ABSTRACT

Title of Thesis: ESTIMATION AND CONTROL OF AUTONOMOUS RACING DRONE

Swapneel Naphade
Master of Science, 2020

Thesis Directed by: Professor Huan Xu
Institute for Systems Research

Autonomous Drone Racing (ADR) is an annual competition, organized at the International Conference on Intelligent Robots and Systems (IROS), in which research groups all over the world participate to demonstrate the state-of-the-art technology in the autonomous aerial robotics field. This work describes the system development of the Autonomous Racing Drone System for the IROS ADR competition. A gate detection based, computationally light-weight visual-inertial localization (VIL) system is developed. We show that the proposed VIL system has a significantly lower memory usage than the state-of-the-art Monocular VIO systems which makes it suitable to run on resource constraint hardware. A non-linear model predictive control (NMPC) strategy is implemented for high-speed way-point navigation of the racing drone. We show that the NMPC strategy provides better trajectory tracking performance as compared with the traditional PD controller. The VIL system proposed in this work was utilized in the autonomous drone racing system which won the second-place in the IROS ADR 2019, Macau competition.

ESTIMATION AND CONTROL OF AUTONOMOUS RACING DRONE

by

Swapneel Naphade

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2020

Dedication

This thesis is dedicated to my parents who have been a constant source of inspiration and support.

Acknowledgments

The success of this work is owed to the hard work and determination of many individuals who have assisted me directly or indirectly over the period of the 2 years of my Masters' studies.

First of all, I would like to extend my gratitude towards my advisor, Dr. Huan Xu, who provided me the wonderful opportunity to work on a novel project and provided valuable guidance and assistance in successfully accomplishing the project.

I would like to thank all the University of Maryland Autonomous Drone Racing (UMD ADR) team members: Derek Thompson, Sharon Shallom and Micah Moten, without whom it would have been impossible to accomplish this project in the short period. Especially Derek, who worked alongside me for extended hours to make sure the developed system is effective and reliable. I would also like to give special thanks to Vincenz Frenzel. His work for the IROS ADR 2018 competition laid the foundations of this work. I also extend my gratitude towards the team members from the Autonomous Micro Aerial Vehicle team at UMD: Ian Moss, Qingwen Wei and Zachery Lacey, who shared their valuable experience on MAVs with me.

I express my gratitude towards the Maryland Robotics Center for financially supporting this project. I would also like to thank the IROS ADR 2019 organizing committee for organizing such a novel competition which brings autonomous UAV researchers all around the world together under one roof to share their knowledge and experiences.

I would like to thank Dr. John MacCarthy, my previous academic program advisor, and all the ISR faculty members who taught me how to become a great Systems Engineer.

Last but not the least, I would like to thank my parents and my family members including my brother, cousins, aunts, uncles and grandparents who have always supported me, put faith in me and encouraged me to excel in all my endeavors.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ADR	Autonomous Drone Racing
ARDS	Autonomous Racing Drone System
BDD	Block Definition Diagram
CNN	Convolution Neural Network
EKF	Extended Kalman Filter
ESC	Electronic Speed Controller
FPV	First Person View
GCS	Ground Control System
GPS	Global Positioning System
HSV	Hue-Saturation-Value
IBD	Inteface Block Diagram
IBDD	Internal Block Defnintion Diagram
IMU	Inertial Measurement Unit
IROS	International Conference on Intelligent Robots and Systems
ISR	Insitute for Systems Research
LCM	Life Cycle Model
LED	Light Emitting Diode
LMPC	Linear Model Predictive Control(ler)
LOS	Line of Sight
LTV	Linear Time Variant
KPP	Key Performance Parameter
MAV	Micro Aerial Vehicle
MBSE	Model Based Systems Engineering
MPC	Model Predictive Control(ler)
MSCKF	Multi-State Constraint Kalman Filter
MOE	Measure Of Effectiveness
NLopt	Non-Linear Optmization
NMPC	Non-Linear Model Predictive Control(ler)
OpenCV	Open Computer Vision
PD	Proportional - Derivative
PnP	Perspective-n-Point
RMSE	Root Mean Squared Error
SLAM	Simultaneuos Localization And Mapping
SQP	Sequential Quadratic Programming
SSD	Single Shot Detector
SVO	Semi-direct Visaul Odometry
UAV	Unmanned Aerial Vehicle
UBLAS	u Basic Linear Algebra Subprograms
UKF	Unscented Kalman Filter

VIL	Visual Inertial Localization
VIO	Visual Inertial Odometry
YOLO	You Only Look Once

List of Notations

(u_i, v_i)	Image Coordinates in pixels
R	Rotation matrix $\in SO(3)$
(X_i, Y_i, Z_i)	Gate Corner Coordinates in Camera Frame
$r_{P/O}$	Position vector of P w.r.t O [1]
$[{}^{\mathcal{I}}v_{P/O}]_{\mathcal{B}}$	Velocity of point P w.r.t. O in Frame $\mathcal{I}$ expressed in Frame $\mathcal{B}$ [1]
$\frac{{}^{\mathcal{B}}d}{dt}(r_{P/O})$	Time derivative of position vector in Frame $\mathcal{B}$ [1]
${}^{\mathcal{I}}R_{\mathcal{D}}$	Orientation of Drone Body Frame w.r.t. Inertial Frame in form of Rotation Matrix $\in SO(3)$
${}^{\mathcal{I}}\omega^{\mathcal{D}}$	Angular Velocity of Drone Body Frame w.r.t Inertial Frame
Ω	Infinitesimal Rotation Matrix $\in \mathfrak{so}(3)$
$E[\bullet]$	Expected Value
ϕ	Drone Body Roll angle (radians)
θ	Drone Body Pitch angle (radians)
ψ	Drone Body Yaw angle (radians)
M	Moment (Nm)
T	Thrust (N)
τ	Specific Thrust (m/s^2)
m	Drone mass (kg)
$r = [x\ y\ z]^T$	Drone position
$v = [v_x\ v_y\ v_z]^T$	Drone velocity
F_d	Induced drag force (N)
F_g	Gravitational force (N)
g	Gravitational acceleration (m/s^2)
Q_x	NMPC state error penalties
R_u	NMPC control action penalties
P_x	NMPC terminal state error penalties
Θ_{max}	Maximum allowed tilt angle (roll and pitch)
τ_{max}	Maximum allowed specific thrust

[1] Notation is based on book [2]

Chapter 1: Introduction

First Person View (FPV) Drone Racing has been becoming an increasingly popular sport in recent years, worldwide. In these competitions, the racing drones or quadcopters are equipped with a small FPV camera that relays the real-time video feed to an FPV headset worn by the human pilot. The human pilot uses a radio-controller to guide the drone through a racecourse, illuminated with LED light strips and hoops. Human piloted drones in an FPV drone race can fly at speeds up to 80 mph with an average speed of 30 mph. There has been a growing interest in the robotics research community to develop autonomous aerial robots that are capable of accomplishing the challenging task of drone racing.

With a vision to inspire the advancement of autonomous capabilities of existing drone technology, the Autonomous Drone Racing (ADR) competition was first started in 2016 at the International Conference on Intelligent Robots and Systems (IROS) 2016 in Daejeon, Korea. Since then, the competition is being held annually at every IROS conference. Subsequently, ADR competitions were held in Vancouver, Canada in 2017; Madrid, Spain in 2018 and Macau, China in 2019. The challenge in the competition is to develop an autonomous aerial robotic system, capable of traversing a known drone racing course, without collisions, using only onboard sensing and computing resources. The gate configurations in the competition change with time and the competing autonomous systems have to adapt to them to successfully finish the race.

The competition rules for IROS ADR 2019 were quite simple. Each competing team had 3 trials to go through the racecourse with 5 minutes per trial. In these 5 minutes, the racing-drone has to make as many laps as possible through the racecourse and the team with the highest number of laps is declared the winner. The drones were required to be fully autonomous, meaning no external piloting, computing or sensing was allowed. The racecourse had two sets of gates; one stacked horizontally and the other stacked vertically. These gates had LED lights attached to them which were randomly illuminated. The autonomous system had to detect which gate was illuminated and go through it. Figure 1.1 shows the gates and figure 1.2 shows the top view of the racing arena map. Figure 1.3 shows the autonomous drone racing system developed by the team at the University of Maryland.

1.1 Motivation and Objective

Scientists and engineers have been trying to develop systems that exceed human cognitive capabilities to accomplish complex tasks for decades. From the Artificial Intelligence-powered Chess-playing computer, Deep Blue to the Deep Neural Network powered AlphaGo playing the complex game of Go, these systems have proven that they can beat humans in high cognitive tasks by a big margin. Recently, there have been many developments in the autonomous vehicles field which

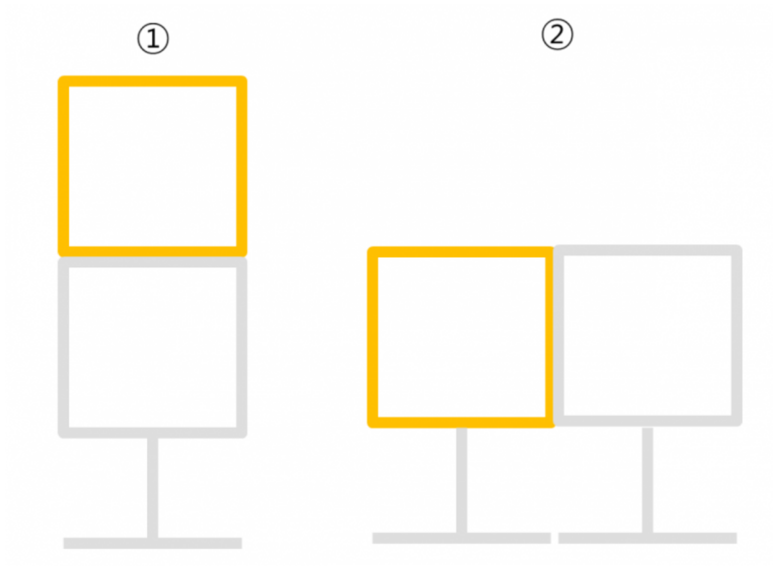

Figure 1.1: IROS ADR 2019 Gate Configuration, (1) Vertically Stacked Gate, (2) Horizontally Stacked Gate

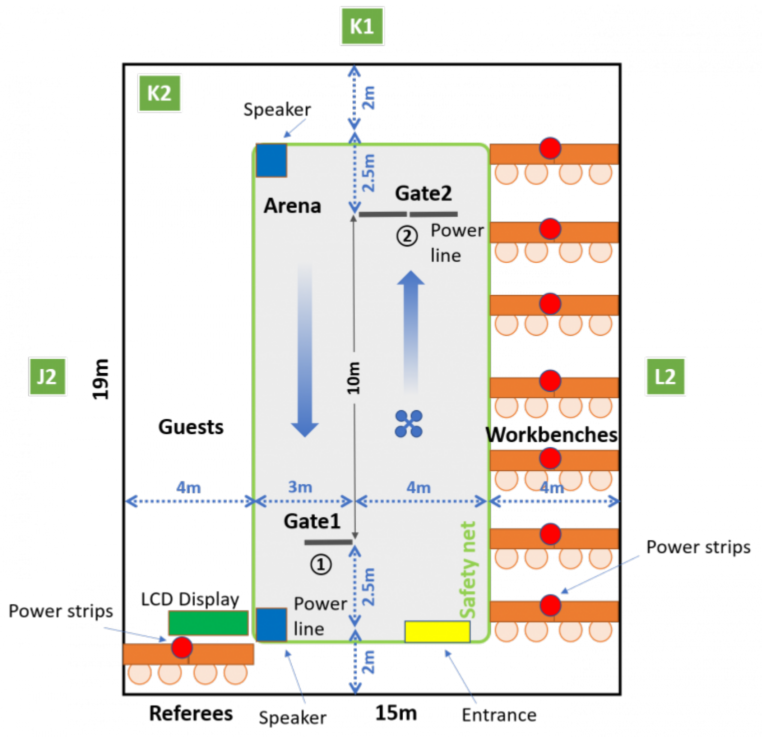

Figure 1.2: IROS ADR 2019 Top View of the Arena

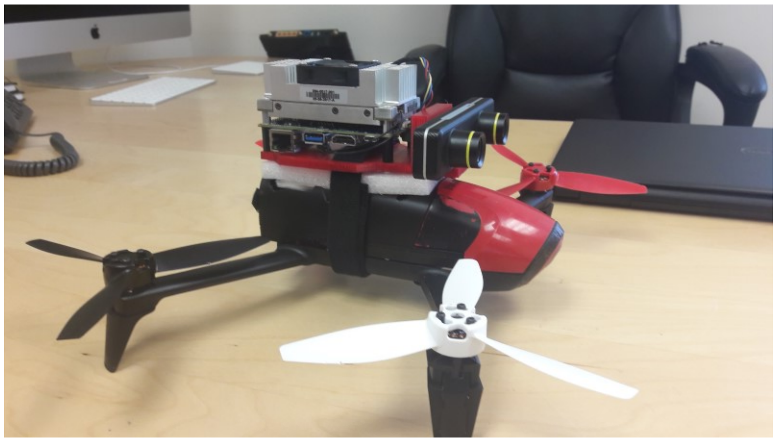

Figure 1.3: University of Maryland's Autonomous Racing Drone

are automating the highly complex task of driving for making road transport safer and more efficient. With the autonomous drone racing challenges like IROS ADR and AlphaPilot, there has been an emergence of a new interest group that wants to make the difficult task of FPV drone racing, autonomous. The motivation to automate the highly involving task of FPV drone racing is not just limited to the notion of beating a human but there is a deeper purpose of pushing the horizons of the existing autonomous unmanned aerial vehicle (UAV) technology. Such autonomous aerial systems can be employed in high-speed relief work during disasters or can be employed in managing indoor warehouses with more efficiency and speed.

The objective of this thesis is to develop a fully autonomous, high speed, aerial robotic system which can traverse through a moderately changing drone-racing course, using only onboard sensing and computing resources. This will be achieved by developing computationally light-weight vision, estimation and control algorithms that can be implemented on inexpensive and light in weight computing hardware onboard the autonomous drone.

1.2 Background on Autonomous Drone Racing

Since, the inception of IROS ADR competition in 2016, numerous developments have taken place in terms of vision, estimation, planning and control of autonomous racing drones [3].

S. Jung et al. [4] were the first group to propose a Single Shot Detector (SSD), which is a Convolution Neural Network (CNN), for gate detection in indoor environments for ADR. They used an Nvidia TK1 as the computer vision hardware for deep learning-based gate detection. The SSD is based on a neural network structure similar to YOLO [5] algorithm and uses VGG-16 as a base network. The detection with SSD is robust even in low lighting conditions however the detection frequency is only 10 Hz which is very low for high-speed gate detection. S. Jung et al. also developed a direct visual servoing method [6] for gate distance measurement and UAV guidance. They used a Luenberger observer for acceleration estimation and a combination of second-order filters and a complimentary filter for velocity estimation. Using the integrated velocity estimates and visual servoing they were able to get accurate position estimates of the racing drone. For gate measurement within 1 m distance of the gate they used a depth-based collision avoidance method in which the gate distance is calculated using the heading of the drone and minimum depth point of the gate in the image. Though this method gave accurate gate distance measurements, it required the drone to spend more time near each gate making the drone slower in the race. Improving upon their gate detection algorithm, S. Jung et. al. [7] proposed ADRNet which is an AlexNet based neural network modified for real-time gate detection. They were able to improve the detection rate at 30 Hz and the detection accuracy of 85.2 %. For guidance and control of the quadrotor, they used the Line of Sight (LOS) vector guidance method. While the LOS guidance strategy works well even with moving gates, it requires the subsequent gates to be always in sight for successful navigation through the racecourse.

Kaufmann et al. [8] applied a deep learning approach for combined perception and control of the racing drone. They used CNN to predict goal direction and desired velocity from a single image from the front-facing camera. The network is trained on a dataset [9] collected by following a minimum-snap trajectory [10] by considering it to be the expert policy. The output of the network is normalized desired velocity which is then scaled according to the desired aggressiveness of the drone. The task completion performance of this system is significantly better than the VIO baseline at lower speeds. Kaufmann et al. [11] developed a deep neural network-based gate detection and measurement algorithm which outputs the relative distance and heading of gates in the drone body frame along with the variance of a multivariate normal distribution of these estimates. Then an Extended Kalman Filter was used to estimate the joint probability of the gate's pose. D. Falanga et al. [12] also introduced a perception aware model predictive controller which computes control inputs to the drone by minimizing state, action as well as perception objectives. The perception objective consists of a quadratic cost function containing the distance and velocity of a projected point of interest on the image plane. This system was the winning entry in the 2018 IROS ADR event.

S. Li et al. [13] developed a visual model predictive localization method for localizing the racing drone with gate measurements on a 72 g racing drone platform. They use a 4 degree of freedom model of drone dynamics in the XY plane to predict the relative position of the gate in drone body frame and then localize the drone by identifying the detected gate using a minimum distance gate-assignment method. They showed that for low-frequency measurements (~ 30 Hz) their RANSAC based fitting approach outperforms the EKF based estimation as it is better at rejecting outliers.

1.3 Contributions of the Thesis

The primary requirements of an autonomous racing drone are to have fast but precise localization capability and low weight for achieving large acceleration with low energy consumption. The existing Neural Networks based solutions for visual gate detection for ADR are very robust to environmental noise but they are slow (update frequency < 30 Hz) and require specialized GPU hardware for computation which are expensive, heavy and have high energy usage. Also, many of them depend on the assumption that a gate will always be visible for position estimation. Some solutions include use of the state-of-the-art Visual-Inertial-Odometry (VIO) systems which provide accurate pose estimation but require high processing and memory resources which add to the overall weight and cost of the autonomous racing drone platform. Hence, there is a need to develop a drone localization system that is faster and lighter than the existing solutions but precise enough for successful racing drone operations.

The main contribution of this thesis is the development of a visual gate detection based localization system for racing drones called the Visual-Inertial-Localization (VIL) system. The VIL system fuses information available from the visual gate detection, optical flow sensor, IMU and known gate positions for the localization of the racing drone. It utilizes a Linear Time-Variant Kalman Filter which consists of asynchronously executed prediction and measurement steps. The proposed system has significantly low memory usage than the existing localization solutions for autonomous racing drones and has very high update frequency (up to 100 Hz).

The proposed VIL system has many applications outside autonomous drone racing where the environment is known and fairly static. Any static indoor environment can be easily made navigable for autonomous drones by adding inexpensive April tags in place of the racing gates. Such systems can be employed in efficient high-speed warehouse management and material handling applications.

1.4 Outline of the Thesis

This thesis presents a Model-Based Systems Engineering (MBSE) approach for autonomous aerial racing system design. Computationally light-weight vision, estimation and control algorithms are developed for autonomous drone racing applications.

Chapter 1 of the thesis introduces the autonomous drone racing event and highlights the motivation and objective of the thesis. The previous work related to autonomous drone racing is also discussed in this chapter.

Chapter 2 describes the system design of the Autonomous Racing Drone System (ARDS). The system development approach is discussed in this chapter. The stakeholder and system requirements are identified and the system architecture is developed.

Chapter 3 describes the visual estimation sub-system of the ARDS. Visual gate detection, gate position estimation and localization of the drone in the racing arena map are discussed in this chapter.

Chapter 4 describes the control system design of the ARDS. The attitude, position and mission control systems of the ARDS are discussed in this chapter. A non-linear model predictive control (NMPC) strategy is presented for the position control of the drone.

Chapter 5 presents the simulation results and discussion of the estimation and control sub-systems of the ARDS. The gate position estimation and localization modules are integrated and verified in a simulation environment. The NMPC strategy is also compared with a PD controller for trajectory tracking performance in simulation.

Chapter 6 concludes the thesis by summarizing the results and identifies a few potential research opportunities for future developments.

