2022-08

False Data Injection of a Robotic Navigation System: Concepts and Simulations

Kersten, Philip

Monterey, California. Naval Postgraduate School

NPS-SP-22-010

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

FALSE DATA INJECTION OF A ROBOTIC NAVIGATION

SYSTEM: CONCEPTS AND SIMULATIONS

by

Philip Kersten

August 2022

Prepared for: Universitaet der Bundeswehr Muenchen

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE 08/26/2022	**2. REPORT TYPE** Technical Report	**3. DATES COVERED** START DATE 04/07/2022	END DATE 08/31/2022

4. TITLE AND SUBTITLE
False Data Injection of a Robotic Navigation System: Concepts and Simulations

5a. CONTRACT NUMBER	**5b. GRANT NUMBER**	**5c. PROGRAM ELEMENT NUMBER**
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
ENS Philip Kersten, B. Sc., German Navy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Guidance Navigation and Control Laboratory Department for Mechanical and Aerospace Engineering Naval Postgraduate School Monterey, CA, 93943	**8. PERFORMING ORGANIZATION REPORT NUMBER** NPS-SP-22-010

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Universitaet der Bundeswehr Muenchen Fakultaet fuer Luft- und Raumfahrttechnik Institute for Space Technology and Applications Werner-Heisenberg-Weg 39 85579 Neubiberg, Germany	**10. SPONSOR/MONITOR'S ACRONYM(S)** UniBw M	**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Statement A: Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES
Thesis Examiners: Univ.-Prof. Mag. Dr. habil. Thomas Pany (UniBwM), Prof. Dr. Mark Karpenko (NPS)

14. ABSTRACT
Today, robotic vehicles (RV) play an essential role in both modern industry and the military. More and more RVs become autonomously operated and are prone to cyber-attacks of their embedded control systems. These attacks can be detected and prevented with the appropriate tools but can also be almost undetectable if executed correctly.
In this work, a type of cyber-attack, namely false data injection is performed to spoof an Extended Kalman Filter (EKF) and a conventional fault detection mechanism is used to attempt to identify the attack. A two-wheeled differentially steered robotic vehicle based on the TurtleBot 3 ‘Burger’ is simulated in MATLAB. The navigation system is attacked by injecting false data into the measurement device - a LiDAR sensor. The false data injection alters the range and bearing measurements and aims to change the measured heading angle of the robot and alter its trajectory. The attack is then tuned to make it undetectable by the implemented counter measures.
In this thesis, it is demonstrated that false data injection can be executed in such a way that it is almost impossible for the fault detection mechanism to perceive an attack. The simulations have shown that the key to stealth is to inject the false data in a slow and steady manner to deceive the EKF by steering its output. The deviation introduced is mostly unnoticeable as long as it is in the same order of magnitude of the noise, which is part of the robot’s navigation system. In contrast, sudden and severe changes of measurement data are detected easily by the EKF and the fault detection mechanism.

15. SUBJECT TERMS
False Data Injection, Spoofing, Stealthy, Robotic Vehicle, Navigation, Extended Kalman Filter, Fault Detection

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**
a. REPORT unclassified	**b. ABSTRACT** unclassified	**C. THIS PAGE** unclassified	unclassified	72

19a. NAME OF RESPONSIBLE PERSON Philip Kersten	**19b. PHONE NUMBER** *(Include area code)*

THIS PAGE INTENTIONALLY LEFT BLANK

The report entitled "False Data Injection of a Robotic Navigation System: Concepts and Experiments" was prepared for "Universitaet der Bundeswehr Muenchen" funded by "German Armed Forces".

This report was prepared by:

ENS Philip Kersten, German Navy B.Sc.	Mark Karpenko Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

False Data Injection of a Robotic Navigation System: Concepts and Simulations

Master Thesis

Philip Kersten

matriculation number: 1185224

August 2022

Declaration of Authorship

I declare that the thesis has been composed by myself and that the work has not be submitted for any other degree or professional qualification. I confirm that the work submitted is my own, except where work which has formed part of jointly-authored publications has been included. My contribution and those of the other authors to this work have been explicitly indicated below. I confirm that appropriate credit has been given within this thesis where reference has been made to the work of others.

Eigenständigkeitserklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Monterey, 31.08.2022

Philip Kersten

Abstract

Today, robotic vehicles (RV) play an essential role in both modern industry and the military. More and more RVs become autonomously operated and are prone to cyber-attacks of their embedded control systems. These attacks can be detected and prevented with the appropriate tools but can also be almost undetectable if executed correctly.

In this work, a type of cyber-attack, namely false data injection is performed to spoof an Extended Kalman Filter (EKF) and a conventional fault detection mechanism is used to attempt to identify the attack. A two-wheeled differentially steered robotic vehicle based on the *TurtleBot 3 'Burger'* is simulated in *MATLAB.* The navigation system is attacked by injecting false data into the measurement device - a LiDAR sensor. The false data injection alters the range and bearing measurements and aims to change the measured heading angle of the robot and alter its trajectory. The attack is then tuned to make it undetectable by the implemented counter measures.

In this thesis, it is demonstrated that false data injection can be executed in such a way that it is almost impossible for the fault detection mechanism to perceive an attack. The simulations have shown that the key to stealth is to inject the false data in a slow and steady manner to deceive the EKF by steering its output. The deviation introduced is mostly unnoticeable as long as it is in the same order of magnitude of the noise, which is part of the robot's navigation system. In contrast, sudden and severe changes of measurement data are detected easily by the EKF and the fault detection mechanism.

Table of contents

1 Introduction and Motivation

As more and more industrial sectors experience an increase in use of robotic vehicles (RVs) in manufacturing, logistics etc., the reliance on these systems rises to the same degree. In the military, unmanned ground and unmanned aerial vehicles have reached a high significance and are indispensable, both in close quarter and long-range reconnaissance. Many of these systems are autonomously operated and usually not well protected, which leads to a rising threat of cyber-attacks on the control and navigation systems of those RVs [1,2]. Since RVs rely heavily on sensors and actuators for perception and navigation, they are especially prone to attacks aiming to disrupt these systems.

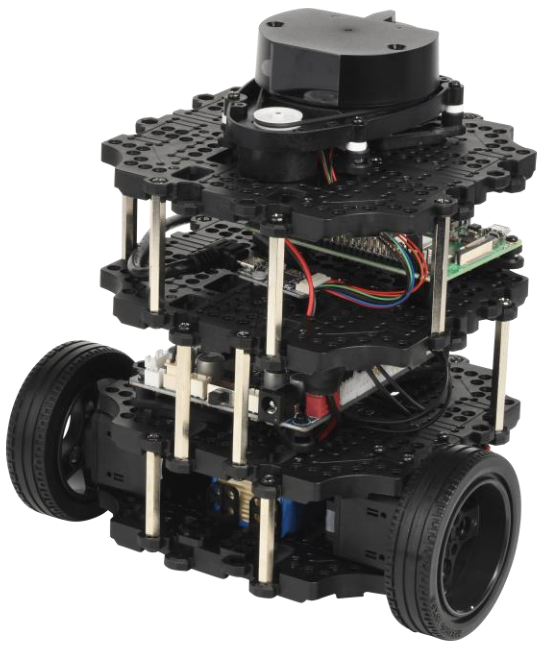

Figure 1: Differentially steered robot (TurtleBot 3 ‘Burger’) [3]

One way to perform an attack on a RV control system is through false data injection [4]. In this work, a differentially steered two-wheeled RV (Figure 1) is simulated in MATLAB. The measurement device, namely a LiDAR, is then influenced via false data injection. An approach from two sides is taken: The attacking side, which is aiming to disrupt the RV and the defending side, which tries to ensure the functionality of the RV and detect ongoing attacks. On the one hand, an attempt is made to fabricate attacks on the simulated RV that have a severe impact on the behavior of the robot (such as changing its course) but are ideally undetectable. On the other hand, an Extended Kalman Filter (EKF) is integrated into the simulated system, which improves navigation capabilities and feeds a fault detection mechanism with the innovation of predicted and measured system states to trigger an alarm in the event of an attack. The difficulty in designing both the EKF and the fault detection mechanism is to be able to detect even small attacks executed over a period of time. Such attacks can lead to a severe failure without raising any alarm [4].

Scope

In the context of this work, the question shall be answered whether it is possible to construct an attack that overcomes the fault detection mechanism, remains unnoticed and at the same time strongly influences the behavior of the robot. A simulation has been made with the objective of implementing it into a physical system in the future.

Outline

The second chapter will address the key aspects of the background material. In the third chapter, the MATLAB script used for simulation is explained. The fourth chapter presents the results, that have been generated using MATLAB script. These results are interpreted in the fifth chapter followed by a Conclusion of the findings of this thesis and an Outlook in the sixth chapter.

2 Theoretical Background

The following chapter provides the background material necessary for the understanding of this thesis.

