1mm}1\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.medium1.vss),
',\tau=',sprintf('%1.1f',vel.medium1.tau),')$'],...

['$trial\hspace{1mm}2\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.medium2.meanVel
),')$'],...

['$trial\hspace{1mm}2\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.medium2.vss),
',\tau=',sprintf('%1.1f',vel.medium2.tau),')$'],...

['$trial\hspace{1mm}3\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.medium3.meanVel
),')$'],...
```

```
['$trial\hspace{1mm}3\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.medium3.vss),
',\tau=',sprintf('%1.1f',vel.medium3.tau),')$'],...

['$trial\hspace{1mm}4\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.medium4.meanVel
),')$'],...

['$trial\hspace{1mm}4\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.medium4.vss),
',\tau=',sprintf('%1.1f',vel.medium4.tau),')$'],...

['$trial\hspace{1mm}5\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.medium5.meanVel
),')$'],...

['$trial\hspace{1mm}5\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.medium5.vss),
',\tau=',sprintf('%1.1f',vel.medium5.tau),')$'],...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);

title('$Medium\hspace{1mm}Velocity\hspace{1mm}First\hspace{1mm}Order\hspace{1mm}Step\hspace{1mm}R
esponse\hspace{1mm}Curves\hspace{1mm}(\bar{v}\approx2.9\hspace{1mm}to\hspace{1mm}3.0\hspace{1mm}[
m/s])$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    xlim([0,11]);
    set(gcf,'Position',[0 0 1920 1280]);

    figure(3);
    clf;
    hold on;
    plot(vel.fast1.velTimeVec,...
        vel.fast1.linVelAbsData(:,3),...
        'b','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.fast1.velTimeVec,...
        vel.fast1.fitted,...
        'b','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.fast2.velTimeVec,...
        vel.fast2.linVelAbsData(:,3),...
        'r','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.fast2.velTimeVec,...
        vel.fast2.fitted,...
        'r','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.fast3.velTimeVec,...
        vel.fast3.linVelAbsData(:,3),...
        'g','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.fast3.velTimeVec,...
        vel.fast3.fitted,...
        'g','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.fast4.velTimeVec,...
        vel.fast4.linVelAbsData(:,3),...
        'c','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.fast4.velTimeVec,...
        vel.fast4.fitted,...
        'c','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.fast5.velTimeVec,...
        vel.fast5.linVelAbsData(:,3),...
        'm','LineStyle','none','Marker','.','LineWidth',2,'MarkerSize',markerSize);
    plot(vel.fast5.velTimeVec,...
        vel.fast5.fitted,...
        'm','LineStyle','--','Marker','none','LineWidth',2,'MarkerSize',markerSize);
    hold off;
    grid on;
    ylabel('$\dot{x}\hspace{1mm}[m/s]$','Interpreter','latex');
    xlabel('$time\hspace{1mm}[s]$','Interpreter','latex');
    legend(...

['$trial\hspace{1mm}1\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.fast1.meanVel),
')$'],...

['$trial\hspace{1mm}1\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.fast1.vss),',
\tau=',sprintf('%1.1f',vel.fast1.tau),')$'],...
```

```
['$trial\hspace{1mm}2\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.fast2.meanVel),
')$'],...

['$trial\hspace{1mm}2\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.fast2.vss),',
\tau=',sprintf('%1.1f',vel.fast2.tau),')$'],...

['$trial\hspace{1mm}3\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.fast3.meanVel),
')$'],...

['$trial\hspace{1mm}3\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.fast3.vss),',
\tau=',sprintf('%1.1f',vel.fast3.tau),')$'],...

['$trial\hspace{1mm}4\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.fast4.meanVel),
')$'],...

['$trial\hspace{1mm}4\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.fast4.vss),',
\tau=',sprintf('%1.1f',vel.fast4.tau),')$'],...

['$trial\hspace{1mm}5\hspace{1mm}(actual\hspace{1mm}\bar{v}=',sprintf('%1.1f',vel.fast5.meanVel),
')$'],...

['$trial\hspace{1mm}5\hspace{1mm}fit\hspace{1mm}(\bar{v}_{ss}=',sprintf('%1.1f',vel.fast5.vss),',
\tau=',sprintf('%1.1f',vel.fast5.tau),')$'],...
        'Interpreter','latex');
    set(gca,'FontName','Times New Roman','FontSize',defaultFontSize);

title('$Fast\hspace{1mm}Velocity\hspace{1mm}First\hspace{1mm}Order\hspace{1mm}Step\hspace{1mm}Res
ponse\hspace{1mm}Curves\hspace{1mm}(\bar{v}\approx3.9\hspace{1mm}to\hspace{1mm}4.2\hspace{1mm}[m/
s])$',...
        'FontSize',titleFontSize,...
        'Interpreter','latex');
    xlim([0,8]);
    set(gcf,'Position',[0 0 1920 1280]);

end
```