Chapter 2: Autonomous Racing Drone System Design

2.1 System Description

The autonomous racing drone system (ARDS) is a quadcopter unmanned aerial vehicle (UAV) that traverses a drone racing course autonomously; without any operator piloting commands; at high speeds; and using only onboard sensor systems in a GPS-denied environment. It utilizes a vision system to detect and estimate the relative position of the racecourse gates, localizes itself in a drone-racing arena map and computes optimal control actions to follow a predetermined path along the racecourse.

2.2 System Development Approach

A semi-formal, V-development Life Cycle Model (LCM) for system development is followed for the ARDS development. Figure 2.1 describes the V-development LCM. Model-Based System Engineering (MBSE) is used to develop the system architecture in the preliminary design phase of the system development.

Initially, the stakeholder and system requirements are identified followed by the preliminary design phase. In the preliminary design phase, the system architecture

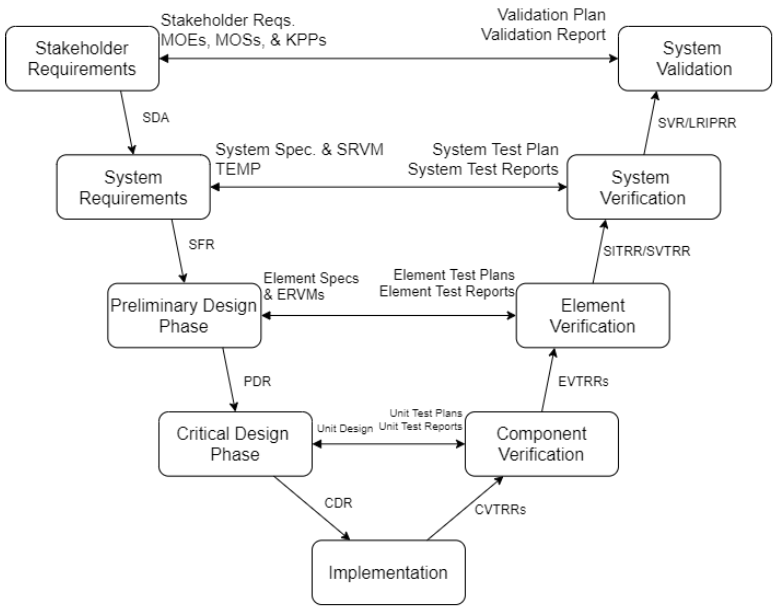

Figure 2.1: V-Developement Approach for ARDS System Developement

is defined which includes system block definition diagram, system interface block diagram and system state-machine diagram. These diagrams describe the system composition, system interface data/control flow and system behavior respectively. The subsequent chapters of this thesis present the critical design and verification stages of the system development LCM.

2.3 Stakeholder and System Requirements

2.3.1 Stakeholder Requirements

The first phase of the system development process is identifying stakeholder requirements. The primary stakeholders of the system are the University of Maryland's Autonomous Drone Racing (ADR) team, the Maryland Robotics Center and IROS ADR 2019 competition organizers and judges. The following stakeholder requirements for the ARDS are directly derived from the IROS ADR 2019 competition rules:

2.3.1.1. The system shall pass through the maximum number of gates in 5 minutes.

2.3.1.2. The system shall traverse the racecourse without collision with gates

2.3.1.3. The system shall traverse the racecourse without collision with arena nets.

2.3.1.4. The system shall use only onboard sensing devices.

2.3.1.5. The system shall use only onboard computing devices.

2.3.1.6. The system shall not receive any piloting inputs from any operators except for the start and emergency stop commands.

2.3.1.7. The system shall pass only through illuminated gates.

2.3.2 System Requirements

The system requirements are high-level technical requirements that are derived from the stakeholder requirements of the ARDS.

2.3.2.1. The system platform battery shall last for at least 5 minutes in one charge.

2.3.2.2. The system platform shall have a high thrust-to-weight ratio.

2.3.2.3. The system platform shall have dimensions less than the gate dimensions.

2.3.2.4. The system platform shall have sensors for orientation, position and velocity estimation onboard.

2.3.2.5. The system platform shall have a computing unit onboard.

2.3.2.6. The system shall have high frames-per-second video capturing capability.

2.3.2.7. The system shall have high precision and accuracy gate position estimation.

2.3.2.8. The system shall have a high estimation update frequency.

2.3.2.9. The system shall be robust to intermittent and noisy visual measurements.

2.3.2.10. The system shall be robust in gate detection in complex background environments.

2.3.2.11. The system shall be able to perform controlled aggressive maneuvers.

2.3.2.12. The system shall have navigation capability in the absence of visual gate cues.

2.4 System Measures of Effectiveness and Key Performance Parameters

Based on the stakeholder and system requirements, a few Measures of Effectiveness (MOE) and Key Performance Parameters (KPP) of the ARDS are identified as shown in Table 2.1.

2.5 System Level Block Definition Diagram

Figure 2.2 shows the System Level Block Definition Diagram (BDD) of the ARDS. The BDD describes the composition of the system and its components. The ARDS consists of subsystems like Hardware, Software, Environment and the User.

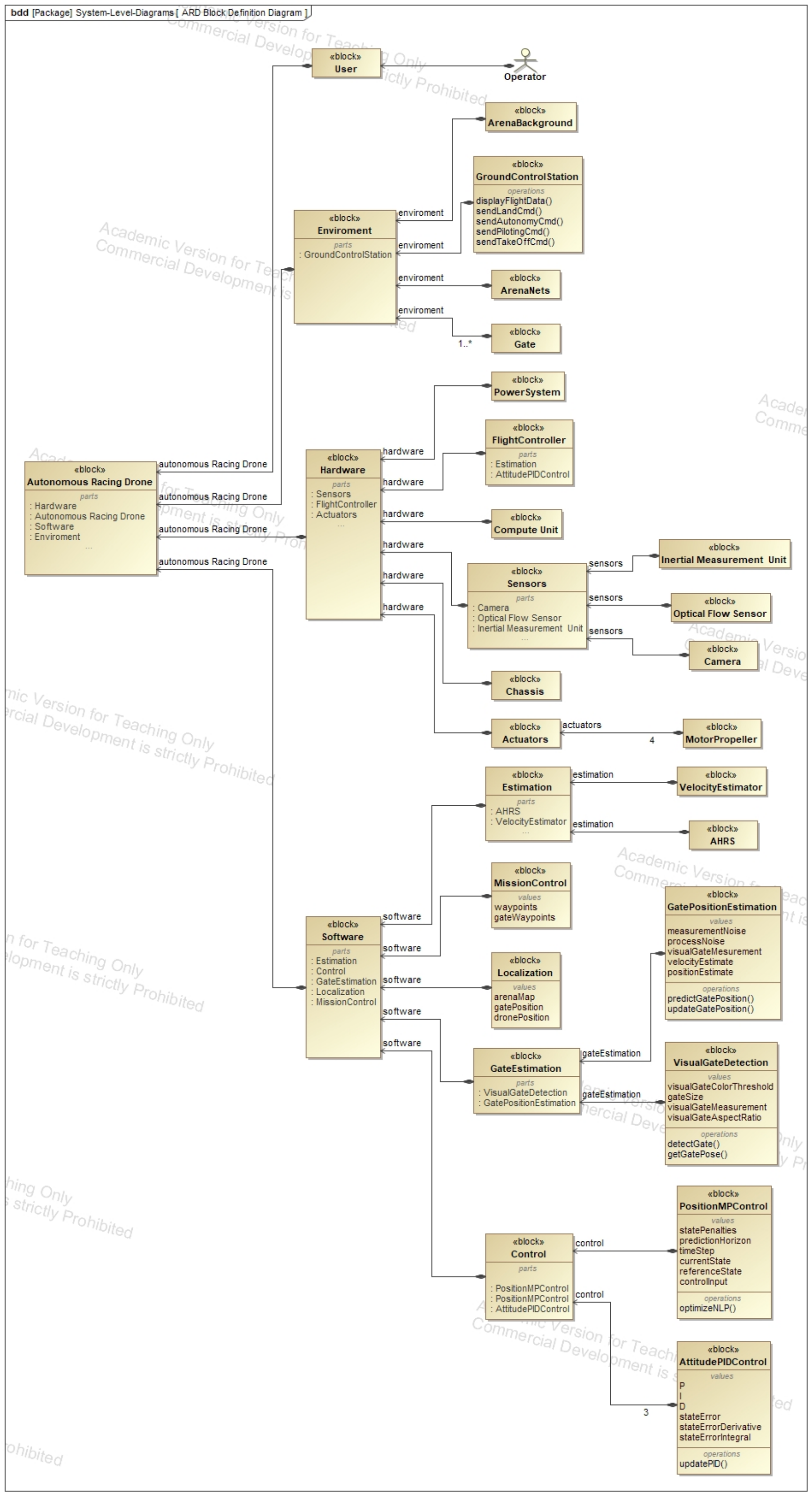

Figure 2.2: ARDS Block Definition Diagram

MOE /KPP ID	MOE /KPP	Description	Unit	Expected value
KPP1	Flight time	Average flight time of the drone between battery charges.	$min.$	$> 7\ min.$
KPP2	Drone Mass	Mass of the drone.	kg	$< 1\ kg$
KPP3	Dimensions	Dimensions of the drone in terms of height, width and length.	m	$max(l, w, h) < 1.0\ m$
MOE1	Maximum Thrust	Maximum Thrust produced by the drone.	N	$> 200\ N$
MOE2	Maximum Linear Velocity	Maximum Linear Velocity achieved by the drone.	m/s	$> 2\ m/s$
MOE3	Estimation rate	Frequency of drone position estimation update.	Hz	$> 30\ Hz$
MOE4	Estimation accuracy	Accuracy of drone position estimation in terms of estimation bias.	m	$< 0.5\ m$
MOE5	Estimation precision	Precision of drone position estimation in terms of one standard deviation.	m	$< 0.5\ m$
MOE6	Trajectory Tracking Accuracy	Accuracy of trajectory tracking in terms of trajectory tracking error bias.	m	$< 0.25\ m$
MOE7	Trajectory Tracking precision	Precision of trajectory tracking in terms of one standard deviation.	m	$< 0.25\ m$
MOE8	Gate Detection Accuracy	Accuracy of Gate Detection in terms of percentage of successful gate detections.	-	$> 90\ \%$

Table 2.1: ARDS MOEs and KPPs

2.5.1 Hardware

The hardware subsystem of the ARDS comprises the power system, sensors, compute unit, flight controller, actuators, communication and chassis. The power system of the ARDS consists of the battery, voltage regulation, power distribution and electronic speed control (ESC) units. Inertial Measurement Unit, Optical Flow sensor and Camera make the sensor suite of the ARDS. An onboard flight controller is used for the low-level (rotational motion) estimation and control of the drone while a compute unit is used for the high-level planning, estimation and control of the drone.

2.5.2 Software

The software subsystem of the ARDS consists of software modules that perform visual estimation, localization, high-level mission control and position control.

The visual estimation module consists of the gate-detection algorithm, gate position measurement algorithm and gate position estimation algorithm. It consumes the video stream data from the camera to detect and measure the relative position of the largest visual gate in each image frame and then utilizes the drone velocity estimates to make an optimal estimate of the relative gate position. The localization module utilizes the estimated gate position, previous drone position estimate and racing arena map information to localize the drone in the racing arena. The Mission Control module manages the current reference waypoint, which is chased by the drone. It also manages the autonomous/manual state of the system. Lastly, the control module computes the optimal control action for the drone to reach the reference waypoint.

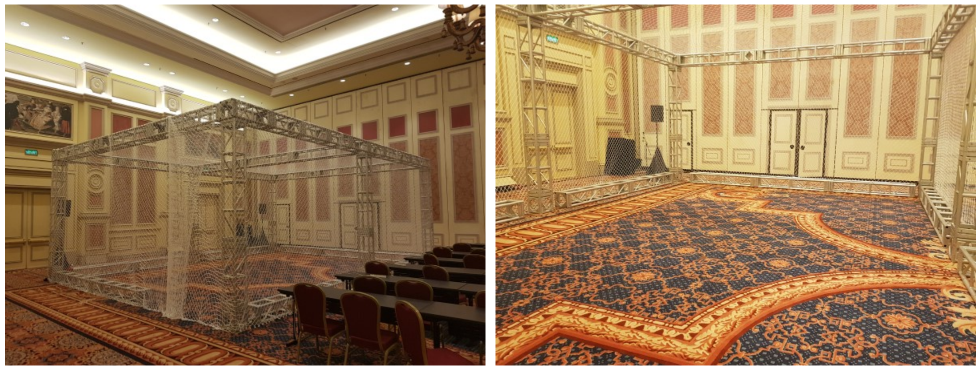

Figure 2.3: IROS ADR 2019, Macau Drone Racing Arena

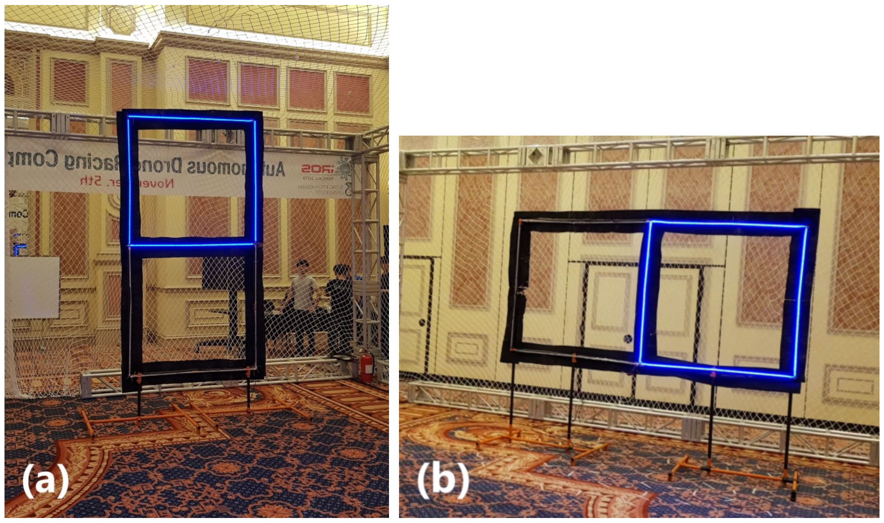

Figure 2.4: IROS ADR 2019, Macau Drone Racing Arena Gates (a) Vertically Stacked Gate, (b) Horizontally Stacked Gate

2.5.3 Environment

The ARDS environment consists of drone racing gates, arena nets and ground control station. The IROS ADR 2019, Macau arena is a 10m long, 3m high and 7m wide netted cage as shown in figure 2.3. There are two sets of LED illuminated gates present in the arena as shown in figure 2.4. The gates are 1.5 m side square gates stacked horizontally and vertically.

The gates are illuminated in random order when the drone makes a successful

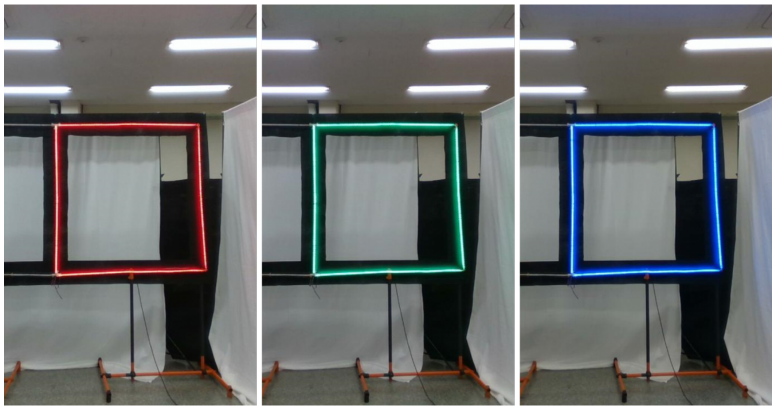

Figure 2.5: RGB Illuminated Gates for IROS ADR 2019, Macau

pass through one gate. A gate pass is successful when the drone goes through an illuminated gate and the entire drone body is out of the gate plane from the other side.

The arena is inside a hall in a hotel hence there is no GPS signal available for the drone for localization. Also, the lighting conditions are dimmed to facilitate the detection of illuminated gates further. Figure 2.5 shows the illuminated gates in three colors.

The ground control system (GCS) works as an interface for the operator to send commands to the ARDS and receive mission-critical data for analysis. The ground control station for the ARDS 2019 was developed by Vincenz Frenzel [14].

2.5.4 User

The user of the system is the operator who sends commands to start or stop the autonomous state of the system as well as piloting commands in the manual state of the system. The user also provides the desired path for traversal in the

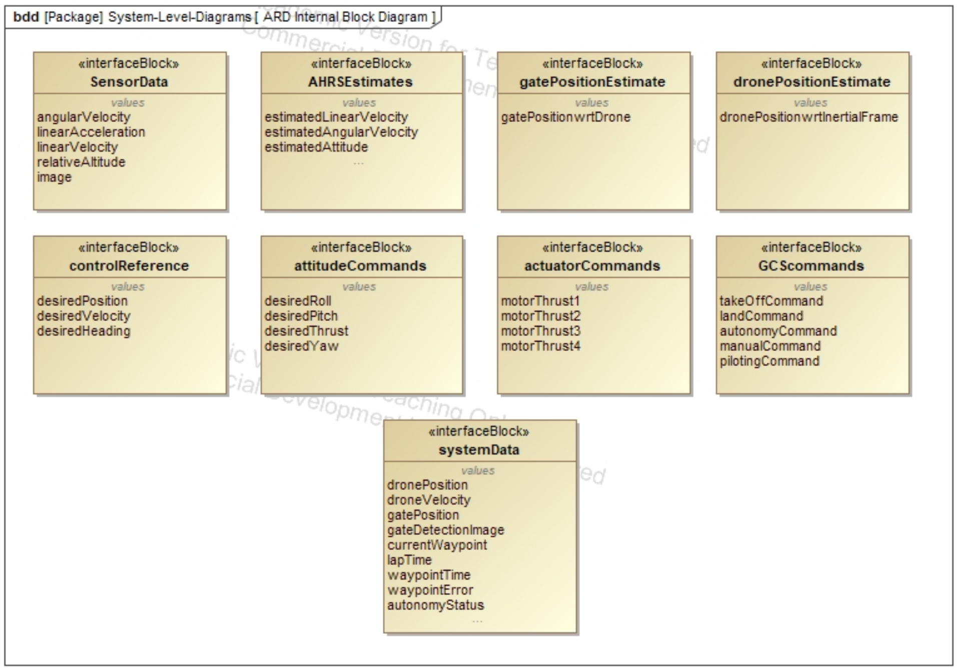

Figure 2.6: ARDS Internal Block Definition Diagram

form of a waypoint list to the Mission Control.

2.6 System Level Interface Block Definition Diagram

The system level Interface Block Diagram (IBD) gives a white box view of the internal data and control flow of the system. Figure 2.7 shows the system level IBD of the ARDS.

The system level Internal Block Definition Diagram (IBDD) describes the interface data definition that flows through the system. Figure 2.6 shows the system level IBDD of the ARDS.