2.1 Two wheeled robotic vehicles and dynamics model

The simulated platform used for the various experiments is a two wheeled differentially steered robot relating to the *TurtleBot 3 'Burger'*. To properly represent and simulate the behavior of such a robotic system, a transition model has been derived to construct predictions for the expected next system state based on the previous system state and the control input. It transitions the previous system state into the next. To achieve a more detailed simulation of a two wheeled differentially steered robot, the dynamics model of Dusek et al. [5] has been used and appropriately modified. This model allows us to simulate the system more accurately, considering the dynamics describing wheel speed dependency on the motor supply voltages, construction, geometry and other parameters of the chassis and motors.

The input for the dynamics model can be described as u_L and u_P, which are signals that control the supply voltage of each motor and are fed into the linear part of the detailed system model. The linear model, that describes the dynamics of the robot (motor current and wheel rates), can be expressed as a standard state-space model in matrix form [5]:

$$\frac{dx}{dt} = \boldsymbol{A}x + \boldsymbol{B}u \tag{2.1.1}$$

The output equation gives the velocity v and the heading angle rate ω.

$$y = \boldsymbol{C}x \tag{2.1.2}$$

The state control and output vectors are:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{i_l} \\ \boldsymbol{i_r} \\ \boldsymbol{\omega_l} \\ \boldsymbol{\omega_r} \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{U_l} \\ \boldsymbol{U_r} \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} \tag{2.1.3}$$

Some substitutions are introduced to compress the space-matrices as seen in 2.1.4 - 2.1.7:

$$\begin{aligned} a_l &= k_r + \frac{k_v l_r r_G^2}{l_l + l_r} & a_l &= k_r + \frac{k_v l_r r_G^2}{l_l + l_r} \\ b_l &= J + \frac{m l_r r_G^2}{l_l + l_r} & b_r &= J + \frac{m l_l r_G^2}{l_l + l_r} \\ c_l &= k_r l_l + \frac{k_\omega r_G^2}{l_l + l_r} & c_r &= k_r l_r + \frac{k_\omega r_G^2}{l_l + l_r} \\ d_l &= J l_l + \frac{J_B r_G^2}{l_l + l_r} & d_r &= J l_r + \frac{J_B r_G^2}{l_l + l_r} \end{aligned} \qquad \textbf{(2.1.4)}$$

$$\boldsymbol{A} = \begin{bmatrix} -\frac{R + R_z}{L} & -\frac{R_z}{L} & -\frac{K}{L} & 0 \\ -\frac{R_z}{L} & -\frac{R + R_z}{L} & 0 & -\frac{K}{L} \\ \frac{K(d_r + b_r l_l)}{b_l d_r + b_r d_l} & \frac{K(d_r - b_r l_l)}{b_l d_r + b_r d_l} & -\frac{d_r a_l + b_r c_l}{b_l d_r + b_r d_l} & -\frac{d_r a_r + b_r c_r}{b_l d_r + b_r d_l} \\ \frac{K(d_l - b_l l_l)}{b_l d_r + b_r d_l} & \frac{K(d_l + b_l l_r)}{b_l d_r + b_r d_l} & -\frac{d_l a_l + b_l c_l}{b_l d_r + b_r d_l} & -\frac{d_l a_r + b_l c_r}{b_l d_r + b_r d_l} \end{bmatrix} \qquad \textbf{(2.1.5)}$$

$$\boldsymbol{B} = \begin{bmatrix} \frac{U_0}{L} & 0 \\ 0 & \frac{U_0}{L} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad \textbf{(2.1.6)}$$

$$\boldsymbol{C} = \begin{bmatrix} 0 & 0 & \frac{l_r r_G}{l_l + l_r} & \frac{l_l r_G}{l_l + l_r} \\ 0 & 0 & -\frac{r_G}{l_l + l_r} & \frac{r_G}{l_l + l_r} \end{bmatrix} \qquad \textbf{(2.1.7)}$$

Notation	Meaning
l_l	distance of the left wheel from center of axis [m]
l_r	distance of the right wheel from center of axis [m]
i_l	current of the left motor [A]

i_r	current of the right motor [A]
ω_l	angular velocity of the left rotor [1/s]
ω_r	angular velocity of the right rotor [1/s]
k_r	coefficient of the resistance against rotating of rotor and gearbox [kg/s]
k_v	coefficient of the resistance against linear motion [kg/s]
k_ω	coefficient of the resistance against robot rotation [kg*m^2/s]
m	total weight of the robot [kg]
J	total moment of inertia of rotor and gearbox [kg*m^2]
J_B	total moment of inertia of robot [kg*m^2]
r_G	reduced radius of the wheel [m]
R	motor winding resistivity [Ω]
R_Z	source resistance [Ω]
K	electromotoric constant [kg*m^2/s^2*A]
L	motor inductance [H]
U_0	source voltage [V]

Table 1: DC Motor and chassis parameters

The output variables of the dynamics model (actual robot response) are fed into the transition model for the navigation filter and are from now on considered the new control input vector u_k, consisting of the instantaneous angular and translational velocity derived from odometry.

For the navigation filter, the input needed to derive the next filter state $(k+1)$ consists of two vectors, the vector of the current state $(\mathrm{X_k})$, representing the location and orientation of the robot in a global coordinate frame and the input vector $(\mathrm{u_k})$, representing the steering commands of the robot.

$$X_{k+1} = f(X_k, u_k) \tag{2.1.8}$$

$$X_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} \tag{2.1.9}$$

$$u_k = \begin{bmatrix} v_k \\ \omega_k \end{bmatrix} \qquad \textbf{(2.1.10)}$$

The state vector consists of the position and orientation of the Robot at a given point in time (k), which can be described with two coordinates on a 2D plane (x, y) and an angle of orientation (θ) relative to the y-axis of the global coordinate frame. The input vector is composed of the translational velocity (v) and the angular velocity (ω), which are derived from their respective equations of motion. The revolutions of the right and left wheel (ω_r, ω_l) are taken from the odometry sensors of the respective wheel. [6]

$$v_k = \frac{\omega_{r_k} \cdot R + \omega_{l_k} \cdot R}{2} \qquad \textbf{(2.1.11)}$$

$$\omega_k = \frac{\omega_{l_k} \cdot R - \omega_{r_k} \cdot R}{b} \qquad \textbf{(2.1.12)}$$

The transition model from time k to $k+1$ for the system can be expressed as

$$\mathrm{X_{k+1}} = \mathrm{f}(\mathrm{X_k}, \mathrm{u_k}) = \begin{bmatrix} \mathrm{x_{k+1}} \\ \mathrm{y_{k+1}} \\ \theta_{\mathrm{k+1}} \end{bmatrix} = \begin{bmatrix} \mathrm{x_k} \\ \mathrm{y_k} \\ \theta_{\mathrm{k}} \end{bmatrix} + T \begin{bmatrix} (v_{\mathrm{k}} + dv)\,.\cos(\theta_{\mathrm{k}}) \\ (v_{\mathrm{k}} + dv)\,.\sin(\theta_{\mathrm{k}}) \\ \omega_{\mathrm{k}} + d\omega \end{bmatrix} \qquad \textbf{(2.1.13)}$$

in which T is the duration between each timestep k.

The transition model (2.1.13) only applies accurately in ideal conditions. The model doesn’t take an uneven surface, slipping of the wheels or the impact of friction or the general noise of the measurement into account. To accommodate for that fact, a noise model is applied to the transition model ($dv, d\omega$). The noise model has the characteristics of normal distribution with the mean of zero and is applied to the odometry measurements of each wheel and thus to the translational and angular velocity. [5–8]