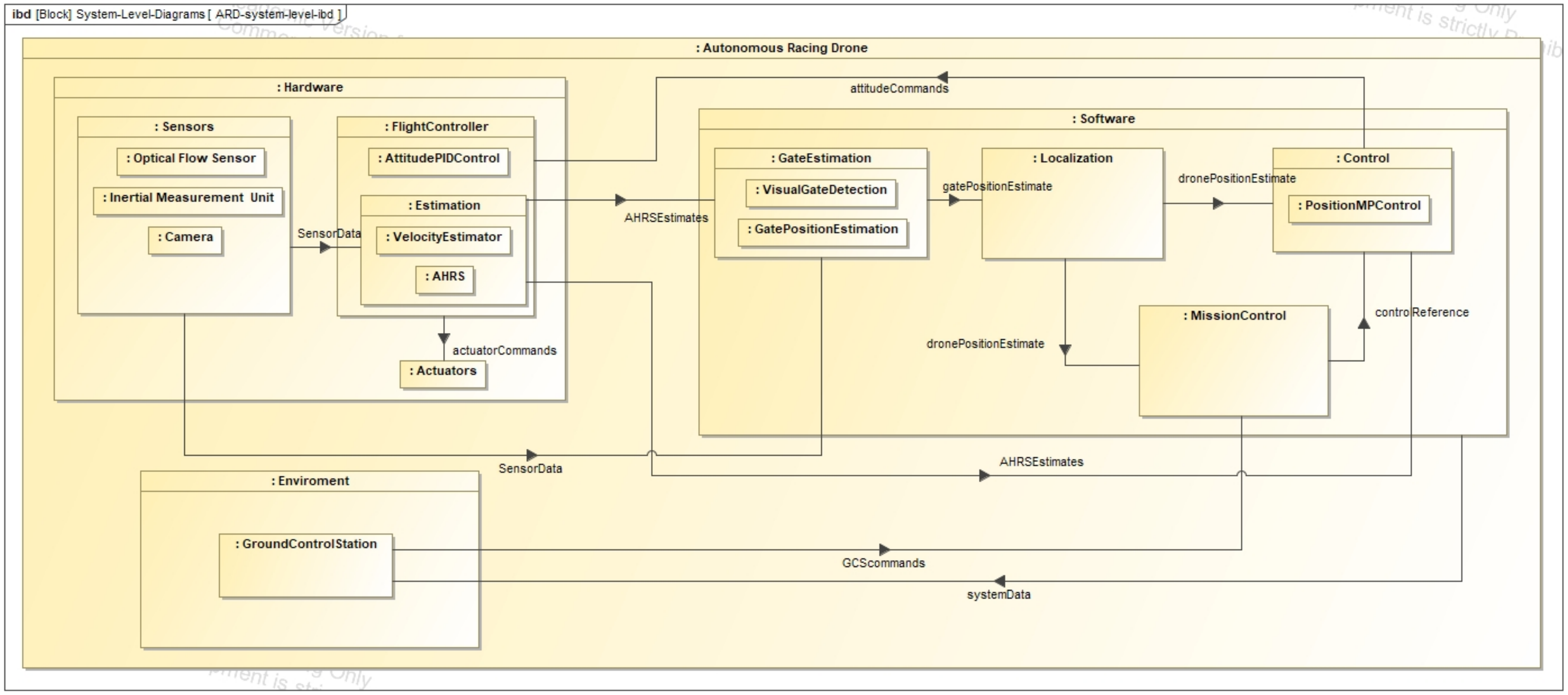

Figure 2.7: ARDS Interface Block Diagram

2.7 State Machine Diagram

The state-machine diagram depicts the behavioral states a system can exhibit during its operation. Figure 2.8 shows the state-machine diagram of the ARDS.

The ARDS can exhibit the following states:

1. Ground: The drone is on the ground and the motors are armed (spinning) at the idle rpm.

2. Takeoff: The drone rises from the ground to attain a pre-determined takeoff height with a constant upward velocity.

3. Hover: The drone hovers at a constant relative altitude to the ground and maintains zero velocity in the inertial frame. It is also known as position hold mode within the UAV community.

4. Autonomous: The drone autonomously follows a pre-determined path, managed by the mission control module. No operator piloting inputs affect the motion of the drone in this state.

5. Manual: The drone receives non-zero manual attitude commands and attains the commanded attitude using the onboard attitude controller. This state is used for manual positioning of the drone in the arena or gain manual control when the drone does not exhibit expected behavior in autonomous mode.

6. Airborne: It is a composite state comprising Takeoff, Hover, Manual and Autonomous states. The drone is off the ground in this state and not descending

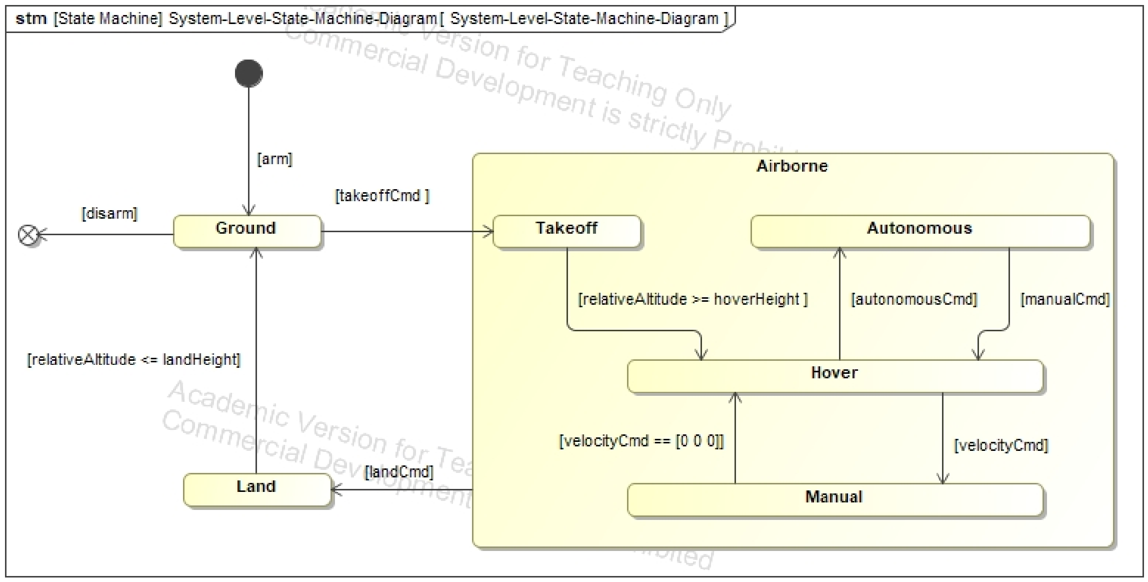
stm [State Machine] System-Level-State-Machine-Diagram[System-Level-State-Machine-Diagram]
[arm]
[disarm]
Ground
[takeoffCmd]
Airborne
Takeoff
Autonomous
[relativeAltitude >= hoverHeight]
[autonomousCmd]
[manualCmd]
Hover
[relativeAltitude <= landHeight]
[velocityCmd == [0 0 0]]
[velocityCmd]
[landCmd]
Land
Manual
Academic Version for Teaching Only
Commercial Development is strictly Prohibited

Figure 2.8: ARDS State Machine Diagram

for landing.

7. Land: The drone descends with a constant downward velocity until it reaches the ground and lands.

The ARDS transitions from one state to another when a corresponding state-transition command is received from the GCS.

2.8 Element Level Activity Diagrams

An activity diagram describes the dynamic behavior of the system components. This section provides the behavioral design of some of the key software components of the ARDS.

2.8.1 Gate Estimation Activity Diagram

Figure 2.9 shows the activity diagram for gate estimation subsystem. The visual gate detection component of the gate estimation subsystem receives an RGB image frame from the camera system for each measurement cycle. This RGB image is converted to an HSV image. The HSV image is then converted to a binary image by applying threshold operation and then blurred with gaussian blur operation. Contours are found on this blurred image and filtered to get the largest gate by area in the image. A perspective-n-point algorithm is used to get the relative translation and orientation of the gate contour with respect to the camera. The relative translation and rotation of the gate contour are transformed to the drone body frame and used as measurements by the gate position estimation module. The gate position estimation module uses an Asynchronous Linear-Time-Variant Kalman Filter to estimate the gate position from available information from the sensors and gate measurement. A detailed visual gate estimation algorithm description is presented in section 3.4 of this thesis.

2.8.2 Localization Activity Diagram

The localization subsystem activity diagram is described in figure 2.10. The localization module receives the estimated gate position in the drone body frame and projects it in the inertial frame using the previous drone position. The projected gate position is then tallied with a list of pre-determined gate positions provided by the user and the gate nearest to the projected gate position is selected as a landmark.

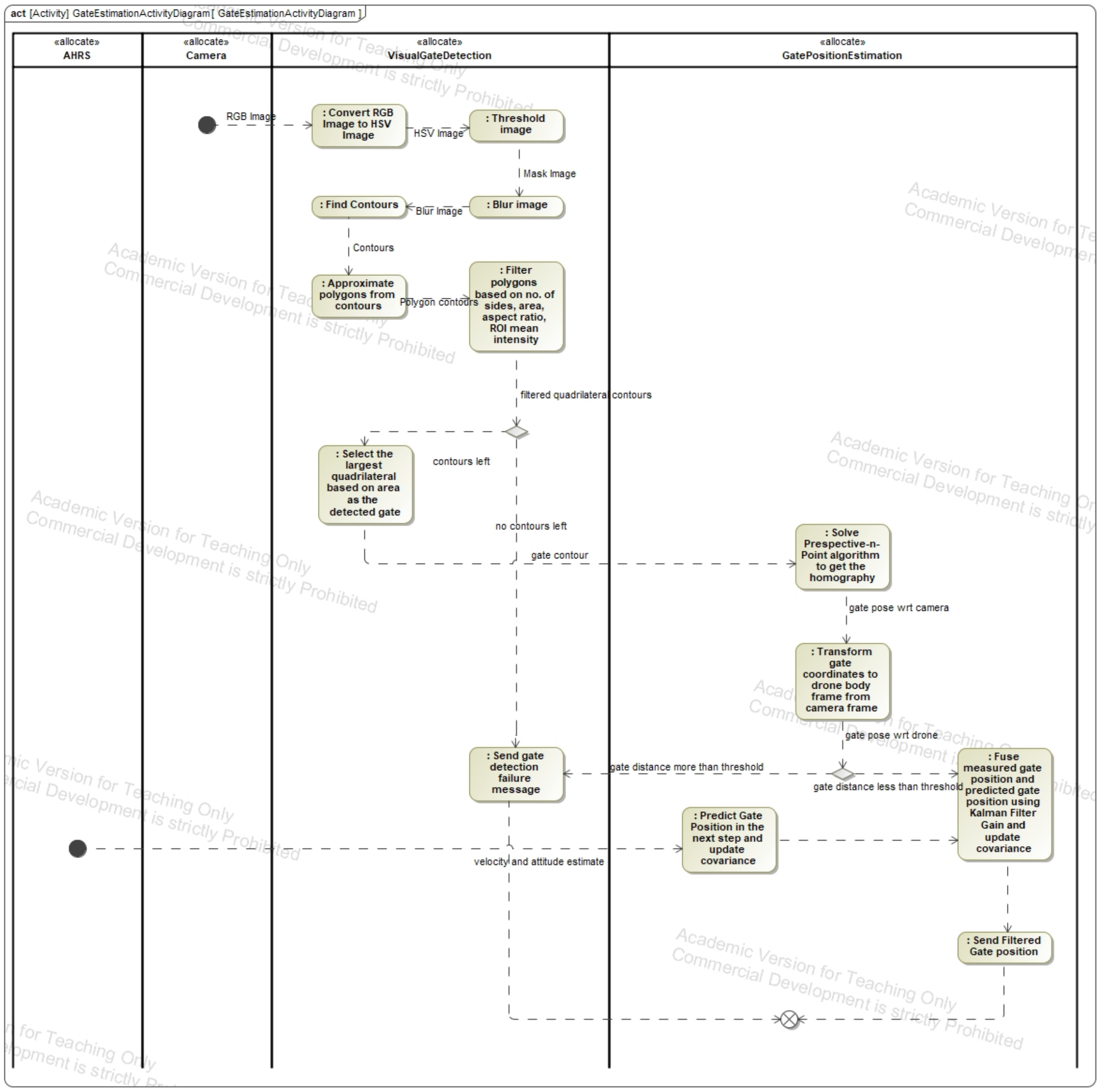
act [Activity] GateEstimationActivityDiagram [GateEstimationActivityDiagram]
«allocate» AHRS
«allocate» Camera
«allocate» VisualGateDetection
«allocate» GatePositionEstimation
RGB Image
: Convert RGB Image to HSV Image
HSV Image
: Threshold image
Mask Image
: Blur image
Blur Image
: Find Contours
Contours
: Approximate polygons from contours
Polygon contours
: Filter polygons based on no. of sides, area, aspect ratio, ROI mean intensity
filtered quadrilateral contours
contours left
: Select the largest quadrilateral based on area as the detected gate
no contours left
gate contour
: Solve Prespective-n-Point algorithm to get the homography
gate pose wrt camera
: Transform gate coordinates to drone body frame from camera frame
gate pose wrt drone
gate distance more than threshold
gate distance less than threshold
: Send gate detection failure message
: Fuse measured gate position and predicted gate position using Kalman Filter Gain and update covariance
: Predict Gate Position in the next step and update covariance
velocity and attitude estimate
: Send Filtered Gate position

Figure 2.9: ARDS Gate Estimation Activity Diagram

The drone position in the inertial frame is then calculated using the landmark position and estimated gate position in the drone body frame. The localization module is presented in detail in section 3.5 of this thesis.

2.8.3 Mission Control Activity Diagram

Figure 2.11 describes the Mission Control Activity Diagram. The Mission Control module is responsible for high-level mission control of the ARDS. It receives the pre-determined path in the form of a list of waypoints from the user. The Mission Control sends the current waypoint reference to the control module so that the control action can be computed. There are two types of waypoint change conditions viz. 1) Time-based and 2) Distance-based. The Mission Control continuously updates the time elapsed and the distance error for the current waypoint reference. The next waypoint in the list is set to the current waypoint if the time elapsed is more than threshold time or the distance error is within threshold distance based on the change-type of the current waypoint.

2.8.4 Control Activity Diagram

The control subsystem activity diagram is presented in figure 2.12. The control module receives the desired position, velocity and heading from the Mission control module. The control module also receives the current position, velocity, and heading of the drone. The control module then formulates a non-linear program with constraints and cost function based on the received data. This non-linear program

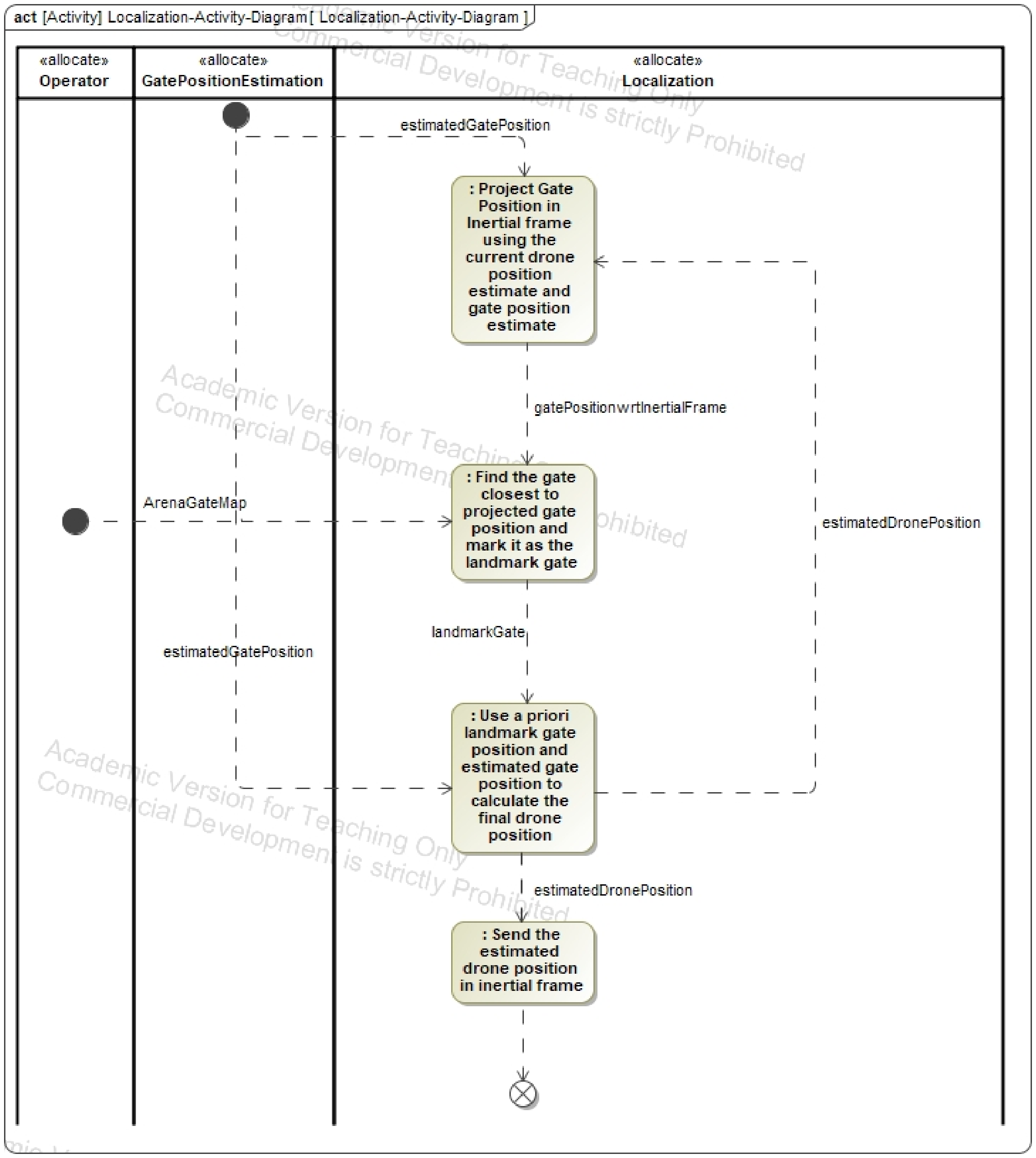
act [Activity] Localization-Activity-Diagram[Localization-Activity-Diagram]
«allocate» Operator
«allocate» GatePositionEstimation
«allocate» Localization
estimatedGatePosition
: Project Gate Position in Inertial frame using the current drone position estimate and gate position estimate
gatePositionwrtInertialFrame
ArenaGateMap
: Find the gate closest to projected gate position and mark it as the landmark gate
estimatedDronePosition
landmarkGate
estimatedGatePosition
: Use a priori landmark gate position and estimated gate position to calculate the final drone position
estimatedDronePosition
: Send the estimated drone position in inertial frame

Figure 2.10: ARDS Localization Activity Diagram

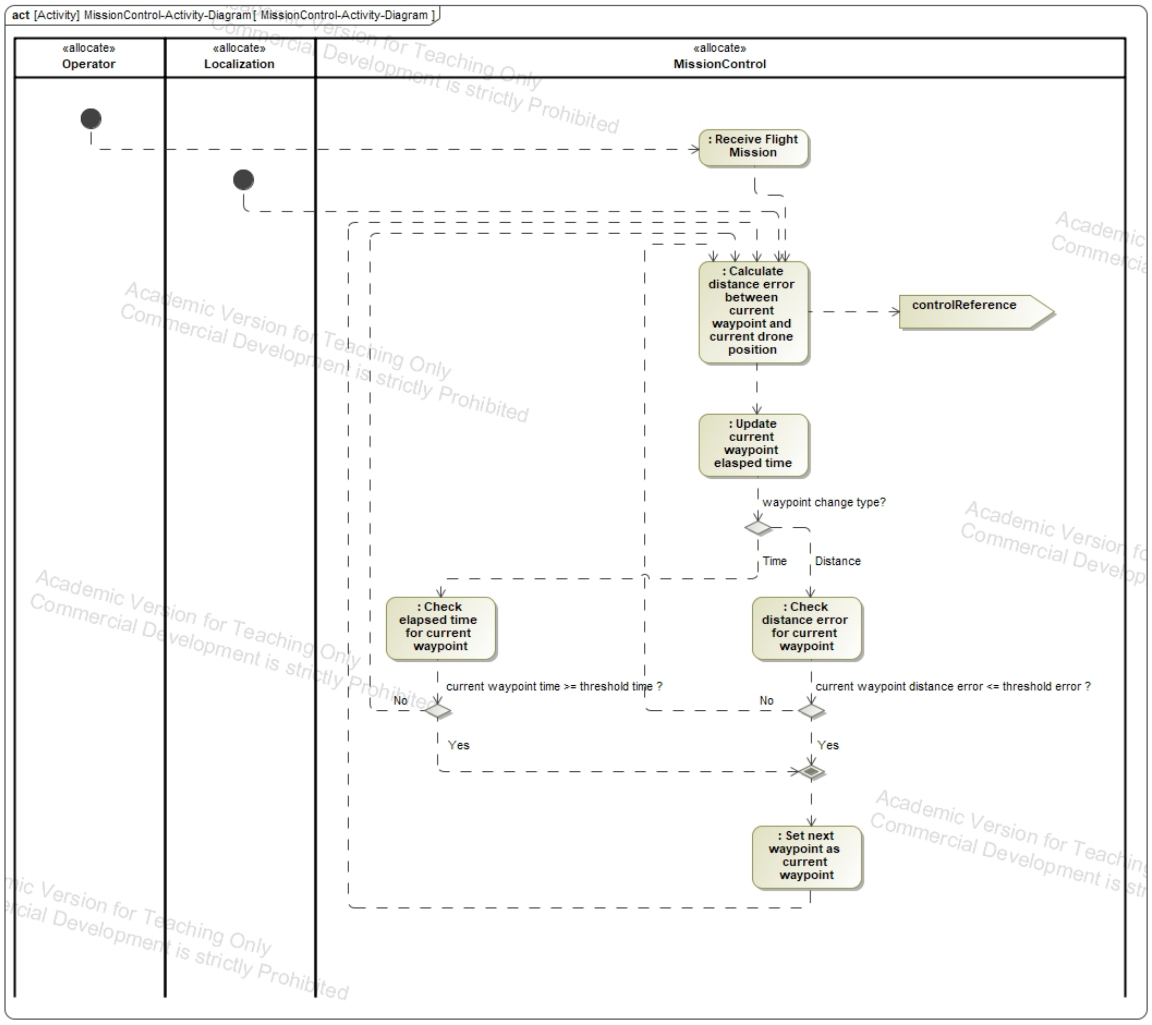
act [Activity] MissionControl-Activity-Diagram[MissionControl-Activity-Diagram]
«allocate» Operator
«allocate» Localization
«allocate» MissionControl
: Receive Flight Mission
: Calculate distance error between current waypoint and current drone position
controlReference
: Update current waypoint elasped time
waypoint change type?
Time
Distance
: Check elapsed time for current waypoint
: Check distance error for current waypoint
current waypoint time >= threshold time ?
current waypoint distance error <= threshold error ?
No
Yes
No
Yes
: Set next waypoint as current waypoint
Academic Version for Teaching Only
Commercial Development is strictly Prohibited

Figure 2.11: ARDS Mission Control Activity Diagram

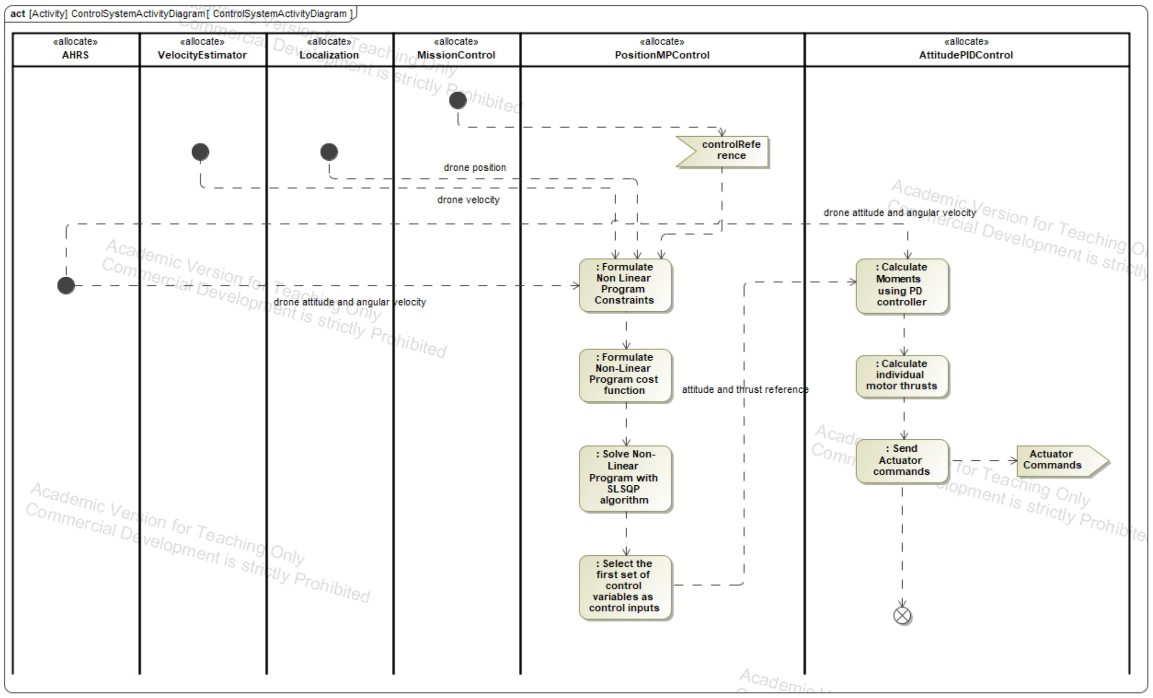
act [Activity] ControlSystemActivityDiagram [ControlSystemActivityDiagram]
«allocate» AHRS
«allocate» VelocityEstimator
«allocate» Localization
«allocate» MissionControl
«allocate» PositionMPControl
«allocate» AttitudePIDControl
controlReference
drone position
drone velocity
drone attitude and angular velocity
drone attitude and angular velocity
: Formulate Non Linear Program Constraints
: Formulate Non-Linear Program cost function
: Solve Non-Linear Program with SLSQP algorithm
: Select the first set of control variables as control inputs
attitude and thrust reference
: Calculate Moments using PD controller
: Calculate individual motor thrusts
: Send Actuator commands
Actuator Commands

Figure 2.12: ARDS Control System Activity Diagram

is solved to determine the states and control inputs which minimize the quadratic cost function. The desired roll, pitch and thrust control inputs are then sent to the attitude controller for low-level control. The control module is presented in detail in Chapter 4 of the thesis.

Chapter 3: Visual Estimation and Localization of the Autonomous Racing Drone System

3.1 Overview of Visual Estimation module

This work proposes a gate detection based visual-inertial localization (VIL) system which estimates the position of a visual gate in drone body frame and localizes the drone with respect to the gate in the inertial frame. Figure 3.1 shows the block diagram of the estimation subsystem.

Raw gate position measurements from the gate detection and measurement module are filtered using a Linear Time-Variant Kalman Filter. The prediction step in the Kalman Filter utilizes linear and angular velocity estimates and attitude esti-

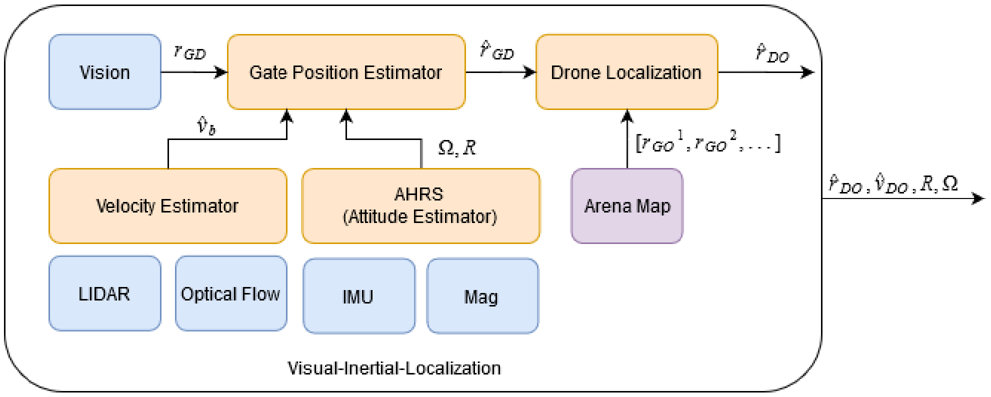

Figure 3.1: Visual-Inertial-Localization Block Diagram

mates to predict the gate position for estimation. Then, this gate position estimate is used to localize the drone in the arena map.

3.2 Background in Monocular Visual Inertial Odometry

The problem of vision-based position estimation for quadrotors has been studied quite extensively [15]. Due to their low weight and low power requirements, Monocular Visual-Inertial-Odometry (VIO) systems have become a popular choice for state estimation in GPS denied environments for autonomous aerial systems. Mourikis, A. and Roumeliotis, S. [16] presented an Extended Kalman Filter based visual-inertial navigation algorithm called “Multi-State Constraint Kalman Filter” (MSCKF) VIO. They derived a measurement model to express the geometric constraint that arises when a static feature is observed by a moving camera with multiple poses. This eliminates the need for maintaining the 3D position of the static feature in the EKF state vector. Leutenegger, S. et al [17] developed “Keyframe-Based visual-inertial SLAM” which utilizes non-linear optimization on a sliding window of keyframe poses. The cost function for optimization is a weighted sum of reprojection errors for visual landmarks and inertial errors. The landmarks are identified using Harris corner detectors [18] and BRISK descriptors [19]. Forster, C. et al [20] introduced a computationally light-weight visual-odometry system called “Semi-direct Visual Odometry” (SVO). SVO utilizes tracking of FAST corner features [21] in images and aligns them with the scene structure by minimizing the re-projection error using a non-linear least-squares optimization method.

Most of the generalized VIO methods are based on feature detection and tracking among image frames and fusion with inertial measurements from the IMU using optimization methods. Though these approaches provide a precise and robust estimation of the pose, they are often highly resource-intensive and unfit for applications on hardware with limited resources like the ones on Micro Aerial Vehicles (MAV). This work proposes a light-weight visual-inertial localization (VIL) system for autonomous racing drones which is based on drone racing gate measurements and optical-flow velocity estimates. The proposed method exploits a priori information available about the gate positions and efficient gate detection and measurement algorithm for low resource-intensive visual-odometry. It also uses a Linear Time-Variant Kalman Filter whose prediction and measurement steps are executed asynchronously. This provides a better position estimate in the presence of noisy and intermittent measurements.

3.3 Visual Gate Detection and Measurement

The vision system of the ARDS consists of two cameras: a forward-facing camera mounted on top of the drone platform for visual gate detection and measurement and a downward-facing camera for estimating the horizontal velocity through optical-flow.

Figures 3.2(a) through 3.2(f) depict the steps for visual gate detection once an image frame is received from the camera. First, the image is converted to Hue-Saturation-Value (HSV) format as shown in Figure 3.2(b). HSV image format makes it easier to detect illuminated objects like lights and LEDs in an image as the "Value" amount is very high for illuminated objects in the image. A binary threshold operation is applied to make only the illuminated objects in the image visible as shown in figure 3.2(c). This binary image is then blurred to remove noise in the image as shown in figure 3.2(d) and a find-contour operation is applied. A contour is the locus of equal intensity gradient points. By doing so, we get the boundaries of all illuminated objects in the image as shown in figure 3.2(e). Now, we need to identify the most prominent gate among these contours. We first apply the iterative end-point fit algorithm on the set of contour points to approximate regular polygons from the irregular shaped contours. Then we filter out contours that are not 4 sided and are smaller than a threshold contour area. We also filter out highly oblong shapes whose aspect ratio (ratio of width to height) is greater than 1.25 and less than 0.8. This way we get near square shaped contours which are better candidates for the gate. Lastly, we remove contours that are bounding high-intensity regions of the image. These, contours are more likely to represent light sources like ceiling lights, windows or doors. Finally, if any contours remain, we sort them according to their area and select the largest contour to be the most prominent gate as shown in figure 3.2(f). If there are no contours left, we conclude that there is no gate in the image and a gate detection failure message is sent by the program.

Once the gate contour is detected, the image coordinates of the 4 corners the contour are used to determine the relative rotation and translation of the gate with respect to the camera in the camera coordinate system using Perspective-n-Point (PnP) algorithm. The camera coordinate system ($\mathcal{C}$), image coordinate system and

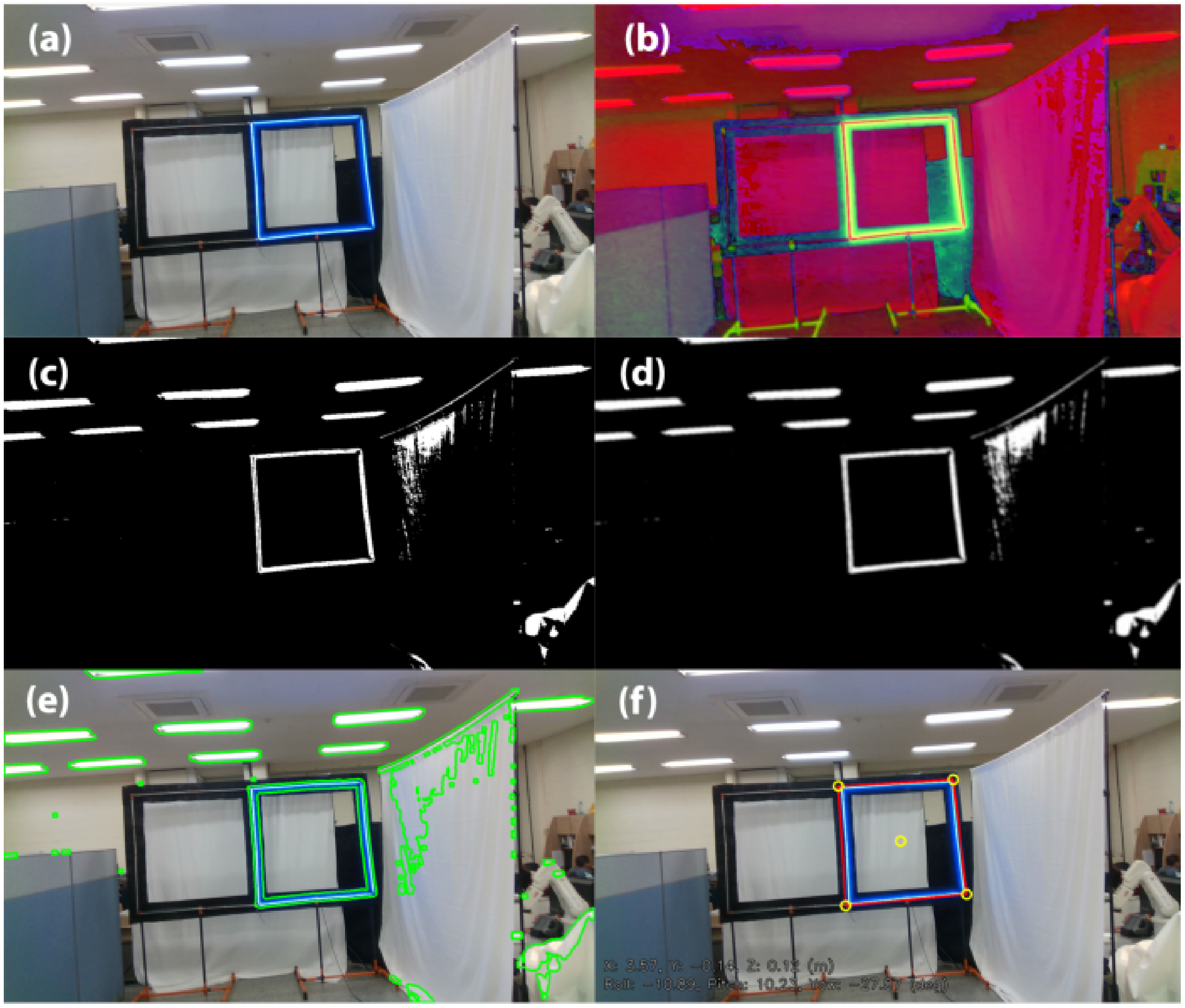
(a)
(b)
(c)
(d)
(e)
(f)
X: 3.57, Y: −0.14, Z: 0.12 (m)
Roll: −10.89, Pitch: 10.23, Yaw: −27.87 (deg)

Figure 3.2: Visual Gate Detection of ADR Gate, (a) RGB Image Captured by Front Camera, (b) HSV Image, (c) Binary Mask Image, (d) Blurred Mask Image, (e) Identified Contours, (f) Detected Gate

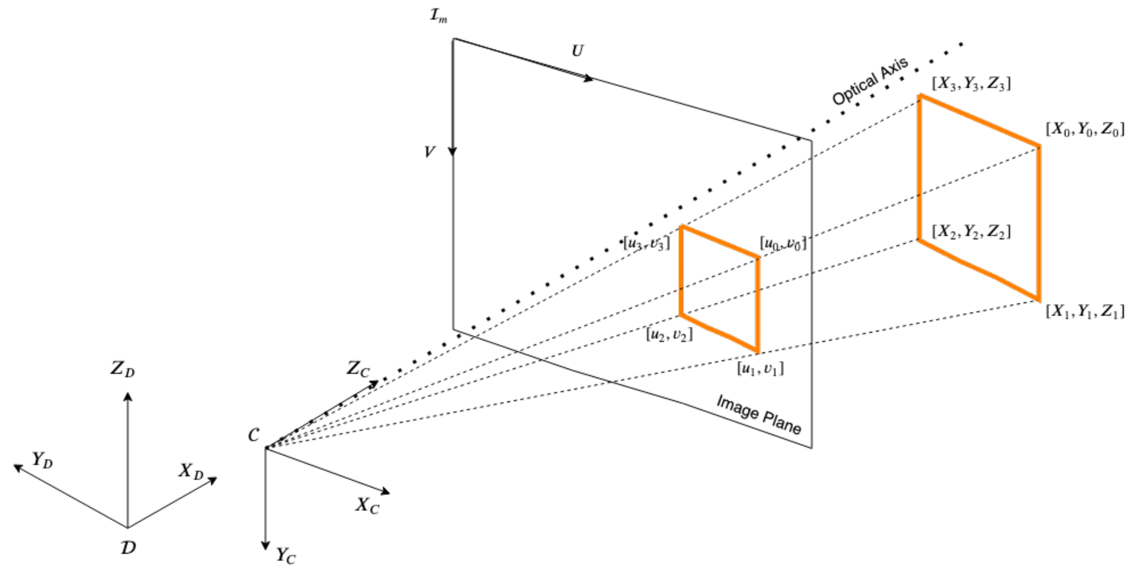

Figure 3.3: Gate Image Formation with Pinhole Camera Model

the drone coordinate system ($\mathcal{D}$) are described in figure 3.3. Image coordinates (u_i, v_i) of object points (X_i, Y_i, Z_i) have the following relationship:

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = M \begin{bmatrix} R_{3\times3} & T_{3\times1} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{3.1}$$

where, $M \in \mathbb{R}^{3\times3}$ is the camera matrix, $R_{3\times3} \in SO(3)$ is the relative orientation and $T_{3\times1} \in \mathbb{R}^3$ is the relative translation of the camera with respect to object coordinate system. The PnP algorithm solves equation (3.1) simultaneously for the 4 gate points to get the relative orientation and translation of the gate with respect to the camera.

The relative orientation and translation of the gate in the camera frame are then transformed to get the orientation and translation of the gate in the drone body frame.

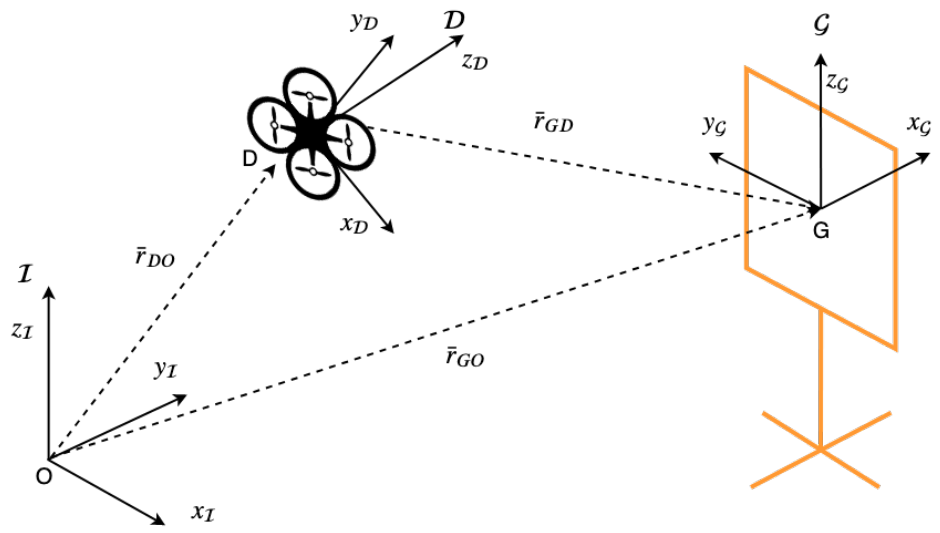

Figure 3.4: Inertial, Drone body and Gate Coordinate Frames with Position Vectors

3.4 Gate position estimation

3.4.1 Gate position dynamics

To estimate the gate position w.r.t. the drone body frame, there is a need to derive the gate position dynamics in the drone body frame. Figure 3.4 describes different coordinate frames and position vectors of the drone and the gate.