2.2 Extended Kalman Filter

The Kalman Filter is an algorithm that provides estimates of various unknown variables by using a series of noisy measurements observed over time. The output estimates of the Kalman Filter usually are more accurate than those based on a single measurement alone. Mathematical models can be added as constraints to account for the dynamic relationship between each system state. As an example, equations of motion can be used to precisely estimate changing positions and speeds. It has to be stated that the Kalman Filter can only be applied to linear problems. For nonlinear systems, a nonlinear approach has to be taken in the form of an Unscented Kalman Filter (UKF) or Extended Kalman Filter (EKF). For highly nonlinear systems, usually a UKF approach is taken. But for an adequately accurate depiction of the behavior of a two wheeled robot and its well-defined dynamics model, an EKF approach is sufficient to reliably represent the behavior of the system. The EKF linearizes about an estimate of the current mean and covariance. To achieve this linearization the Jacobian (F, H) of the kinematics transition model (f), and the measurement model (h) are used. [6,9-13]

$$F_x(k) = \frac{\delta f(X,u)}{\delta(x,y,\theta)} = \begin{bmatrix} 1 & 0 & -T * \sin(\theta_k) \\ 0 & 1 & T * \cos(\theta_k) \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2.1}$$

$$z_{pred_k} = h(X_k, L_k) = \begin{bmatrix} \sqrt{\left(x_{l_k} - x_{pred_k}\right)^2 + \left(y_{l_k} - y_{pred_k}\right)^2} \\ \tan^{-1}\left(\frac{y_{l_k} - y_{pred_k}}{x_{l_k} - x_{pred_k}}\right) + \theta_k + \frac{\pi}{2} \end{bmatrix} \tag{2.2.2}$$

$$H(k) = \frac{\delta h(X)}{\delta X}$$
$$= \begin{bmatrix} \frac{x_{l_k} - x_{pred_k}}{\sqrt{\left(x_{l_k} - x_{pred_k}\right)^2 + \left(y_{l_k} - y_{pred_k}\right)^2}} & \frac{y_{l_k} - y_{pred_k}}{\sqrt{\left(x_{l_k} - x_{pred_k}\right)^2 + \left(y_{l_k} - y_{pred_k}\right)^2}} & 0 \\ \frac{y_{l_k} - y_{pred_k}}{\sqrt{\left(x_{l_k} - x_{pred_k}\right)^2 + \left(y_{l_k} - y_{pred_k}\right)^2}} & \frac{x_{l_k} - x_{pred_k}}{\sqrt{\left(x_{l_k} - x_{pred_k}\right)^2 + \left(y_{l_k} - y_{pred_k}\right)^2}} & 1 \end{bmatrix} \tag{2.2.3}$$

Here x_l, y_l are the locations of the different landmarks and x_{pred}, y_{pred} are the predicted coordinates of the RV. The index k represents the current timestep.

The algorithm in the form of a block diagram is illustrated in Figure 2. The input u_k is not altered before entering the Filter, meaning Block G equals 1 in this example. For the prediction phase, the Kalman filter produces estimates of the current state variables $X(x, y, \theta)$ using the transition model (Block F) already presented in section 2.1 along with their covariance P, the state variables of the previous timestep and the input $\mathrm{u_k}$. The covariance is updated during each iteration.

$$P_{k+1} = F_{x_k} \cdot P_k \cdot F_{x_k}^T + Q_{f_k} \tag{2.2.4}$$

For P, the covariance of the noise of the kinematics model (Q_f) is calculated with the Jacobian of the kinematics model (F) with respect to the input (u_k) and the input measurement covariance (Q_u), which consists of the input noise of the translational and angular velocity (q_v, q_ω).

$$Q_{f_k} = F_{u_k} \cdot Q_u \cdot F_{u_k}^T \tag{2.2.5}$$

$$F_u(k) = \frac{\delta f(X, u)}{\delta(v, \omega)} = \begin{bmatrix} T \cdot \cos(\theta_k) & 0 \\ T \cdot \sin(\theta_k) & 0 \\ 0 & T \end{bmatrix} \tag{2.2.6}$$

$$Q_u = \begin{bmatrix} q_v^2 & 0 \\ 0 & q_\omega^2 \end{bmatrix} \tag{2.2.7}$$

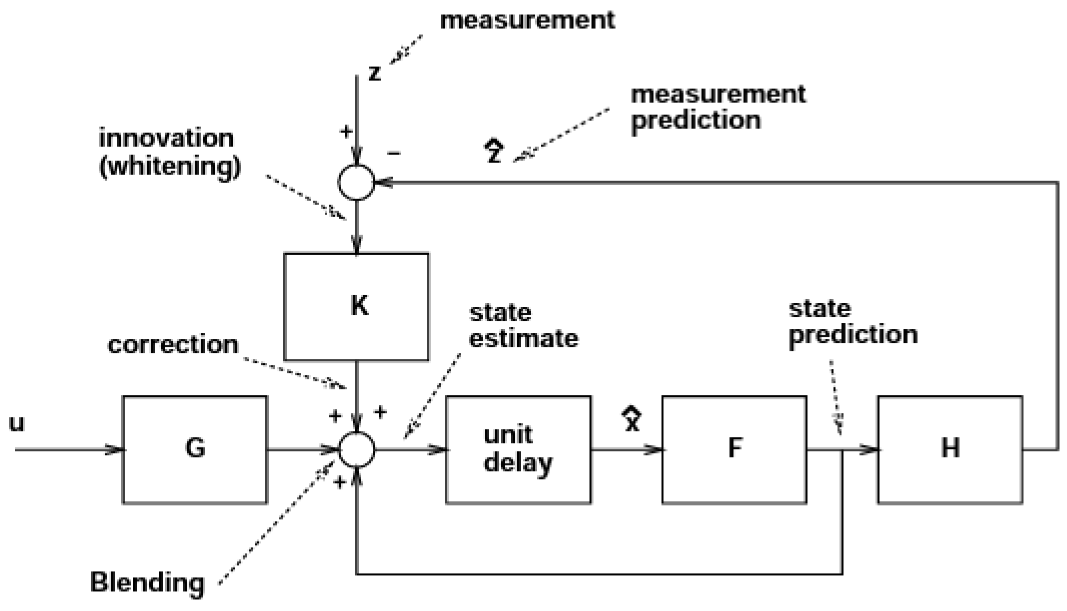

Figure 2: Illustration of the Kalman Filter algorithm [14]

Simultaneously, the installed LiDAR measures distance and bearing (r_k, ϕ_k) from the rover to a set of landmarks located around the RV. From the predicted state a set of predicted distance and bearing measurements is also calculated using the measurement model $(h(X_k, L_k))$ (Block H). These measurements and predicted measurements are then used to calculate the innovation (whitening), which is usually a good indicator for the functionality of the system. Ideally the innovation, the difference between measurement and prediction, should be small but is usually not zero due to uncertainties in the kinematics transition model and noisy measurements.

$$Innovation_k = \nu_k = z_{actual_k} - z_{pred_k} = z_{actual_k} - h(X_k, L_k) \quad \textbf{(2.2.8)}$$

Once the innovation is calculated, the predicted estimates of the system state are updated $(X_{pred,updated})$ based on the difference of the measured and the predicted state (ν). The innovation covariance (S) is also calculated. The innovation covariance is a measure for the confidence in a measurement. If S is high, the filter has a lower confidence in the reliability of those measurements and vice versa. The innovation covariance is the used for the derivation of the Kalman gain (K) (Block K), which is the weight given to the measurements versus the current prediction. The relevant equations are:

$$S_k = H_k \cdot P_k \cdot H_k^T + R_k \tag{2.2.9}$$

$$K_k = \frac{P_k \cdot H_k^T}{S_k} \tag{2.2.10}$$

$$X_{pred,updated_k} = X_{pred_k} + K_k \cdot \nu_k \tag{2.2.11}$$

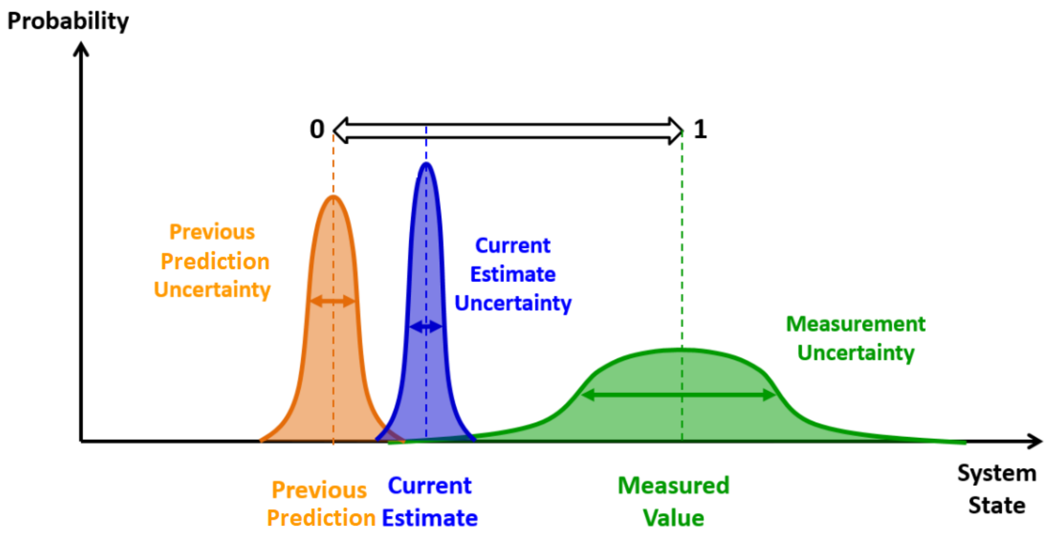

Figure 3: Visualization of the influence of the Kalman Gain (low Gain) [15]

The impact of the Kalman Gain for the estimation process is illustrated in Figure 3. For a large gain (high prediction uncertainty and low measurement uncertainty), the Kalman Filter prefers the measurements over the prediction and thus adapts to changing measurements and its noise more responsively resulting in a possibly jumpy trajectory. On the other hand, a relatively low Kalman gain close to zero would result in a smoother trajectory, since the filter conforms to the process model more closely decreasing the impact of measurement noise but also decreasing its responsiveness to changing measurements.