The relationship between the gate position vector w.r.t. drone ($r_{G/D}$), gate position vector w.r.t. ground ($r_{G/O}$) and drone position vector w.r.t. ground ($r_{D/O}$) is given by equation (3.2).

$$r_{G/O} = r_{G/D} + r_{D/O} \tag{3.2}$$

Differentiating equation (3.2) in the inertial frame w.r.t. time, we get the following relationship between the velocities (all expressed in the inertial frame):

$$[{}^{\mathcal{I}}v_{G/O}]_{\mathcal{I}} \;=\; [{}^{\mathcal{I}}v_{G/D}]_{\mathcal{I}} \;+\; [{}^{\mathcal{I}}v_{D/O}]_{\mathcal{I}} \tag{3.3}$$

Now, let the orientation of the drone frame w.r.t. the inertial frame be ${}^{\mathcal{I}}R_{\mathcal{D}} \in SO(3)$ and the angular velocity be ${}^{\mathcal{I}}\omega^{\mathcal{D}} = [\omega_1\;\omega_2\;\omega_3]^T$. Then, equation (3.3) can be written as:

$$[{}^{\mathcal{I}}v_{G/O}]_{\mathcal{I}} \;=\; {}^{\mathcal{I}}R_{\mathcal{D}}\;[{}^{\mathcal{I}}v_{G/D} \;+\; {}^{\mathcal{I}}v_{D/O}]_{\mathcal{D}} \tag{3.4}$$

Now using the transport equation, ${}^{\mathcal{I}}v_{G/D}$ can expressed as:

$$ {}^{\mathcal{I}}v_{G/D} \;=\; \frac{{}^{\mathcal{I}}d}{dt}(r_{G/D}) \;=\; \frac{{}^{\mathcal{D}}d}{dt}(r_{G/D}) \;+\; \Omega\; r_{G/D} \tag{3.5}$$

where,

$$\Omega = [{}^{\mathcal{I}}\omega^{\mathcal{D}}\times] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \in \mathfrak{so}(3) \tag{3.6}$$

We also know that the gates are stationary w.r.t. the inertial frame:

$$ {}^{\mathcal{I}}v_{GO} = 0 \tag{3.7}$$

Substituting equation (3.5) and equation (3.7) in equation (3.4), we get:

$$0 \;=\; {}^{\mathcal{I}}R_{\mathcal{D}} \left[\frac{{}^{\mathcal{D}}d}{dt}(r_{G/D}) \;+\; \Omega \, r_{G/D} \;+\; {}^{\mathcal{I}}v_{D/O} \right]_{\mathcal{D}} \tag{3.8}$$

Rearranging the terms in equation (3.8):

$$\frac{{}^{\mathcal{D}}d}{dt}(r_{G/D}) \;=\; -\,\Omega \, r_{G/D} \;-\; [{}^{\mathcal{I}}v_{D/O}]_{\mathcal{D}} \tag{3.9}$$

Equation (3.9) describes how the gate position evolves w.r.t. time in the drone body frame.

3.4.2 Asynchronous Linear Time-Variant Kalman Filter

A Linear Time-Variant Kalman Filter is utilized for the estimation of gate position with respect to the drone expressed in the drone body frame. The estimated state vector $x_k \in \mathbb{R}^3$ is the gate position $(r_{G/D})$ and the forcing vector $u_k \in \mathbb{R}^3$ is the velocity of the drone provided by the optical flow sensor $([{}^{\mathcal{I}}v_{D/O}]_{\mathcal{D}})$. The prediction step is based on the integration of gate position dynamics expressed by equation (3.9). Since, the nature of vision-based gate measurement is intermittent, the prediction and measurement steps are separated and executed asynchronously. This way, the estimation is not blocked when a measurement is not received in a time step. Figure 3.5 shows how the Kalman Filter steps are run asynchronously.

During the prediction process, it is assumed that the gate position and drone velocity vectors are random vector sequences. It is also assumed that the angular

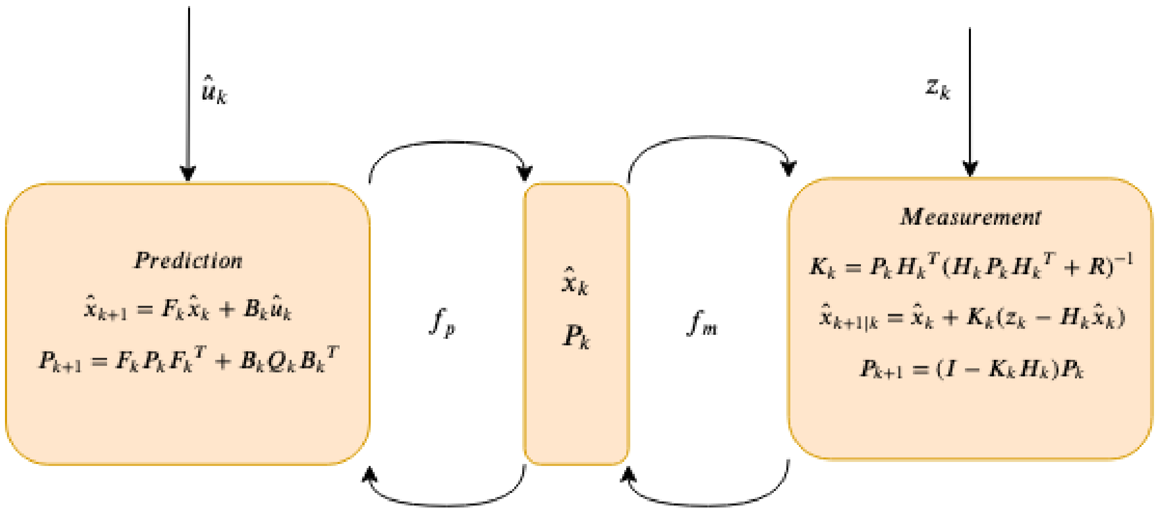
$\hat{u}_k$
z_k
Prediction
$\hat{x}_{k+1} = F_k\hat{x}_k + B_k\hat{u}_k$
$P_{k+1} = F_kP_kF_k^T + B_kQ_kB_k^T$
f_p
$\hat{x}_k$
P_k
f_m
Measurement
$K_k = P_kH_k^T(H_kP_kH_k^T + R)^{-1}$
$\hat{x}_{k+1|k} = \hat{x}_k + K_k(z_k - H_k\hat{x}_k)$
$P_{k+1} = (I - K_kH_k)P_k$

Figure 3.5: Asynchronous Linear Time-Variant Kalman Filter

velocity (${}^{\mathcal{I}}\omega^{\mathcal{D}}$) and orientation (${}^{\mathcal{I}}R_{\mathcal{D}}$) of the drone are deterministic. Thus, the state and control action can be modeled as the sum of expected value and white noise sequences:

$$x_k = \hat{x}_k + p_k \tag{3.10}$$

$$u_k = \hat{u}_k + q_k \tag{3.11}$$

where, p_k and q_k are white noise sequences with known covariance matrices P_k and Q_k respectively. And, $\hat{x}_k = E[x_k]$ and $\hat{u}_k = E[u_k]$ are the expected values of the state and control actions respectively. The discrete form of the dynamic equation (3.9) for a time step of Δt can be obtained by doing 1st order integration as following:

$$x_{k+1} = x_k + \ (-\Omega x_k \ - \ u_k)\Delta t \tag{3.12}$$

$$\therefore x_{k+1} = (\mathbb{I} \ - \Omega\Delta t)x_k \ + (- \ \mathbb{I}\Delta t)u_k \tag{3.13}$$

$$\therefore x_{k+1} = F_k x_k \; + \; B_k u_k \tag{3.14}$$

where, $\mathbb{I} \in \mathbb{R}^{3\times3}$ is the identity matrix, $F_k = (\mathbb{I} \; - \Omega\Delta t)$ and $B_k = (- \; \mathbb{I}\Delta t)$ The covariance of the estimated state error after the prediction step is updated as:

$$P_{k+1} = F_k P_k F_k^T + B_k Q_k B_k^T \tag{3.15}$$

The measured state from visual detection can be modeled as the measurement added with a white noise sequence:

$$z_k = H_k x_k + v_k \tag{3.16}$$

where, v_k is the white noise sequence with constant known covariance R and $H_k = \mathbb{I}$ which implies full state measurement.

Now, the updated state estimate can be calculated as a linear combination of previous state and measurement error:

$$\hat{x}_{k+1|k} = \hat{x}_k + K_k(z_k - H_k\hat{x}_k) \tag{3.17}$$

where, K_k is the Kalman Gain matrix. For optimal estimation [22], the Kalman Gain matrix is obtained from the following expression:

$$K_k = P_k H_k^T \; (H_k P_k H_k^T + R)^{-1} \tag{3.18}$$

And the estimation covariance is updated as:

$$P_{k+1} = (\mathbb{I} - K_k H_k)\ P_k \tag{3.19}$$

3.5 Localization

The localization of the drone is based on the reprojection of the measured gate position in the inertial frame and searching for the actual gate position on the arena map. The arena map is a list of gate positions and headings in series ($[r^0_{G/O}, r^1_{G/O}, ..., r^n_{G/O}]$). Figure 3.6 shows how the gate position is projected into the inertial frame for localization.

Initially, the estimated gate position with respect to drone ($[\hat{r}_{G/D}]_{\mathcal{D}}$) is used to calculate the expected gate position in the inertial frame ($[\hat{r}_{G/O}]_{\mathcal{I}}$).

$$[\hat{r}_{G/O}]_{\mathcal{I}} \ = \ {}^{\mathcal{I}}R_{\mathcal{D}}\ [\hat{r}_{G/D}]_{\mathcal{D}} \ + \ [r_{D/O}]_{\mathcal{I}} \tag{3.20}$$

Then, the gate which is closest to the projected gate position is chosen to be the landmark gate. This can be expressed as the following search problem:

$$\underset{n}{\text{minimize}} \quad \|[\hat{r}_{G/O}]_{\mathcal{I}} - [r^n_{G/O}]_{\mathcal{I}}\| \tag{3.21}$$

where, n is the index of a gate in the map and ($[r^n_{G/O}]_{\mathcal{I}}$) is the nth gate position in the map. Then, the estimated position of the drone in the inertial frame is calculated using the actual gate position.

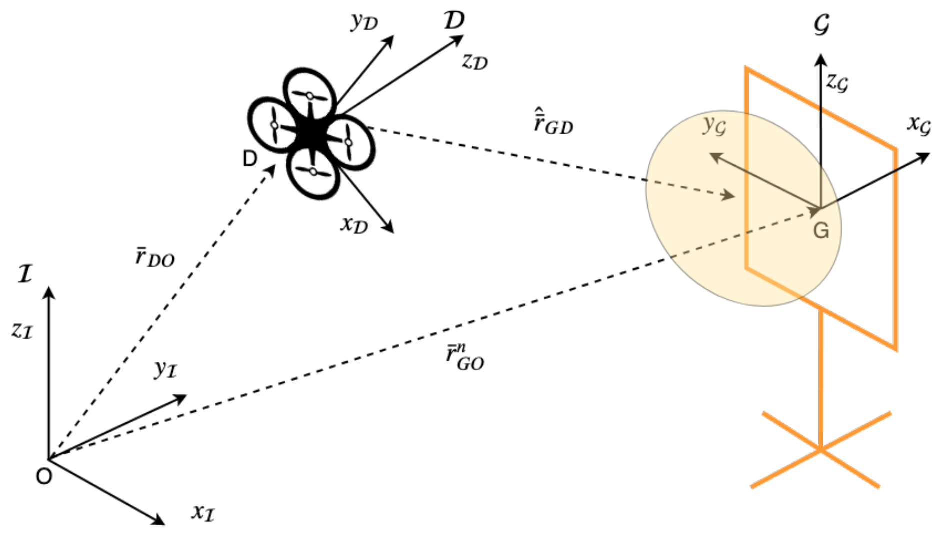

Figure 3.6: ARDS Localization

$$[\hat{r}_{D/O}]_{\mathcal{I}} \;=\; [r^{n}_{G/O}]_{\mathcal{I}} - {}^{\mathcal{I}}R_{\mathcal{D}}\,[\hat{r}_{G/D}]_{\mathcal{D}} \tag{3.22}$$

Since, $[r^{n}_{G/O}]_{\mathcal{I}}$ is deterministic, the covariance matrix for drone position estimation error becomes:

$$P'_k = {}^{\mathcal{I}}R_{\mathcal{D}}\;P_k\;{}^{\mathcal{I}}R^{T}_{\mathcal{D}} \tag{3.23}$$

where, P_k is the covariance matrix of gate position estimation error.

Chapter 4: Control of the Autonomous Racing Drone System

4.1 Overview of the Control System module

The ARDS Control System consists of three levels of control loops. The first level is the inner loop control or attitude control system which controls the attitude of the drone and the second level is the outer loop control or position control system which controls the position of the drone in the inertial frame. Finally, the third level of the control loop is the mission control which manages the navigation of the drone through the racing arena. A Non-Linear Model Predictive Controller (NMPC) is employed for the position control and PD controller for the attitude control of the drone. Figure 4.1 shows the control system of the ARDS.

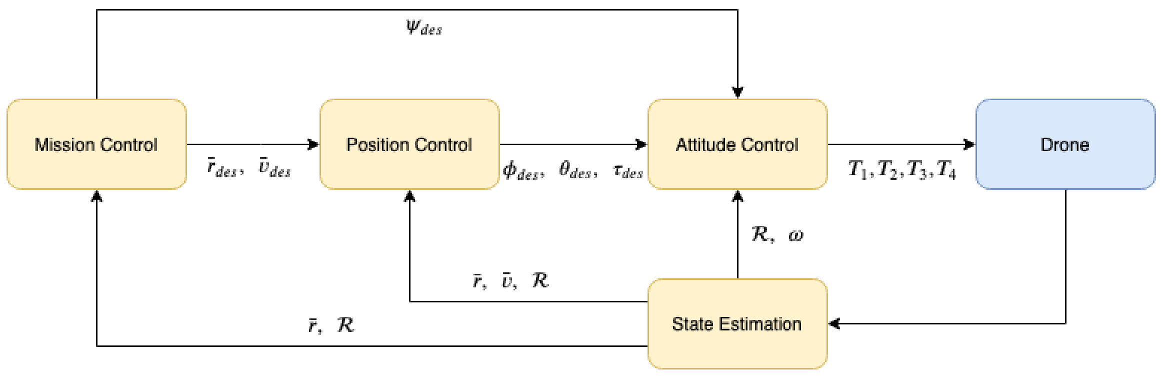

Figure 4.1: Control System Block Diagram of ARDS

4.2 Background in Non-Linear Model Predictive Control of Quadcopters

The early control algorithms for quadcopter control were based on the linearization of the quadcopter system model in the hover state by making small angle assumptions. But for exhibiting aggressive maneuvers a quadcopter is required to attain large roll and pitch angles which are well outside the linearization envelope of the linear models. For solving this problem, many research groups have proposed non-linear control strategies to enable control of quadcopter maneuvers in aggressive flight. Among these control strategies, Model Predictive Control (MPC) for unmanned aerial vehicles (UAV) has attracted a lot of attention in recent years. These strategies provide the major advantages over the other existing non-linear control strategies such as the ability to consider future states; flexibility in cost function selection for optimization; and facility to provide hard constraints on the state and control inputs of the system.

Tenny, Mathew and Wright [23] first described a trust-region feasibility-perturbed sequential quadratic programming (SQP) algorithm for non-linear model predictive control (NMPC). Slegers, Kyle and Costello [24] developed a NMPC strategy for autonomous aircraft with 6 degree-of-freedom (DOF) representations. They applied NMPC in the position control of simulated parafoil glider and fixed-wing aircraft. Though these ideas were known early on, practical implementations of NMPC in quadcopters were not yet carried out due to a limited amount of computational capabilities of the hardware. With the recent advancements of embedded computing hardware, it is possible to execute the lengthy online optimization computations onboard the quadcopter. Bangura and Mahony [25] presented a real-time MPC strategy for quadcopter control on resource constraint hardware. They linearized the non-linear model of a quadcopter and applied an unconstraint Linear MPC strategy to calculate the desired attitude of the quadcopter and then applied a Lyapunov based attitude control strategy. Neunert et al. [26] proposed an iterative optimal control algorithm (SQP) in a MPC setting to solve the underlying nonlinear control problem. Their method performs simultaneous trajectory planning and control for optimal trajectory tracking. They demonstrated this control method on a ball-balancing robot and a hexacopter UAV. Recently, Ru and Subbarao [27] used the idea of state-dependent coefficient factorization of nonlinear dynamics of a quadcopter to develop a pseudo-linear state-space model of the quadcopter. They proposed a NMPC method to solve for the optimal control problem of the derived pseudo-linear model and showed that it guarantees bounded errors and internal stability. Kamel, Burry and Siegwart [28] presented a full system model based NMPC for Micro Aerial Vehicle (MAV) trajectory tracking. They showed that their implementation of NMPC strategy was better in terms of hover performance, step response and aggressive trajectory tracking when compared with a classical Linear Model Predictive Controller (LMPC). They used an order 4 Runge-Kutta integration method to propagate the state and solved the discrete optimal control problem using SQP. Lunni et al. [29] developed a NMPC for 3D trajectory tracking of a quadcopter with a serial link manipulator. They showed that several aerial manipulation tasks can be achieved by either applying weighting strategies in the main optimization algorithm or using a hierarchical approach of nested optimization algorithms. Greeff and Schoellig [30] proposed a Flatness-Based NMPC for quadcopter trajectory tracking. Their approach can be applied to differentially flat nonlinear systems such as a quadcopter. The approach consists of a combination of feedback MPC and feedforward linearization. This makes the optimal control problem similar to solving a convex non-linear program. Falanga, D., et al. [12] proposed a "Perception Aware Model Predictive Control" for quadcopters with vision-based estimation. This approach optimizes state, control and perception objectives to execute point-to-point navigation. They utilized the pinhole camera model to formulate the perception objective function which is a weighted sum of squared position and velocity of image points of a visual feature. Their approach uses multiple shooting as transcription and a Runge-Kutta integration scheme for model propagation.

Though the existing NMPC formulations show promising quadcopter controller performance, they consider the entire state space of the quadcopter which is often unnecessary given the fact that a quadcopter is a differentially flat system. Also, most of the methods use 4th order Runge-Kutta integration method which provides very accurate prediction of the dynamics but requires more intermediate variables for the optimal control problem formulation. Since, the solution of a nonlinear program with SQP using n variables has a time complexity of $O(n^3)$ and space complexity of $O(n^2)$ [31], it is necessary to reduce the number of optimization variables as much as possible.

This work implements a NMPC method which exploits the differential flatness property of the quadcopter system by completely focusing on the position control of the quadcopter. It also uses a 1st order integration approach for model propagation. By doing so, the number of variables to optimize for the optimal control problem is significantly reduced which makes the control algorithm more efficient to run in terms of speed and memory usage, without affecting the overall controller performance.

4.3 Attitude Control

The attitude control of the quadcopter utilizes a Proportional-Derivative (PD) controller. The orientation of the quadcopter is expressed using Euler angles viz. roll (ϕ), pitch (θ) and yaw (θ) about the X, Y and Z axis respectively. The desired moments are calculated using PD control law as following:

$$\begin{aligned} M_x &= K_{p\phi}(\phi_{des} - \phi) - K_{d\phi}\omega_x \\ M_y &= K_{p\theta}(\theta_{des} - \theta) - K_{d\theta}\omega_y \\ M_z &= K_{p\psi}(\psi_{des} - \psi) - K_{d\psi}\omega_z \end{aligned} \tag{4.1}$$

where, M_i = required moments, K_{pi}= Proportional gains, K_{di} = Derivative gains and ω_i = angular velocities. Figure 4.2 shows the drone body frame, thrust vectors and direction of rotation of the propellers of the quadcopter.

Let the total thrust required be T, individual desired thrust be T_i and the arm length of the thrust vector from the origin be l. Then, the individual thrusts can

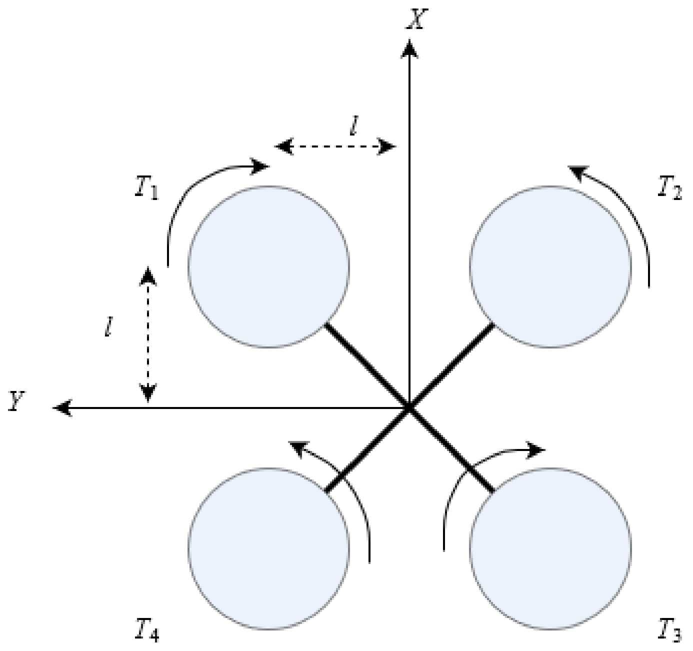

Figure 4.2: Propeller Configuration and Dimensions of the Drone

be calculated by solving the following system of equations:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ l & -l & -l & l \\ l & l & -l & -l \\ k_m & -k_m & -k_m & k_m \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} T \\ M_x \\ M_y \\ M_z \end{bmatrix} \tag{4.2}$$

where, $k_m = \frac{M_p}{T_p}$, which is the ratio of propeller drag moment to propeller thrust.

4.4 Position Control

An online, finite horizon, Non-Linear Model Predictive Control strategy is utilized for the outer loop or position control of the quadcopter. In the online Non-Linear Model Predictive Control, a cost function of state trajectory and control actions is optimized given the non-linear system model constraints and state as

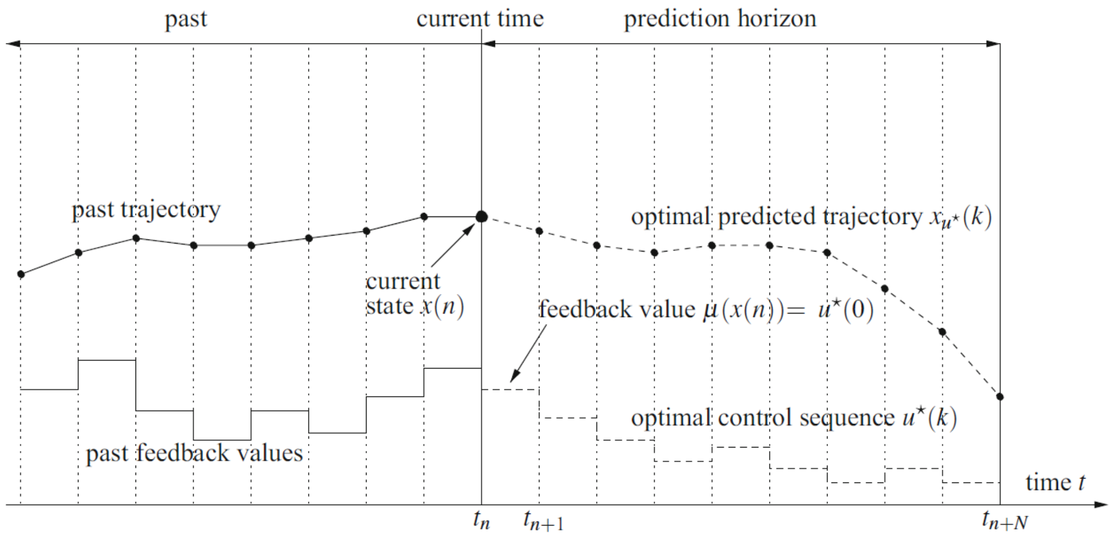
past
current time
prediction horizon
past trajectory
optimal predicted trajectory $x_{u^\star}(k)$
current state $x(n)$
feedback value $\mu(x(n)) = u^\star(0)$
optimal control sequence $u^\star(k)$
past feedback values
time t
t_n
t_{n+1}
t_{n+N}

Figure 4.3: Model Predictive Control [1]

well as control action bounds, over a fixed period in the future. This period is referred to as the prediction horizon for the NMPC as shown in figure 4.3. The NMPC optimization problem is formulated at each time step and the optimization is carried out iteratively using Sequential Quadratic Programming (SQP) [32] method for the fixed prediction horizon in future from the current time step. Hence it is also called receding horizon control strategy. NMPC provides many advantages over other control strategies. NMPC considers the future states of the system to make optimal control decisions which the other non-linear control strategies don't. Another advantage of NMPC is that hard constraints on the control action can be applied to change the aggressiveness of the quadcopter position tracking. This helps in adapting to different drone racing conditions quickly.

4.4.1 Model Development

Let us consider the position and velocity vectors associated with the quadcopter motion in the inertial frame to be $r = [x\ y\ z]^T$ and $v = [v_x\ v_y\ v_z]^T$ respectively. The orientation of the quadcopter is described by the rotation matrix $R \in SO(3)$ which is a function of Euler angles viz. ϕ, θ and ψ about the X, Y and Z axis respectively. The forces acting on the quadcopter include the thrust force by the propellers (T), gravitational force (F_g) and induced drag force (F_d) on the body due to motion. Thus, the equations of translational motion of the quadcopter in the inertial frame can be written as:

$$\dot{r} = v \tag{4.3}$$

$$m\dot{v} = R(\phi, \theta, \psi)\ T + F_g + F_d \tag{4.4}$$

$$\therefore \dot{v} = R(\phi, \theta, \psi) \begin{bmatrix} 0 \\ 0 \\ \tau \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} - c_d * v \tag{4.5}$$

where, $\tau = \frac{T}{m}$ is the specific thrust, c_d is the drag factors vector and $g = 9.8\, m/s^2$. In the above equation, the " * " operator signifies elementwise multiplication. Thus, the discrete dynamic equations of the quadcopter translation dynamics with sampling time Δt can be written as follows:

$$x_{i+1} = x_i + v_{x_i} \Delta t \tag{4.6}$$

$$y_{i+1} = y_i + v_{y_i} \Delta t \tag{4.7}$$

$$z_{i+1} = z_i + v_{z_i} \Delta t \tag{4.8}$$

$$v_{x_{i+1}} = v_{x_i} + (\ \tau_i\ (\ s\phi_i\ s\psi_i + c\phi_i\ s\theta_i\ c\psi_i\ ) - c_{d_x} v_{x_i}\ )\Delta t \tag{4.9}$$

$$v_{y_{i+1}} = v_{y_i} + (\ \tau_i\ (\ -s\phi_i\ c\psi_i + c\phi_i\ s\theta_i\ s\psi_i\ ) - c_{d_y} v_{y_i}\ )\Delta t \tag{4.10}$$

$$v_{z_{i+1}} = v_{z_i} + (\ \tau_i\ c\phi_i\ c\theta_i\ - g - c_{d_z} v_{z_i}\ )\Delta t \tag{4.11}$$

Now, because of the differential flatness property of the quadcopter the complete quadcopter state can be represented as:

$$\zeta = [x\ y\ z\ \psi]^T \tag{4.12}$$

So, the reference trajectory will be specified in terms of desired position, velocity and yaw of the quadcopter. Hence, the states and control inputs of the system for the optimal control problem formulation can be defined as:

$$X = [x\ y\ z\ v_x\ v_y\ v_z]^T \tag{4.13}$$

$$U = [\phi\ \theta\ \tau]^T \tag{4.14}$$

Thus, the non-linear model constraint equations from (4.6) to (4.11) can be expressed in a compact form as:

$$X_{i+1} = g(X_i, U_i, \psi) \tag{4.15}$$

4.4.2 Formulation of Optimal Control Problem

Let the prediction horizon in terms of number of future states be N. Then the NMPC problem can be expressed as the following non-linear program:

$$
\begin{aligned}
&\underset{X_i,\ U_i}{minimize} && \sum_{i=1}^{N-1}\left(\|X_i - X_{ref}\|^2_{Q_x} + \|U_i - U^*\|^2_{R_u} \right) + \|X_N - X_{ref}\|^2_{P_x} \\
&subject\ to: && X_{i+1} = g(X_i, U_i, \psi),\ i = 0,1,2,...,N-1 \\
& && X_0 = X(0) \\
& && |X_i| \leq X_{max},\ i = 0,1,2,...,N \\
& && |\theta_i,\ \phi_i| \leq \Theta_{max},\ i = 0,1,2,...,N \\
& && 0\ \leq\ \tau_i\ \leq\ \tau_{max},\ i = 0,1,2,...,N
\end{aligned}
\tag{4.16}
$$

Here, Q_x , R_u and P_x are state error penalties, control action penalties and terminal state error penalties respectively. The penalties must be positive for the cost function to be convex. X_{ref} is the reference state of the quadcopter and U^* is the control action for hover condition which implies, $\theta^* = 0$, $\phi^* = 0$ and $\tau^* = 9.8$. Θ_{max} is the maximum tilt angle (roll and pitch) and τ_{max} is the maximum specific thrust of the quadcopter allowed during trajectory tracking. This gives a facility to the operator of the racing drone to directly change the aggressiveness of the motion of the quadcopter by changing a few parameters. The aforementioned optimization problem is solved for every time step and the first set of control actions in the optimized set of variables is applied to the system. The optimization problem is solved using a gradient-based iterative method called Sequential Quadratic Programming [32] [33].

The optimizer in the NMPC is implemented using an open-source non-linear optimization package called NLopt [31] in C++. The required condition for the SQP algorithm to converge is that the cost function and constraints should be continuously differentiable [32]. The optimal control problem presented above satisfies these conditions and hence the iterative algorithm will converge to a local minimum.

Chapter 5: Simulation Results and Discussion

5.1 Simulation Setup

The simulation environment for the verification and validation of ARDS is based on open source Gazebo 7 physics simulation engine. The quadrotor dynamics simulation is developed by TU Munich [34]. This work uses a slightly modified version of the TUM simulator which implements the presented attitude PD controller. The ARDS system software is developed using the ROS C++ framework and open-source C++ libraries including OpenCV, BOOST UBLAS and NLopt. The open-source Kinect OpenNI gazebo plugin is used to simulate the onboard camera system on the quadrotor. The simulated drone racing arena consists of two orange gates separated by a distance of 8 m as shown in figure 5.1.

Table 5.1 lists the simulation parameters for ARDS evaluation and table 5.2 lists the system parameters for ARDS simulation.

Simulation parameter	Value	Unit
Quadrotor mass	800	g
Velocity Measurement Noise Variance	[0.01 0.01 0.01]	$(m/s)^2$
Gate Position Measurement Noise Variance	[0.1 0.1 0.1]	$(m)^2$
External Disturbance $(F_x,\ F_y,\ F_z)$	$\mathcal{N}(0,1)$	N

Table 5.1: ARDS Simulation Parameters

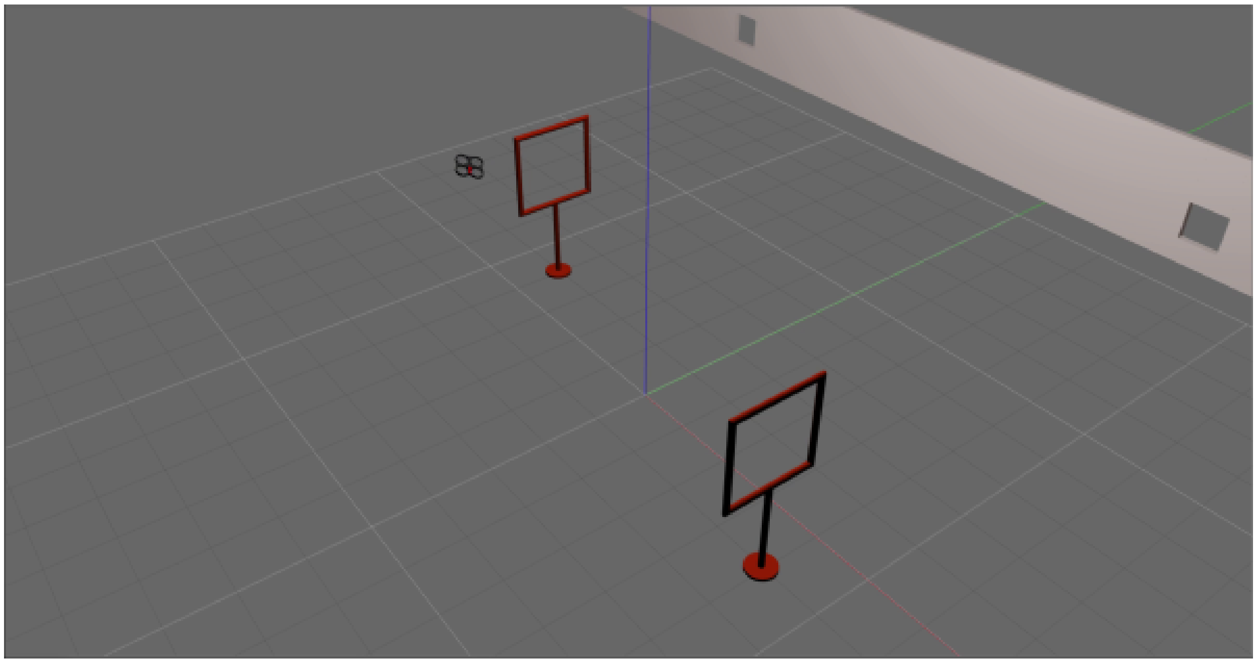

Figure 5.1: ARDS Simulation in Gazebo

5.2 Gate Position Estimation Results

The gate position estimation results for the simulation time from 4 s to 90 s are presented in figure 5.2. A magnified view of the results is also presented for the time from 4 s to 20 s in the figure 5.3. The actual gate position with respect to the drone is drawn with a dashed blue line, the estimated position is drawn with a green line and the measured gate position is drawn with ‘x’ markers. Gaussian noise of 0.1 m standard deviation is added to the visual gate measurement from the simulation to consider for the background noise in the image data. The regular sudden jumps seen in the estimated position are due to the change in the visual gate while traversing the racing arena.

Figure 5.5 shows the probability distribution of measurement error and esti-

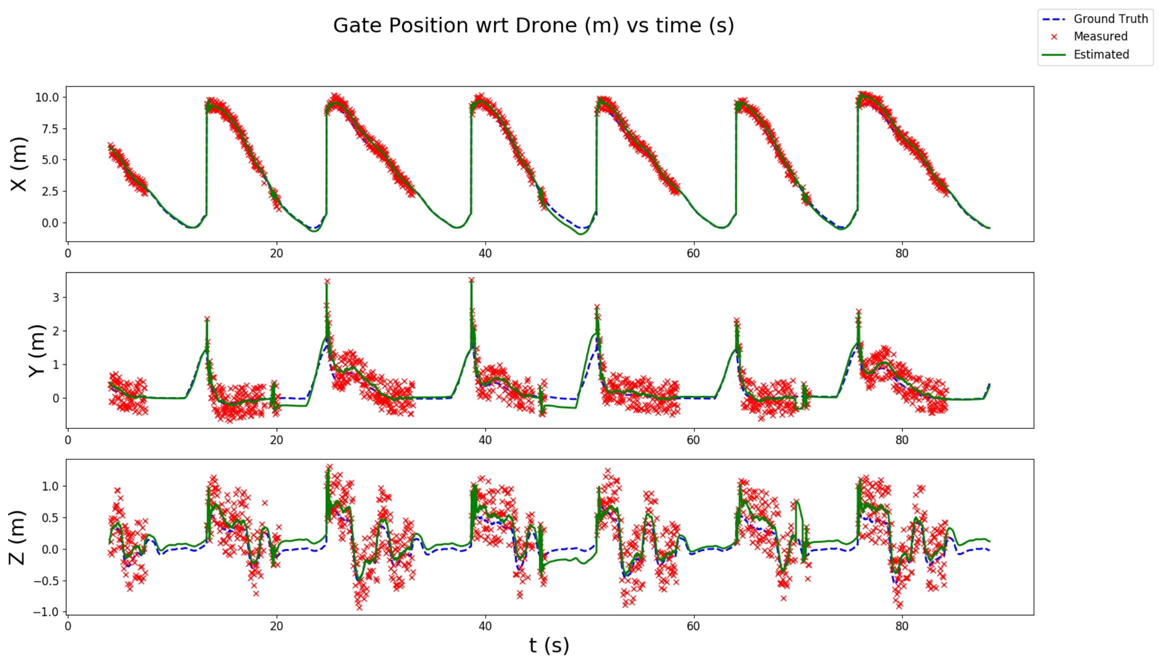

Figure 5.2: Gate Position w.r.t. Drone (m) vs time (s) [4s to 90s]

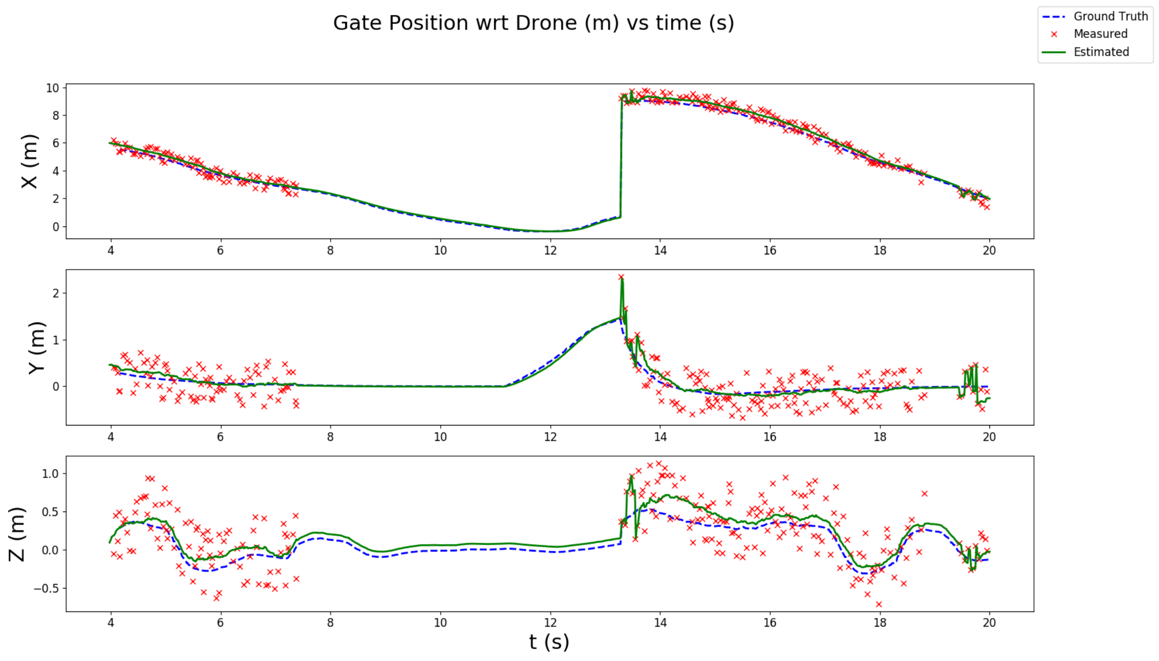

Figure 5.3: Gate Position w.r.t. Drone (m) vs time (s) [4s to 20s]