The updated state vector (correction) is then fed back into the algorithm and is then used as the new current state vector to calculate the next predicted system state. The algorithm is recursive. It can operate in real time, using only the present input measurements and the previously calculated system state and its uncertainty matrix, no additional past information is required. [6,8,9,14,16]

2.3 Measurement devices

To update the estimated system state of the RV, a measurement of the current system state needs to be made, which is then compared to the estimated position from the process model. The measurements are generated by a LiDAR, which is measuring the distance and bearing to a set of pre-defined landmarks around the RV.

LiDAR stands for Light Detection and Ranging. Rapid laser pulses are fired by a Transmitter and reflected by any surface surrounding the RV and then captured by a sensor (receiver). This process is illustrated in Figure 4.

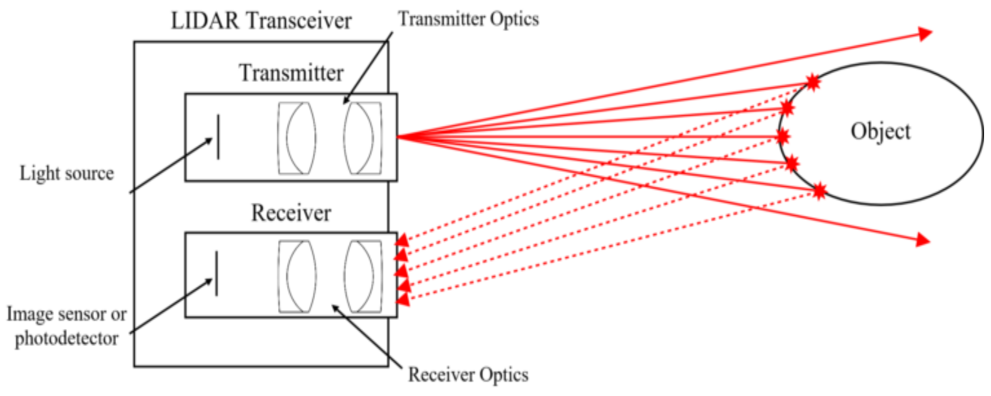

Figure 4: Schematic illustration of LiDAR measurement process [17]

The time difference between the emission and detection of each reflected laser pulse is then measured. Using the speed of light, the distances between the sensor and a given object can be calculated. The duration for the return of each pulse (t) is multiplied by the speed of light (c) and then divided by two, since the measured duration corresponds to the travel time to the landmark and back to the sensor. [18]

$$d = \frac{c \cdot t}{2} \tag{2.3.1}$$

The laser pulses that are fired at different horizontal and—in a 3D environment—vertical angles can generate point clouds, which can be used to digitally map the surroundings of the RV. Given the location of a set of landmarks around the RV, one is able to determine (to some level of accuracy) the location of the RV using the distance and angle measurements (multiangulateration) to each landmark. The TurtleBot possesses a 360° LiDAR with SLAM capabilities (Simultaneous Localization And Mapping) which creates a map using the measurement data generated by the LiDAR using a version of the EKF. [3]

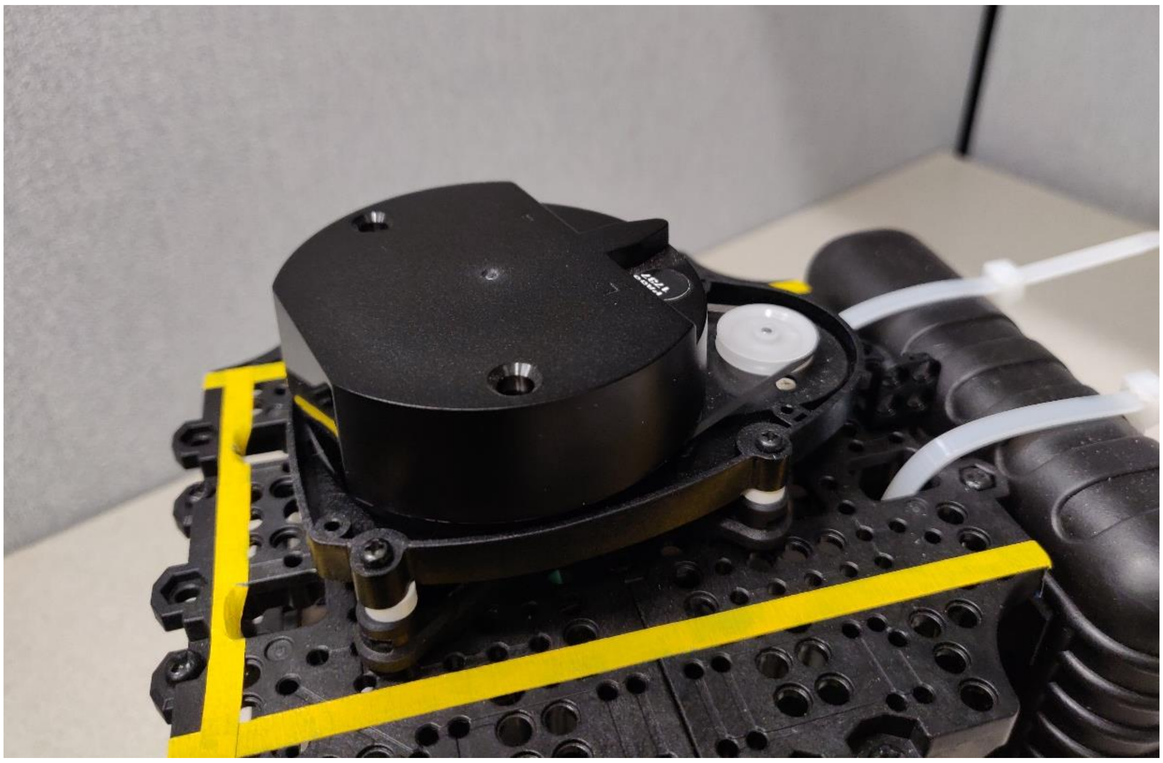

Figure 5: 360° Laser Distance Sensor LDS-01 (LiDAR)

The mapping capabilities are not used in the simulation. Instead a set of known landmarks is created and used to measure the RV position using only the LiDAR. The constellation of landmarks and the multiangulateration process is illustrated in Figure 6. The landmarks are portrayed as blue circles surrounding the 10x10m test area. In this situation the RV is located at the coordinates (1,1) and is marked with a red X. Each LiDAR pulse that gets reflected back from a landmark is portrayed as a red dotted line.