System parameter	Value	Unit
Gate Detection Parameters		
Gate side	1.5	m
High HSV Threshold	[40 255 255]	-
Low HSV Threshold	[0 100 100]	-
Contour Area Threshold	3000	px^2
Contour Aspect Ratio High Threshold	1.25	-
Contour Aspect Ratio Low Threshold	0.80	-
Contour Region Mean Intensity Threshold	100	-
Gate Position Estimation Parameters		
Measurement Noise Variance	[0.2 0.2 0.2]	$(m)^2$
Process Noise Variance	[0.5 0.5 0.5]	$(m)^2$
Measurement Frequency (f_m)	30	Hz
Prediction Frequency (f_p)	30	Hz
NMPC Parameters		
State Error Penalties ($Q_x = [x,\ y,\ z,\ v_x,\ v_y,\ v_z]$)	[1.0 1.0 1.0 0.5 0.5 0.5]	-
Control Action Penalties ($R_u = [\phi,\ \theta,\ \tau]$)	[0.5 0.5 0.01]	-
Prediction Horizon	10	-
Time step	0.1	s
Controller Frequency	50	Hz
Max tilt angle (Θ_{max})	20	$degree$
Max specific thrust (τ_{max})	15	m/s^2

Table 5.2: ARDS System Parameters

mation error of the gate position with respect to the drone. It is apparent from the estimation error distribution results that the estimated position closely follows the ground truth even in the presence of measurement noise and intermittent measurements due to gate changes. The Root Mean Squared Error(RMSE) in estimation is 0.32 m and the estimation bias in X, Y and Z position is 0.14 m, 0.04 m and 0.09 m respectively. The standard deviation of the errors is 0.38 m, 0.14 m and 0.098 m respectively. The bias and standard deviation are significant in the X direction because the visual gate measurement is inaccurate for larger distances from the gate. This is because for larger distances the thickness of the gate boundaries in the camera image is less and the blurring operation for noise removal increases this thickness which leads to a slightly inaccurate measurement. But the measurement accuracy increases as the drone comes closer to a gate. Figure 5.4 shows the variance of the estimation error of the gate position with time. It can be observed that when the gate measurements are available the estimation error variance converges to a minimum and is bounded but when the measurements are unavailable the estimation variance increases linearly due to the reliance on prediction. If the arena gates are arranged in such a way that a gate is always in sight, then the estimation variance is guaranteed to be bounded. But when the gates are not arranged in a continuously visible manner, the boundedness of the variance is subject to how long it takes for the drone to have a gate in sight.

5.3 Localization Results

Figure 5.6 presents the localization results for localizing the drone in the inertial frame. In the figure the dashed blue line depicts the actual trajectory of the drone while the red line depicts the estimated trajectory of the drone by the proposed localization method. The localization is very accurate when the drone is near a gate, which is the direct implication of the results of the gate position estimation presented in the previous section.

Figure 5.7 shows the position vs time graph of the drone localization. Small

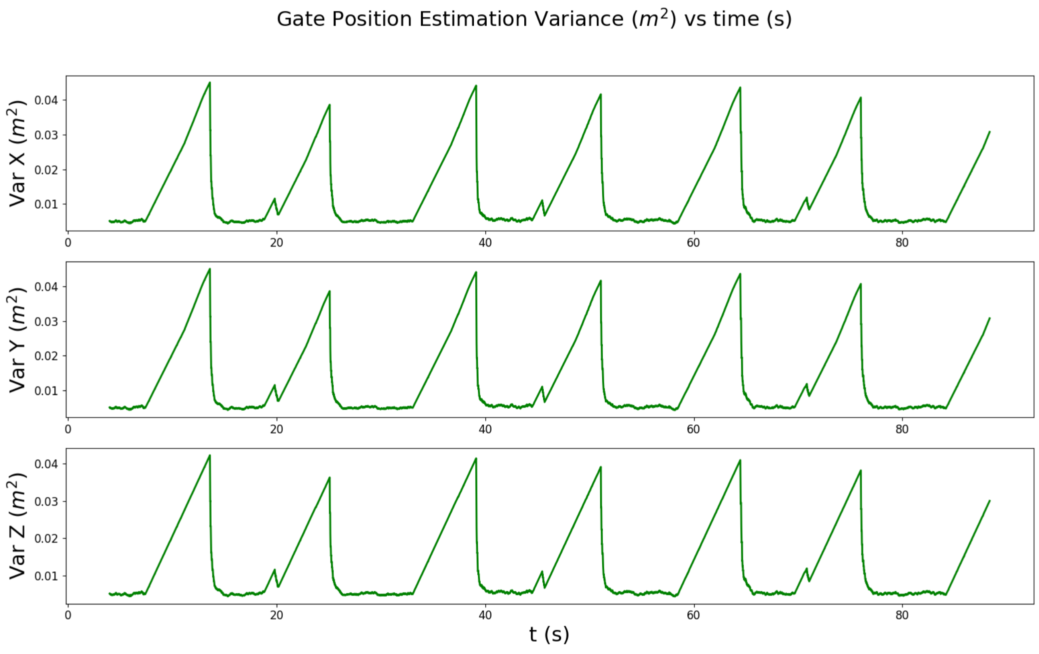

Figure 5.4: Gate Position Estimation Variance (m^2) vs time (s)

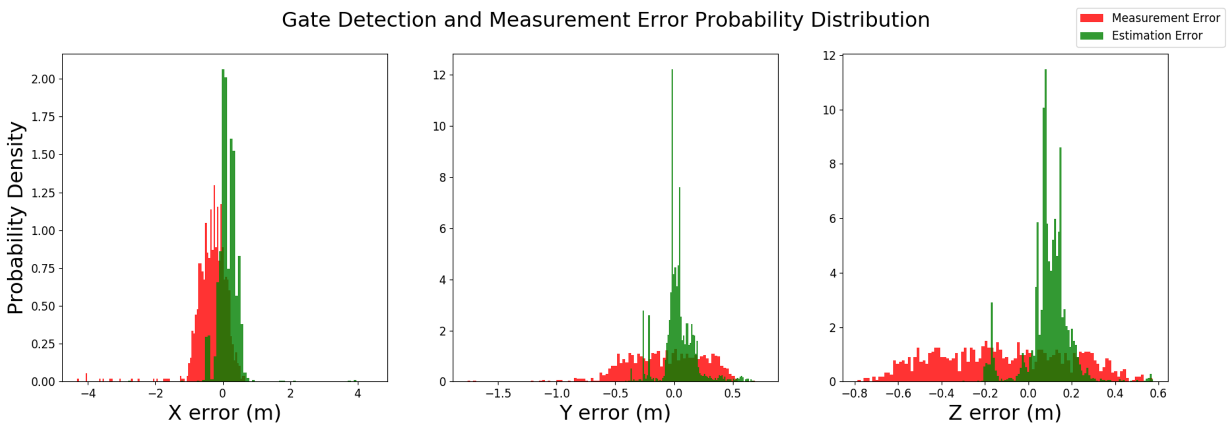

Figure 5.5: Gate Position Measurement and Estimation Error Probability Distribution

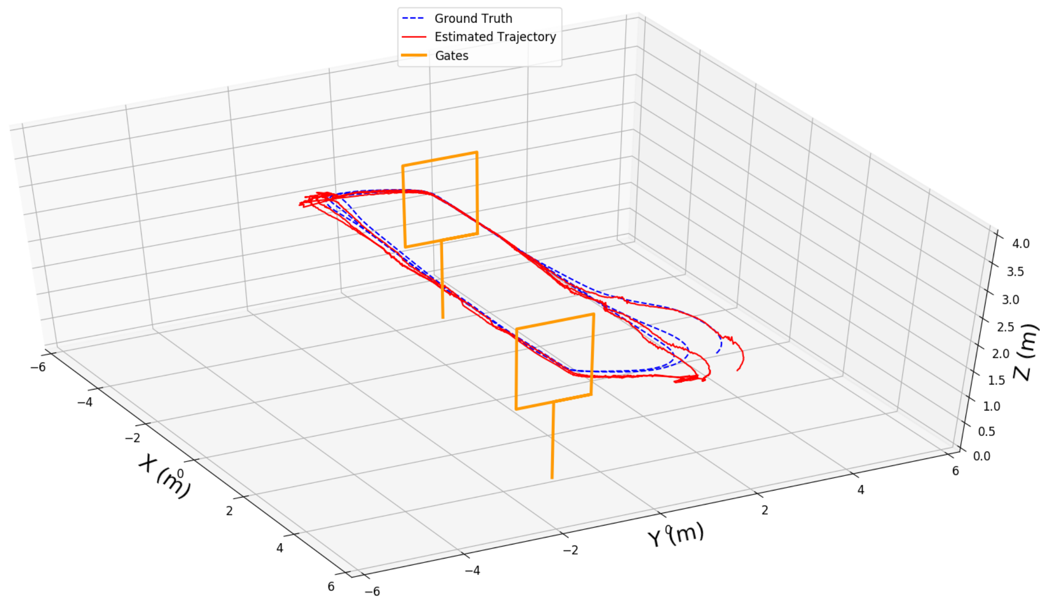

Figure 5.6: Estimated and Actual Trajectory of the Drone

jumps in the estimated trajectory can be observed in the figure when a gate is first visible. However, these jumps are present for a very short period of time and thus they do not affect the trajectory tracking performance and stability of the drone. Figure 5.8 shows the localization error histogram. The statistical measures for localization are similar to those of the gate position estimation statistics as they are linearly related to each other.

Figure 5.9 shows the comparison between different state-of-the-art monocular visual-inertial-odometry (VIO) systems [15] with the proposed VIL system. The average memory usage of the proposed VIL system is 60.55 MiB which is a very small fraction of what the state-of-the-art VIO systems utilize. This is because VIO systems use image feature detection, feature tracking and mapping along with non-linear optimization algorithms to get the camera pose while the VIL system exploits the information available regarding the gate positions and uses a linear time-variant

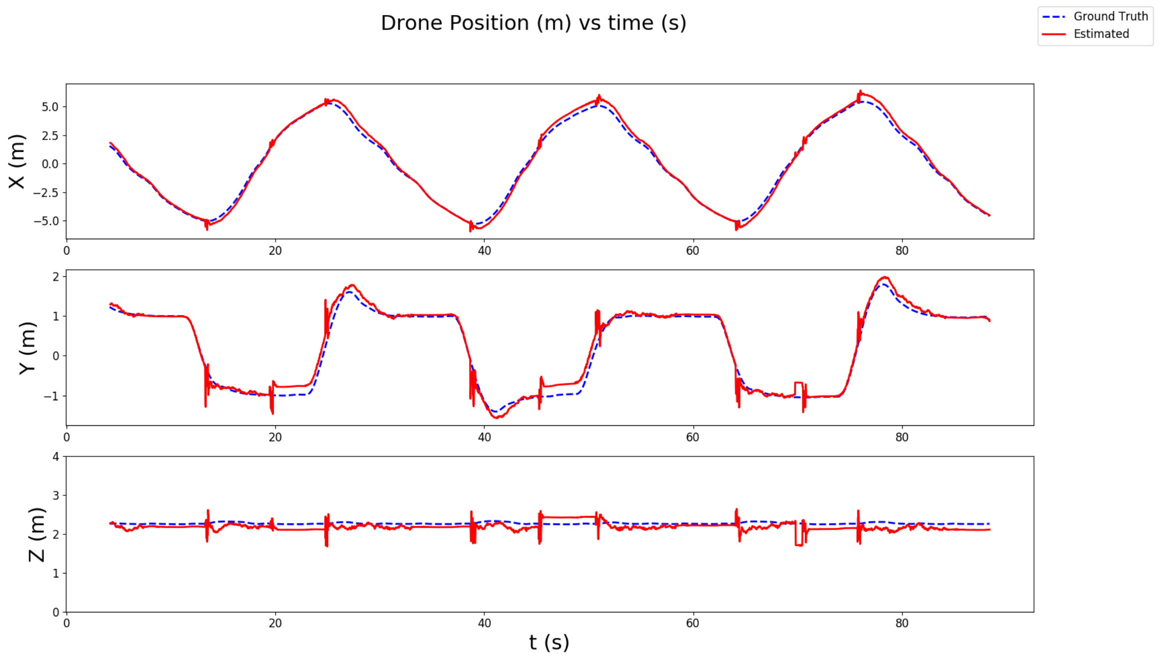

Figure 5.7: Drone Position (m) vs time (s)

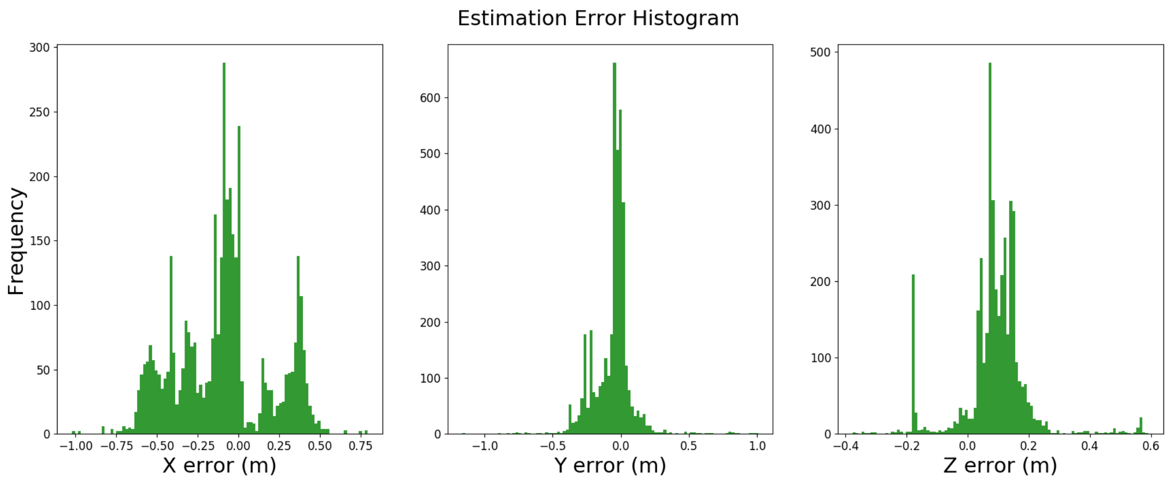

Figure 5.8: Drone Position Estimation Error Histogram

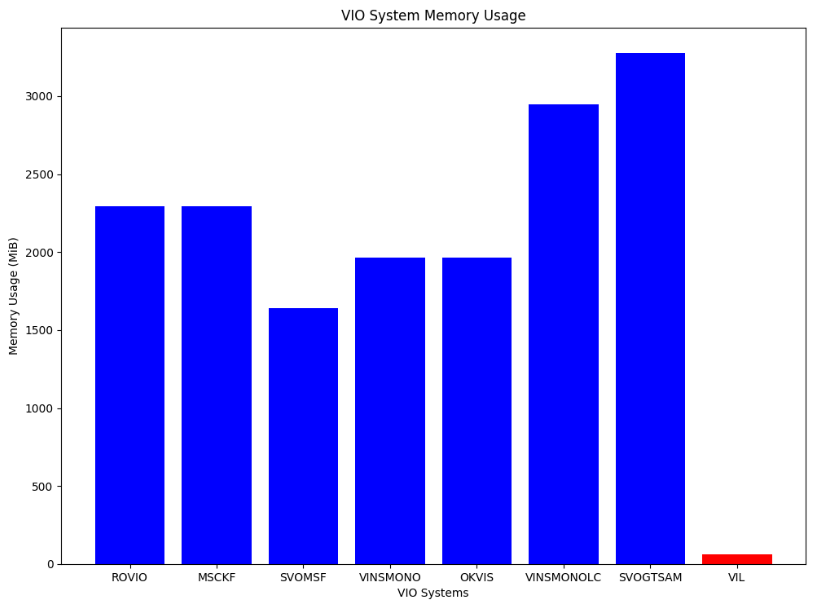

Figure 5.9: Monocular VIO Systems' Memory Usage Comparison

Kalman filter which requires a very small memory footprint. This makes VIL the best fit for use on resource constraint computing hardware present onboard a MAV for drone racing.

5.4 Control System Trajectory Tracking Results

The NMPC strategy presented in this work is compared with a PD controller for position and velocity trajectory tracking performance. The reference trajectory for the performance evaluation is a 5 m radius circular trajectory. The reference trajectory as a function time is given by:

$$\begin{aligned} x(t) &= Asin(2\pi ft) \\ y(t) &= Acos(2\pi ft) \\ z(t) &= 1.0 \\ v_x(t) &= 2\pi fAcos(2\pi ft) \\ v_y(t) &= -2\pi fAsin(2\pi ft) \\ v_z(t) &= 0.0 \\ \psi(t) &= 0.0 \\ A = 5\ m,&\ \ f = 0.1\ Hz \end{aligned} \tag{5.1}$$

Figure 5.10 shows the trajectory tracking performance of both NMPC and PD controller.

Figure 5.11 shows the position vs time plot and 5.12 shows the velocity vs time plot of the reference, NMPC and PD controller trajectories. Figure 5.13 shows the NMPC commands with time. It is important to notice the saturation of applied commands in ϕ_{des} and θ_{des} at 0.34 radians or 20 degrees. This is because the Θ_{max} parameter for the tilt angle bound is set to 0.34 radians.