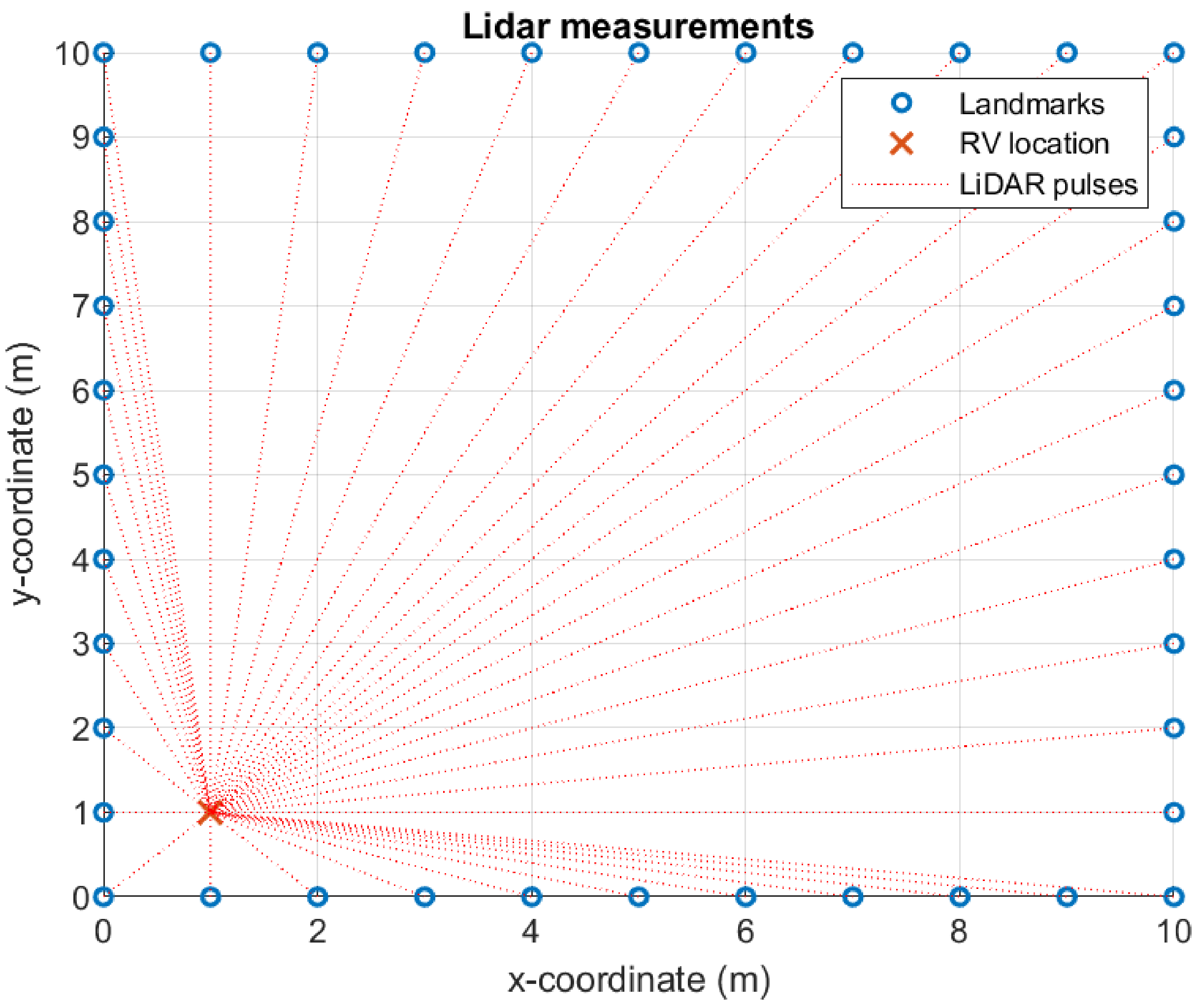

Figure 6: LiDAR measurements of known landmarks

2.4 False Data Injection

False data injection is a form of attack against a control system, for example a navigation system of a robotic vehicle, in which the attacker injects malicious data, i.e. measurements, that can mislead the state estimation process without triggering and alarm. The incorrectly estimated system state can then lead the system to take unpredicted action like deviating from a desired trajectory. Since the measurements used for navigation and estimation are subject to noise, there usually is some tolerance when it comes to identifying deviations because the naturally occurring noise would cause a fault detection mechanism to trigger false alarms. To stay undetectable, the deviation caused by the attack must be smaller than a certain threshold set by the system operator. An approach to derive the detection threshold of a RV has been made by Dash et al. [4]. They have been reverse-engineering the control state estimation model without having root access to the operating system. [4,19]

In this work, false data is injected in the form of altered LiDAR measurements. These LiDAR measurements consist of a distance (r) and angle (ϕ) component.

$$z_k = \begin{bmatrix} r_k \\ \phi_k \end{bmatrix} \tag{2.4.1}$$

Here r_k is the distance between the robot and each specific landmark and ϕ_k the bearing of each landmark relative to the robot and its own orientation. Through multiangulateration the system is able to derive a measured position from the LiDAR output. To alter this position and to influence every measurement in such a way that the compromised measurement still results in one exact position, the set of landmarks may be changed by using homogeneous transformation matrices [20] instead of applying a bias value to each of the measurements individually. This allows for stretching and compressing, sheering and rotation of the set of landmarks. In a two-dimensional environment, the homogeneous transformation matrices are in a 3x3 dimension. Therefore, the 2D coordinates of the landmarks must be supplemented by an entry with the value zero. After the transformation this third entry is deleted again.

$$Translation\colon\ Tl = \begin{bmatrix} 1 & 0 & k \\ 0 & 1 & l \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4.2}$$

$$Shear_x\colon\ S_x = \begin{bmatrix} 1 & m & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4.3}$$

$$Shear_y\colon\ S_y = \begin{bmatrix} 1 & 0 & 0 \\ n & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4.4}$$

$$Stretch\colon\ St = \begin{bmatrix} p & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4.5}$$

$$Rotate:\ R = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.4.6)}$$

These matrices (Equation 2.4.2 - 2.4.6) can be combined and the factors $(k, l, m, n, p, q, \alpha)$ can be tuned to arrange the set of landmarks in a desired way or change them over the course of a time period to provide a fake world map for the measurement instruments. The transformation matrix is then multiplied with the expanded position vector. [20]

$$Transformation:\ Tr = Tl \cdot S_x \cdot S_y \cdot St \cdot R \quad \textbf{(2.4.7)}$$

$$X_{transformed,l} = Tr \cdot \begin{bmatrix} x_l \\ y_l \\ 0 \end{bmatrix} \quad \textbf{(2.4.8)}$$

Figure 7 shows an example of transformed landmarks by using the homogeneous transformation. In this example, the original set of landmarks (blue) has been translationally moved 5 meters down and 5 meters to the left (orange). Afterwards it has been rotated 45 degrees around the origin (yellow) and stretched by a factor of 1.5 to either side (purple). Finally, the landmarks are sheared relative to the x-axis (green). Clearly, the transformed landmarks can be used to represent a different world, which may be used to spoof a navigation system.

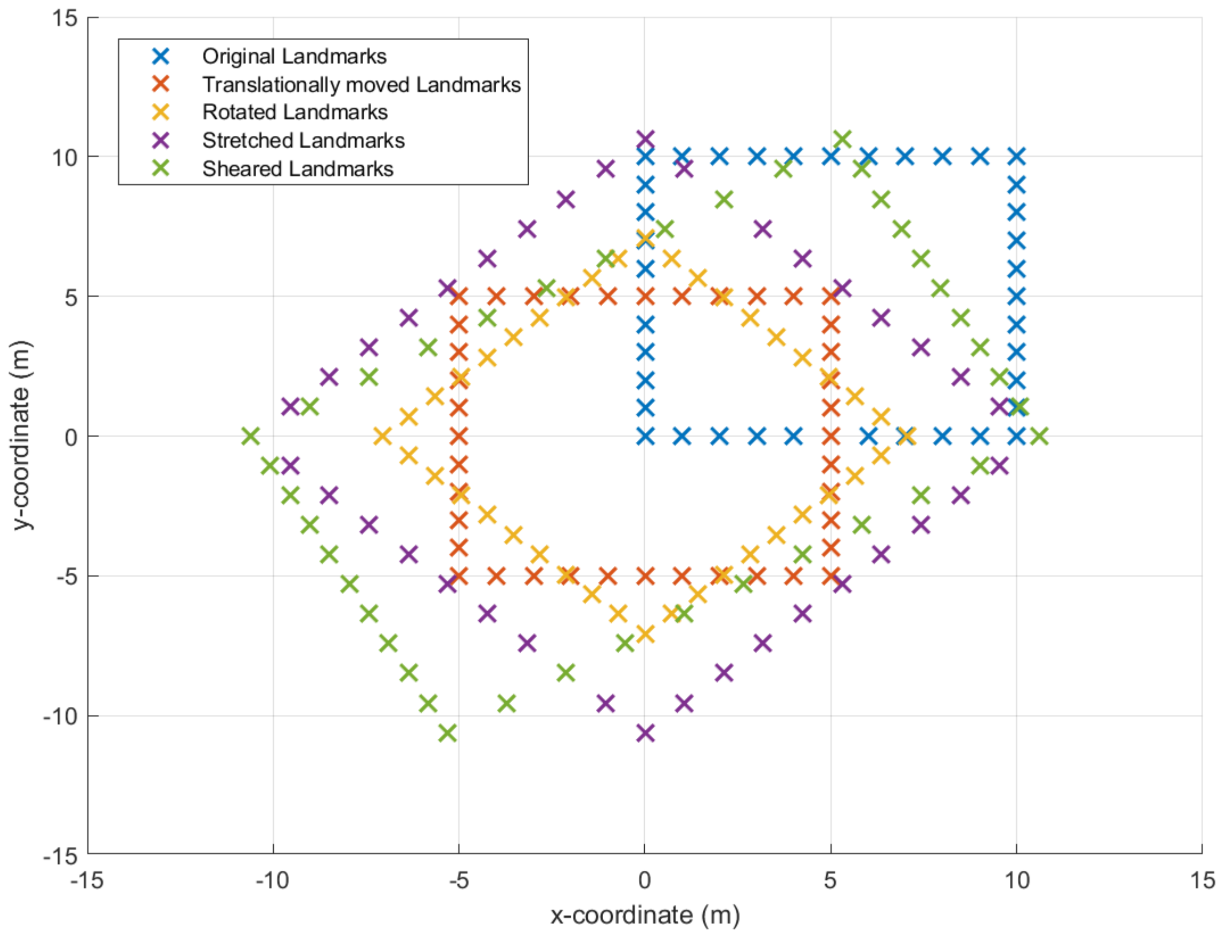

Figure 7: Examples of Landmark Transformation

2.5 Fault Detection Mechanism

It is relatively easy to uncover severe attacks and failures on a system like an autonomous vehicle by simply observing the deviation from the predicted system state [4,21]. However, inconspicuous attacks require more advanced techniques based on error estimation and decision theory to get detected. Usually, an operator or monitoring system has access to the innovation, hence the difference between the predicted system state through the process model and a measured system state. To recognize an attack on a robotic vehicle, a fault detection mechanism has to be implemented that evaluates the accessible system or process data and produces an alarm at a given threshold.

As a statistical hypothesis testing method for examining whether or not a random Gaussian vector has the assumed mean and covariance, the χ^2-test (chi-square-test) is widely applied for detecting soft failures in dynamical systems [14]. The χ^2 random variable ξ for each timestep is calculated using the following equation. Here ν is the innovation of each timestep and S the covariance of the innovation. The ~ means that ξ follows a χ^2 probability distribution.

$$\xi_k = \sum_{i=k-N+1}^{k} \nu_i^T \cdot S_i^{-1} \cdot \nu_i \sim \chi_N^2 \qquad \textbf{(2.5.1)}$$

After calculating ξ, it is compared to a certain chosen decision threshold. This threshold is defined through the degrees of freedom and the area of the distribution to the left of a critical value (rejection area). This becomes clearer looking at a χ^2 distribution. In this example, seen in Figure 8, the rejection area is marked red. If ξ is inside the red area, i.e. higher than the critical value (in this example at 10.2) the system has detected a failure, resulting in an error which is seen by the operator. Conversely, this means that the system is behaving correctly (by the definition of the operator) when ξ is smaller than the critical value. [22,23]

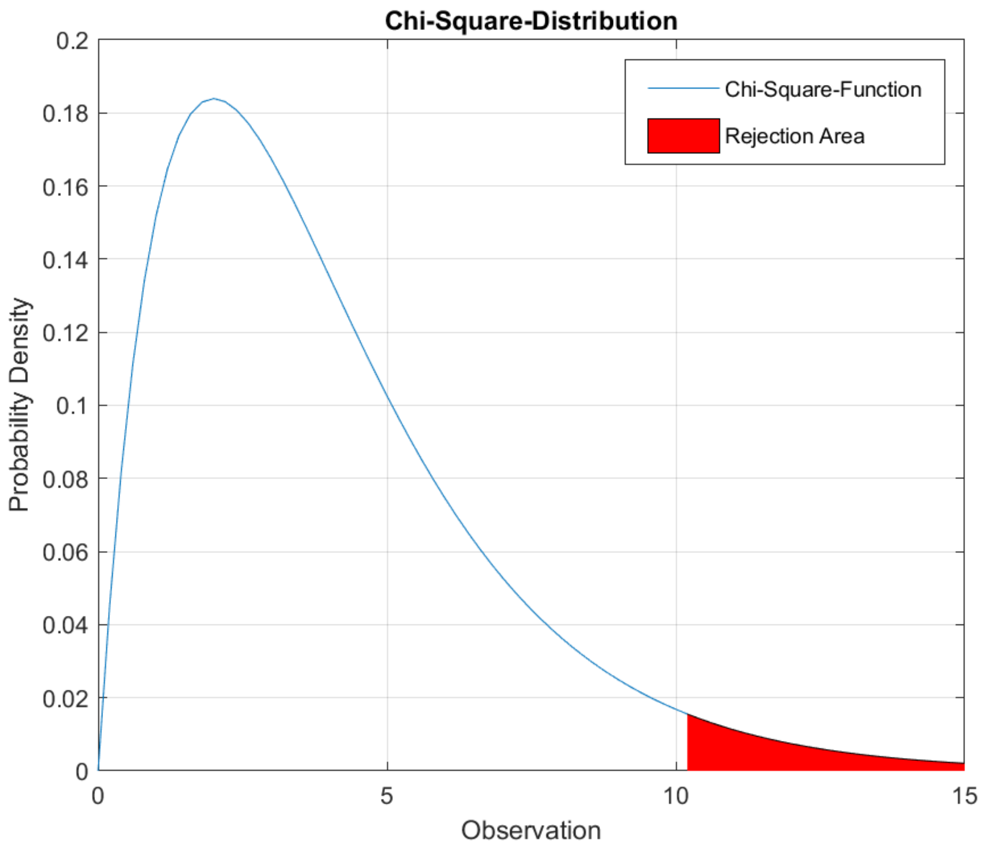

Figure 8: Chi-Square Distribution

3 MATLAB Simulation

The following chapter shall showcase two approaches taken to simulate the behavior of the robot in MATLAB and—for the second approach—the implementation of false data injection the Extended Kalman Filter and the detection mechanism.

3.1 Waypoint based trajectory

For the first attempt, the input that the operator is feeding into the system is a waypoint composed of a x- and y-coordinate. Using the desired waypoint and the initial system state of the robot, the desired heading is determined and thus the revolution speed of each wheel necessary to achieve this heading. The desired heading (θ) can be calculated using the atan2(x, y) function in MATLAB. The input variables are the differences of the x- and y-coordinates. The atan2(x, y) function returns values in the closed interval [-pi, pi] based on the values of x and y, as shown in Figure 9.

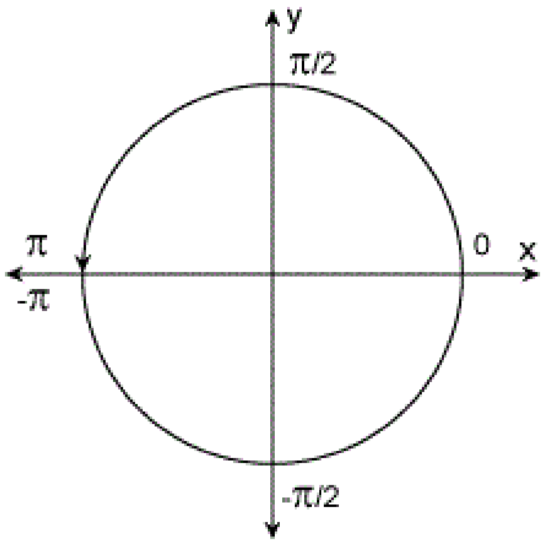

Figure 9: Return values of the atan2 function

The desired velocity of each timestep is calculated using:

$$v_{des} = \frac{\sqrt{(y_{des} - y_k)^2 + (x_{des} - x_k)^2}}{\sqrt{(y_{des} - y_{start})^2 + (x_{des} - x_{start})^2}} \cdot v_{max} + v_{min} \qquad \textbf{(3.1.1)}$$

The numerator consists of the distance between the robot and the waypoint during each timestep the denominator however consists of the distance between the starting point and the waypoint. This fraction is multiplied by a given maximum velocity (v_{max}). It becomes clear, that the desired velocity decreases with the decreasing distance between the robot and the waypoint. To account for that, a constant minimal velocity (v_{min}) is added so that the approach doesn't become too slow. This minimal velocity can be set however deemed necessary by the operator. After calculating the desired heading and velocity they are each fed into a PID controller and compared to the current heading and velocity to calculate the current error.

$$Error_\theta = \theta_{des} - \theta_k \tag{3.1.2}$$

$$Error_v = v_{des} - v_k \tag{3.1.3}$$

The PID Controller consists of a proportional (P), integral (I) and derivative (D) component, which are calculated using the error. The integral terms of the PID controller are added up. [24]

$$P_{k+1} = Error_{k+1} \tag{3.1.4}$$

$$I_{k+1} = (Error_{k+1} + Error_k) \cdot \frac{dt}{2} \tag{3.1.5}$$

$$D_{k+1} = \frac{(Error_{k+1} - Error_k)}{dt} \tag{3.1.6}$$

The three PID terms are then summed up with a respective gain (K).

$$PID_{k+1} = K_p \cdot P_k + K_i \cdot I_{k+1} + K_d \cdot D_k \tag{3.1.7}$$

The PID term is then fed into two integrators (Equation 3.1.8 – 3.1.11) that represent the robot dynamics. Summing up the PID Term (S_1) and calculating the integral of the latest two consecutive steps (S_2). The first integral (S_2) is then summed up (S_3) and integrated again over the latest two consecutive steps to calculate the output of the system (θ in this case). The approximation of input–output dynamics by a double integrator with transport delay is often used in the design of automatic control systems involving moving bodies and has been used for that reason. [25]

$$S_{1_{k+1}} = \sum_{k=1}^{k+1} PID_k \qquad \textbf{(3.1.8)}$$

$$S_{2_{k+1}} = \left(S_{1_{k+1}} + S_{1_k}\right) \cdot \frac{dt}{2} \qquad \textbf{(3.1.9)}$$

$$S_{3_{k+1}} = \sum_{k=1}^{k+1} S_{2_k} \qquad \textbf{(3.1.10)}$$

$$\theta_{k+1} = \left(S_{3_{k+1}} + S_{3_k}\right) \cdot \frac{dt}{2} \qquad \textbf{(3.1.11)}$$