Figures 5.14 and 5.15 show the position tracking error distribution and velocity tracking error distribution for the experiment. It can be observed that NMPC provides more precise trajectory tracking than a PD controller in terms of both position and velocity. This is due to the fact that NMPC controller takes into account the non-linear effects like a simplified induced drag model which a PD

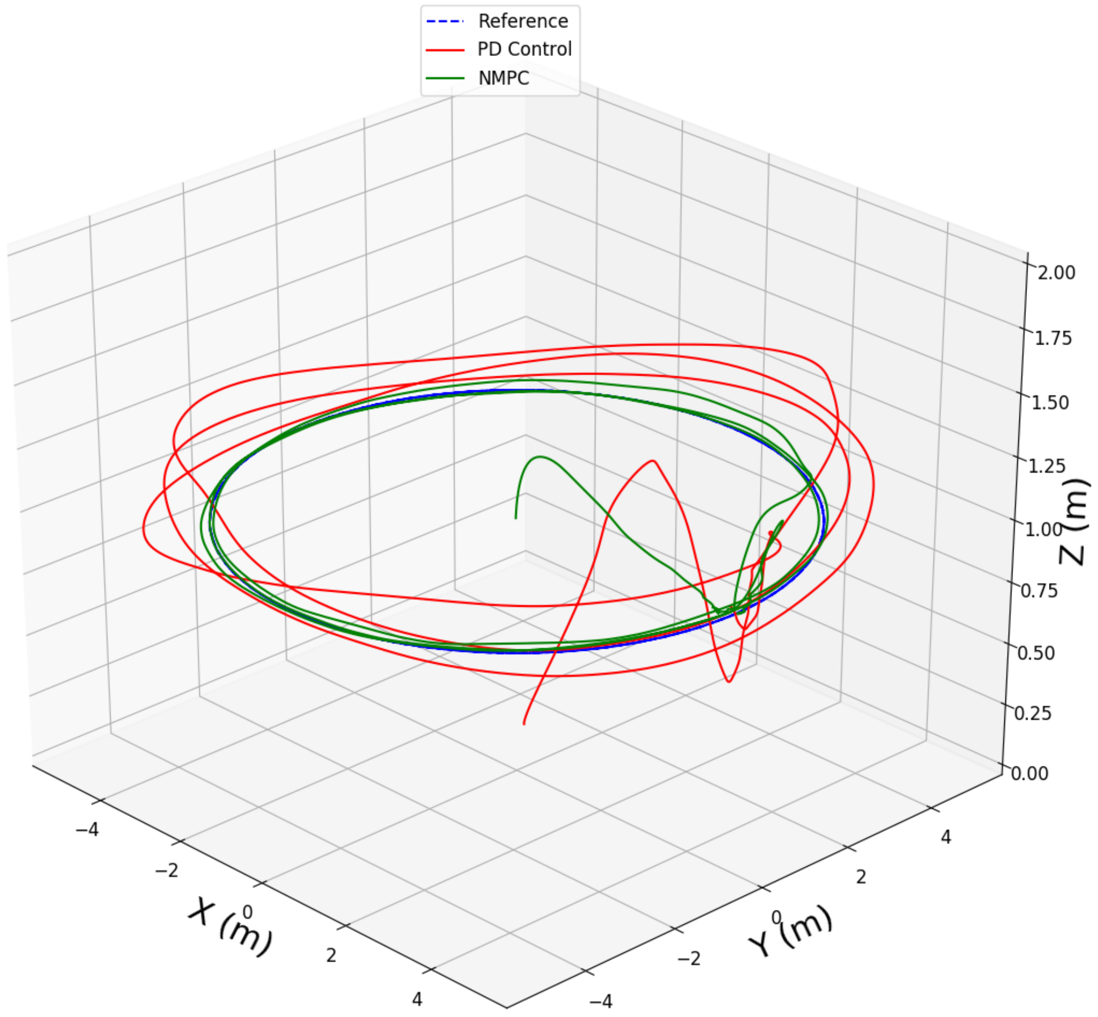

Figure 5.10: Circular Trajectory Tracking Performance

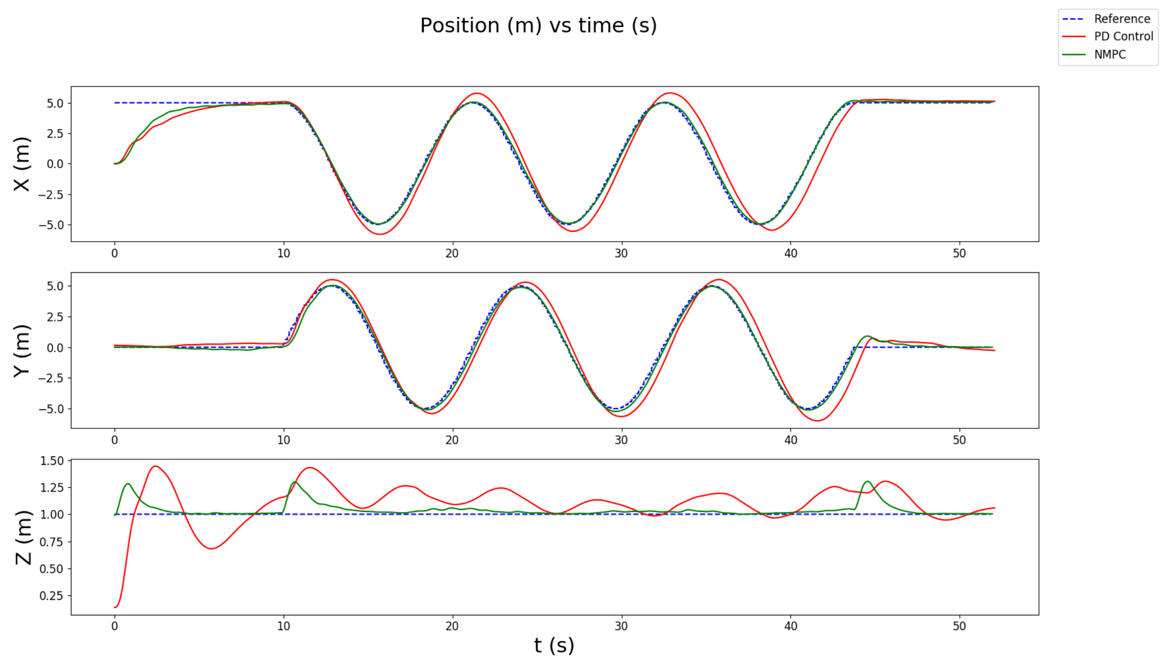

Figure 5.11: Drone Position (m) vs Time (s)

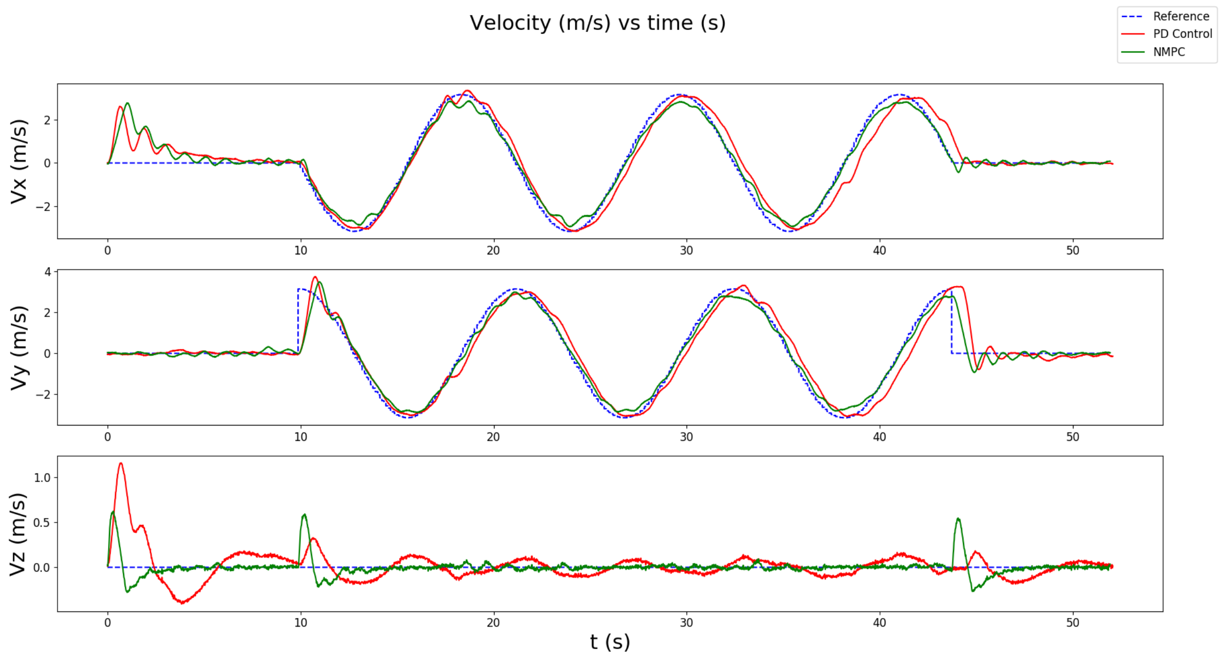

Figure 5.12: Drone Velocity (m/s) vs Time (s)

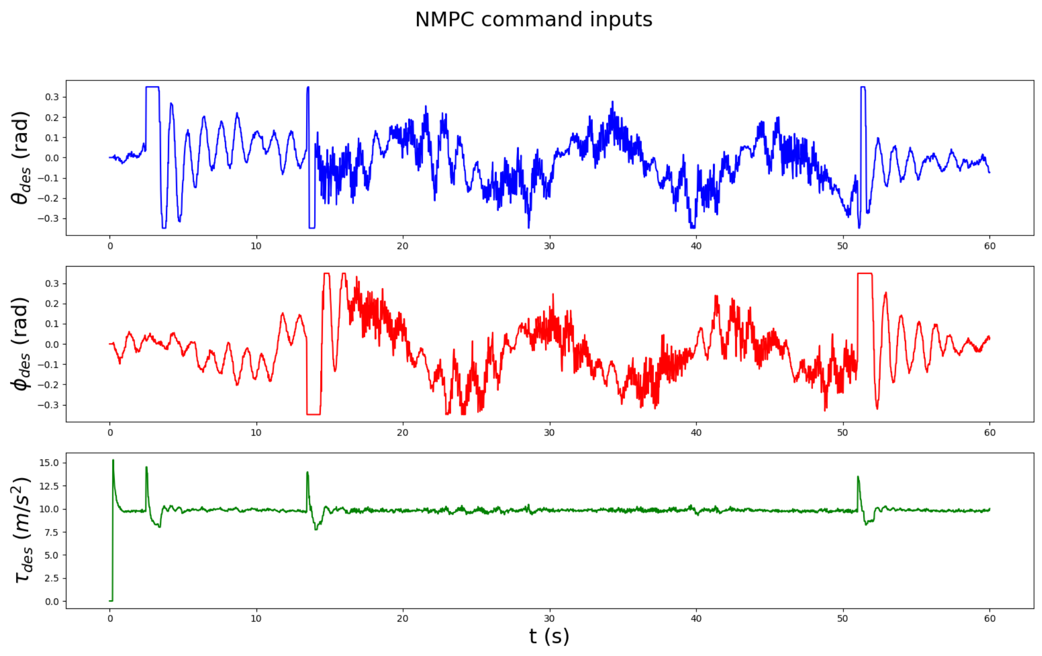

Figure 5.13: NMPC Command Inputs vs Time (s)

controller does not. Also, the X/Y and Z dynamics of the quadcopter are tightly coupled at higher tilt angles. This coupling is addressed by NMPC hence it provides a better tracking performance than a PD controller.

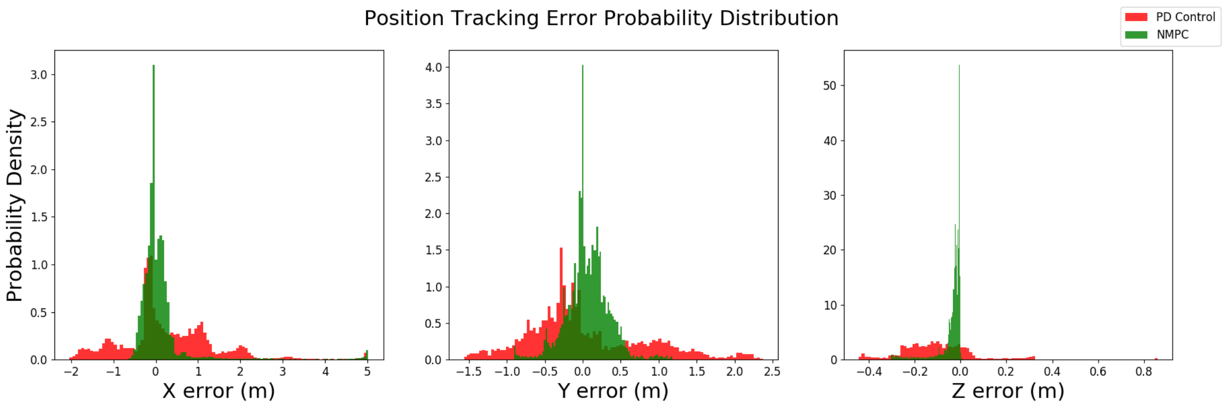

Figure 5.14: Position Tracking Error Probability Distribution

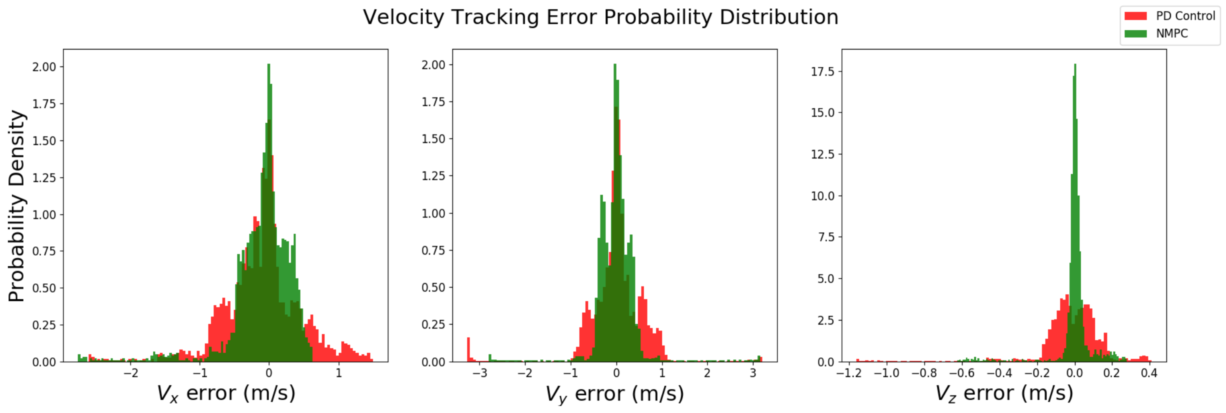

Figure 5.15: Velocity Tracking Error Probability Distribution

Chapter 6: Conclusion and Future Work

The system development of the Autonomous Drone Racing System (ARDS) was presented in this thesis. A semi-formal V-development Life Cycle Model for system development was employed to develop the system. Stakeholder and System Requirements were identified during the Requirements Definition phase. The structural and behavioral models of the system were developed during the Preliminary Design phase. Visual Gate Position Estimation, Localization and Control modules of the ARDS were implemented and verified in simulation using ROS Gazebo simulation. The visual estimation module of the ARDS, called Visual-Inertial-Localization(VIL) System, was validated in the IROS ADR 2019, Macau competition along with a backstepping PD controller. This iteration of the ARDS implementation won the second place in the IROS ADR 2019 competition. The VIL system of the ARDS utilizes an Asynchronous Linear Time Variant Kalman Filter, in which the prediction and measurement steps are executed asynchronously and independently. A Non-Linear Model Predicitve Control (NMPC) strategy was implemented for the Control module to improve the trajectory tracking performance of the ARDS as compared to the PD controller performance. The results of the VIL system performance evaluation showed that the system has promising position estimation capabilities even with noisy and intermittent visual measurements. It was also shown that the VIL system utilizes significantly less memory and computing resources than the state-of-the-art generalized Monocular Visual Inertial Odometry systems. The results of the Control module performance evaluation showed that the NMPC strategy has superior trajectory tracking performance as compared to a PD control strategy.

The ARDS software design was covered in this work. Though there is still more work needed to be done in the hardware design of the ARDS. The presented software sub-system was implemented on an existing commercially available drone platform. To improve the system performance further, a custom Micro Aerial Vehicle (MAV) platform is needed to be developed that is lighter in weight and has more powerful propulsion system. For the estimation of the visual gate position, it was assumed that the angular velocity and orientation of the drone are deterministic to make the prediction step linear time variant. The assumption that the angular velocity and orientation of the drone are stochastic will make the prediction step non-linear which will require non-linear estimation methods like Extended Kalman Filter, Unscented Kalman Filter or Particle Filter. The estimation results of the gate position estimation indicate that there is a small bias for larger distances from the gate. This bias can be estimated by including a bias term in the estimation model. Also, the intermittent nature of visual gate detection can be studied further and a probabilistic model of the gate detection can be created. A trajectory time cost can be added to the cost function in the NMPC's Non-Linear program formulation. This will generate time-optimal control actions for trajectory tracking.

Bibliography

[1] L. Grüne and J. Pannek, "Nonlinear model predictive control," in *Nonlinear Model Predictive Control*, pp. 45–69, Springer, 2017.

[2] N. J. Kasdin and D. A. Paley, *Engineering dynamics: a comprehensive introduction*. Princeton University Press, 2011.

[3] H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, G. de Croon, S. Hwang, S. Jung, H. Shim, H. Kim, M. Park, T. C. Au, and S. J. Kim, "Challenges and implemented technologies used in autonomous drone racing," *Intelligent Service Robotics*, vol. 12, no. 2, pp. 137–148, 2019.

[4] S. Jung, H. Lee, and D. H. Shim, "Real time embedded system framework for autonomous drone racing using deep learning techniques," *AIAA Information Systems-AIAA Infotech at Aerospace, 2018*, no. 209989, 2018.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 779–788, 2016.

[6] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, “A direct visual servoing-based framework for the 2016 IROS Autonomous Drone Racing Challenge,” *Journal of Field Robotics*, vol. 35, no. 1, pp. 146–166, 2018.

[7] S. Jung, S. Hwang, H. Shin, and D. H. Shim, “Perception, Guidance, and Navigation for Indoor Autonomous Drone Racing Using Deep Learning,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2539–2544, 2018.

[8] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, “Deep Drone Racing: Learning Agile Flight in Dynamic Environments,” no. CoRL, pp. 1–13, 2018.

[9] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, “Are we ready for autonomous drone racing? the UZH-FPV drone racing dataset,” *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, pp. 6713–6719, 2019.

[10] D. Mellinger and V. Kumar, “Minimum Snap Trajectory Generation and Control for Quadrotors-2011 mellingerICRA11.pdf,” pp. 2520–2525, 2011.

[11] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, “Beauty and the beast: Optimal methods meet learning for

drone racing," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, pp. 690–696, 2019.

[12] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-Aware Model Predictive Control for Quadrotors," *IEEE International Conference on Intelligent Robots and Systems*, pp. 5200–5207, 2018.

[13] S. Li, E. van der Horst, P. Duernay, C. De Wagter, and G. C. H. E. de Croon, "Visual Model-predictive Localization for Computationally Efficient Autonomous Racing of a 72-gram Drone," 2019.

[14] V. Frenzel, "Development of a highly reliable and efficient autonomous guidance system for unmanned aerial vehicles," Master's thesis, University of Stuttgart, 2019.

[15] J. Delmerico and D. Scaramuzza, "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2502–2509, 2018.

[16] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3565–3572, 2007.

[17] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart, "Keyframe-Based Visual-Inertial SLAM using Nonlinear Optimization," 2016.

[18] C. G. Harris, M. Stephens, *et al.*, "A combined corner and edge detector.," in *Alvey vision conference*, vol. 15, pp. 10–5244, Citeseer, 1988.

[19] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 International conference on computer vision*, pp. 2548–2555, Ieee, 2011.

[20] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*, pp. 15–22, IEEE, 2014.

[21] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 1, pp. 105–119, 2008.

[22] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[23] M. J. Tenny, S. J. Wright, and J. B. Rawlings, "Nonlinear model predictive control via feasibility-perturbed sequential quadratic programming," *Computational Optimization and Applications*, vol. 28, no. 1, pp. 87–121, 2004.

[24] N. Slegers, J. Kyle, and M. Costello, "Nonlinear model predictive control technique for unmanned air vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 5, pp. 1179–1188, 2006.

[25] M. Bangura and R. Mahony, *Real-time model predictive control for quadrotors*, vol. 19. IFAC, 2014.

[26] M. Neunert, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear Model Predictive Control for unified trajectory

optimization and tracking," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1398–1404, 2016.

[27] P. Ru and K. Subbarao, "Nonlinear model predictive control for unmanned aerial vehicles," *Aerospace*, vol. 4, no. 2, pp. 1–26, 2017.

[28] M. Kamel, M. Burri, and R. Siegwart, "Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017.

[29] D. Lunni, A. Santamaria-Navarro, R. Rossi, P. Rocco, L. Bascetta, and J. Andrade-Cetto, "Nonlinear model predictive control for aerial manipulation," *2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017*, pp. 87–93, 2017.

[30] M. Greeff and A. P. Schoellig, "Flatness-Based Model Predictive Control for Quadrotor Trajectory Tracking," *IEEE International Conference on Intelligent Robots and Systems*, pp. 6740–6745, 2018.

[31] S. G. Johnson, "The nlopt nonlinear-optimization package." `http://github.com/stevengj/nlopt`, 2014.

[32] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[33] D. Kraft, "Algorithm 733: Tomp–fortran modules for optimal control calculations," *ACM Transactions on Mathematical Software (TOMS)*, vol. 20, no. 3, pp. 262–281, 1994.

[34] H. Huang and J. Sturm, "Tum simulator ros package." `https://wiki.ros.org/tum_simulator`, 2018.