Finally, the Feedback is calculated which is then returned to the calculation of the Error of the next state. The performance of the PID controller and thus the behavior of the robot can be influenced by tuning the spring and damping coefficient (B and K). The general operation of the controller can be seen in Figure 10.

$$Feedback = S_{3_{k+1}} \cdot B + \theta_{k+1} \cdot K \qquad \textbf{(3.1.12)}$$

$$Error_{\theta,k+1} = \theta_{des} - Feedback_k \qquad \textbf{(3.1.13)}$$

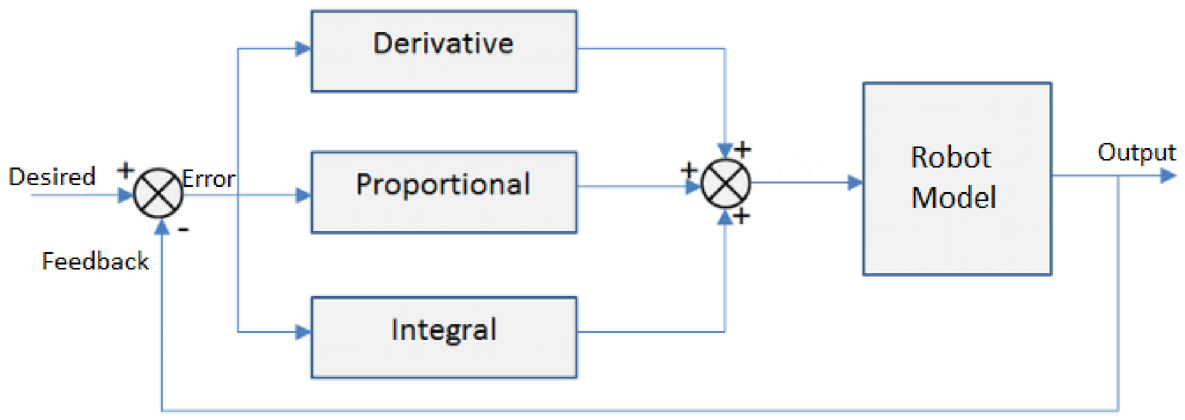

Figure 10: Block diagram closed loop PID Controller

After deriving the desired velocity and heading, the inverse kinematics model of a differentially steered robot is used to calculate the necessary wheel speeds.

$$d\theta_k = \theta_{k+1} - \theta_k \tag{3.1.14}$$

$$\omega_k = \frac{d\theta_k}{dt} \tag{3.1.15}$$

$$\omega_{r_k} = \frac{2 \cdot v_k + \omega_k \cdot b}{2 \cdot U} \tag{3.1.16}$$

$$\omega_{l_k} = \frac{2 \cdot v_k - \omega_{r_k} \cdot U}{U} \tag{3.1.17}$$

To derive the new position of the robot, which is then fed back into the calculation of the desired velocity and heading, the transition model already seen in section 2.1 is used. However, it has to be stated that the implementation of noise was omitted. The transition model is used to derive the current position of the robot. The resulting trajectory is illustrated in Figure 11. The trajectory shows a typical course for a PID controller with a damped overshoot of the desired trajectory. The trajectory can be adjusted by changing the damping and spring coefficients, as well as the gains of each PID element of the velocity and heading PID loop. In Figure 11 only the damping coefficient of the heading PID loop has been decreased, resulting in increased oscillation. All other coefficients were set to 1. The proportional gain was set to 0.9, the integral and derivative gain to 0.3.

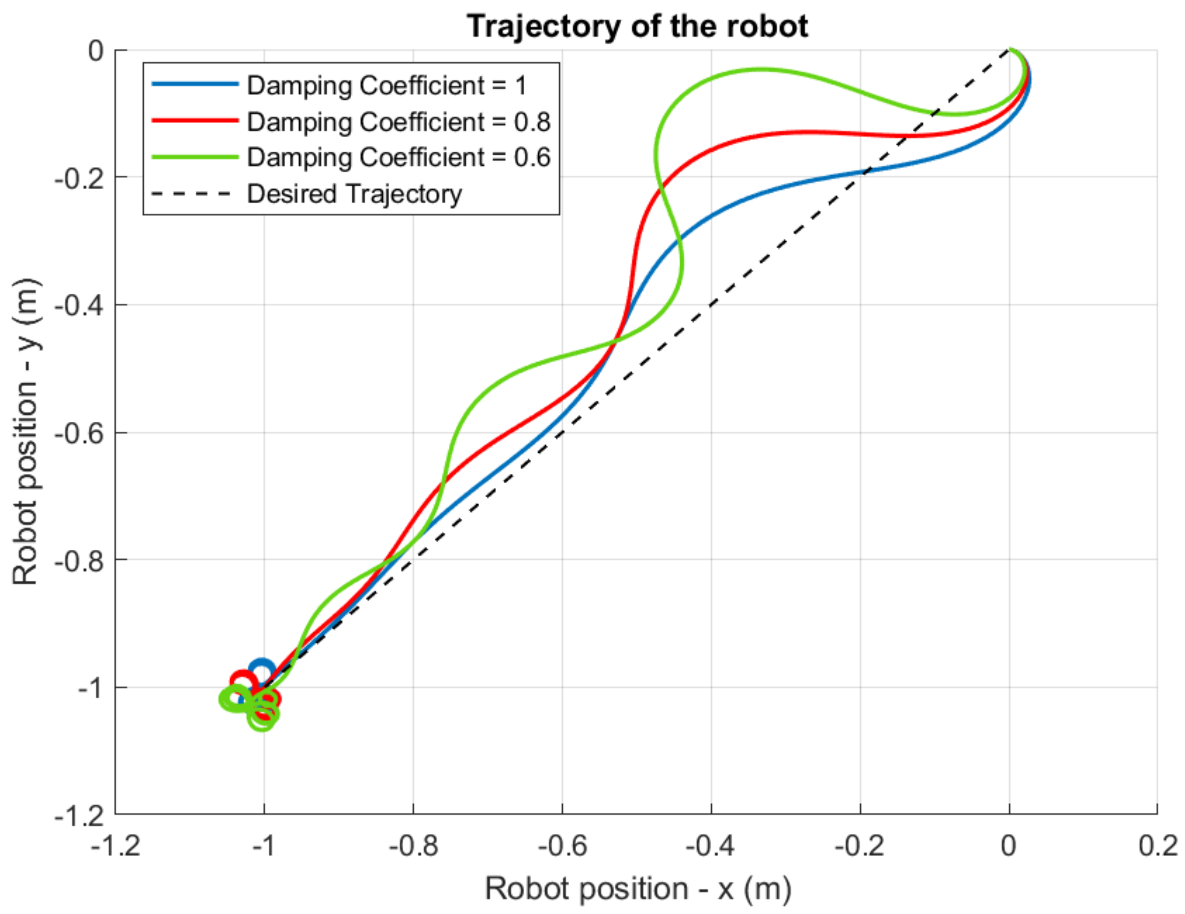

Figure 11: Waypoint based trajectory of the robot with different damping coefficients

Reaching the final destination, the velocity of the robot decreases gradually to a minimum. Once the robot passes the waypoint, the desired heading and thus the Error in the heading PID loop changes abruptly causing the robot to turn. The robot keeps doing loops through the waypoint on minimal velocity until the simulation ends. Alternatively, the controller could be shut off.

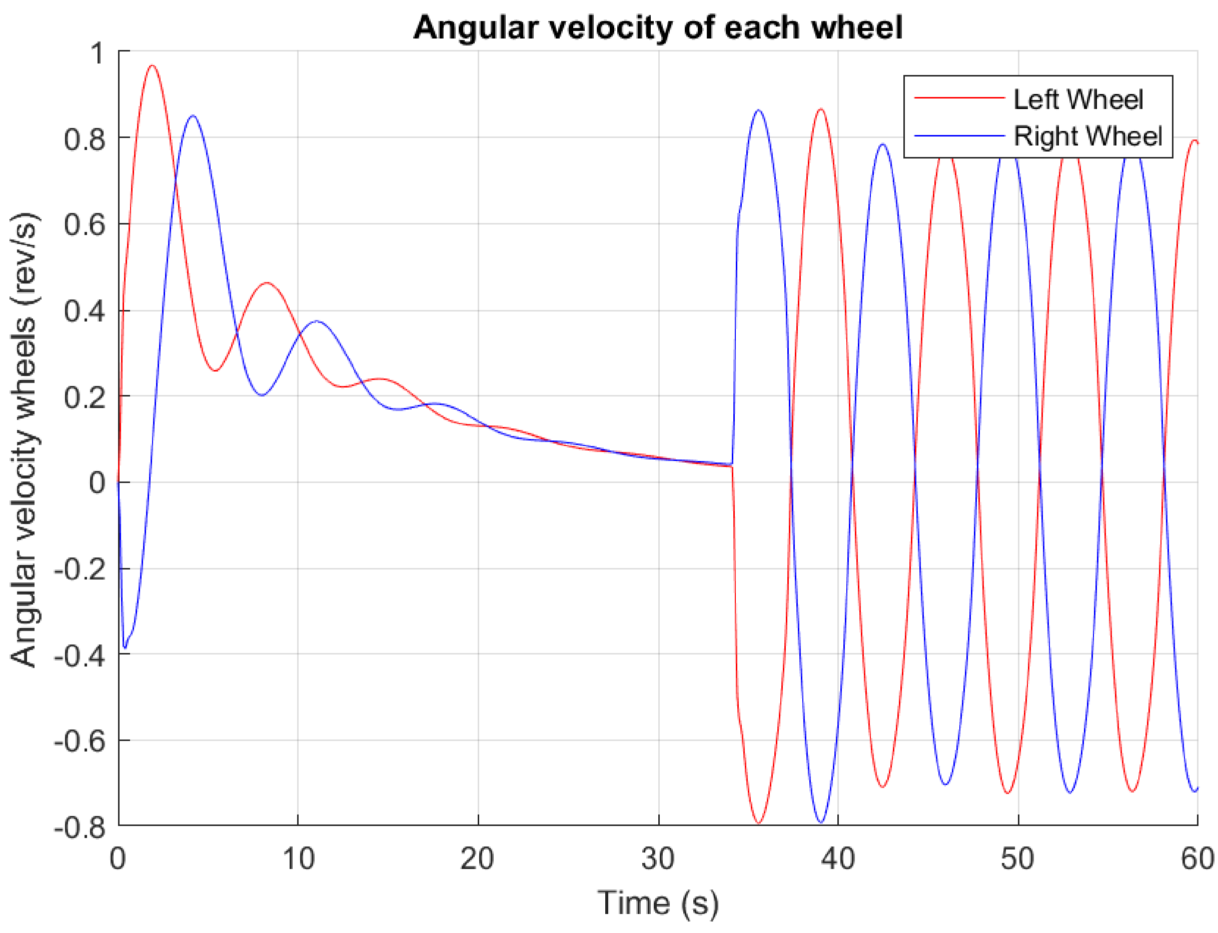

Figure 12: Angular velocity of each wheel

Figure 12 shows the wheel revolutions in revolutions per second throughout the 60-second-long simulation. The coefficients and gains are set as in Figure 11 with a damping coefficient auf 0.8. Due to the distance between robot and waypoint, the desired translational velocity increases causing the wheels to spin both at a positive and relatively high rate. Due to the nature of the PID controller, the trajectory shows an oscillation of the wheel speeds at a still positive rate. While approaching the waypoint, the desired translational velocity gradually decreases. After about 34.5 seconds, the robot reaches the waypoint causing the desired heading angle and thus the desired wheel speeds to change. At this stage the wheels rotate in opposite directions to achieve a swifter turn, which is indicated by the negative and positive deflections, respectively. However, the mean of this phase is not zero, since a minimal translational velocity is still maintained. Practically the controller could be turned off once the waypoint is reached at $t = 34.5$ sec.

The waypoint-based approach has not been pursued further to the implementation of a Kalman Filter, false data injection and a fault detection mechanism since a velocity and heading based input trajectory is more realistic for a real-world application of a ground based robotic vehicle. For future work and for the sake of completeness, exploring cyber-attacks on this system might still be of interest. Especially for a simulation of a UAV, a waypoint-based approach is more appropriate.

3.2 Velocity and heading based trajectory

The second simulation is based on the input of an operator, who is dictating the desired velocity and heading of the robot at any point in time of the simulation. The operator is able to command a certain speed and heading for a specific time interval. Furthermore, it is possible to gradually change the desired velocity and heading to achieve a smoothly curved trajectory. In Figure 13, two potential trajectories are illustrated. The starting position is set to 1 for the x and y coordinates. Marked as blue crosses are the landmarks surrounding the environment. These symbolize the landmarks used for the position determination with the LiDAR measurements. The landmarks are located around a 10x10m area and are standing 1m apart. In these experiments the velocity and heading are constant over the course of the simulation.

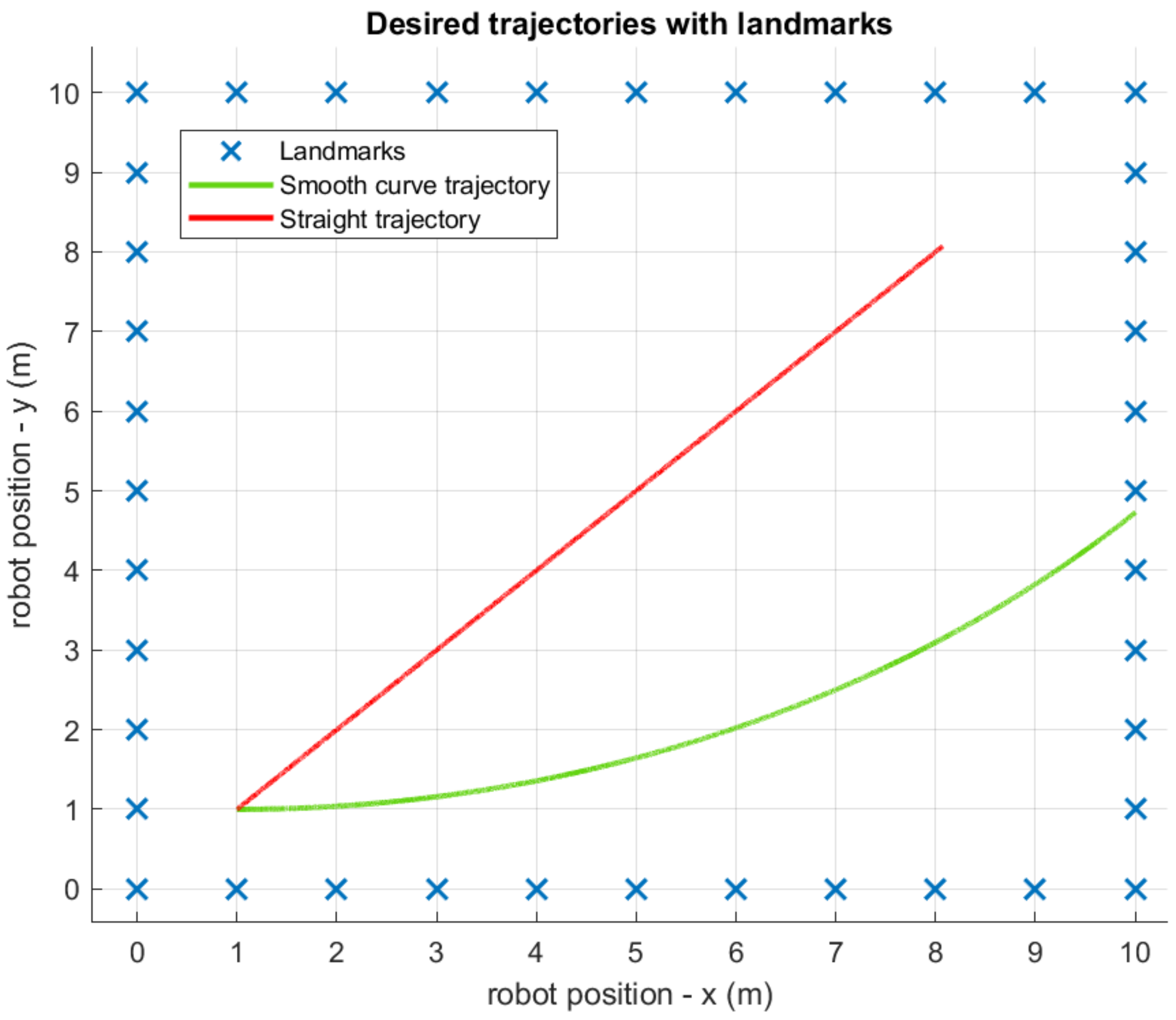

Figure 13: Desired trajectories and landmarks

The duration of the simulation is 200 seconds and is updated every 0.01 seconds. However, the measurement update in the EKF only occurs every second. The general operation of the Simulation is illustrated in Figure 14. The Code consists of three main parts marked in blue.

First is the Dynamics model, which is simulating the behavior of the two wheeled differentially steered robot. The dynamic model is fed with the control input of desired heading and velocity from the operator.

The second part is the Extended Kalman Filter in combination with the LiDAR measurement simulation, which improves and fuses the two position determination methods, hence the prediction of the system state through the dynamics and transition model and the actual measurement. Technically, a version of the dynamics is integrated into the EKF since the prediction of the next system state is done by the transition model, which is derived from the dynamics model. The LiDAR measurements, however, are corrupted in the event of false data injection. In this case the Extended Kalman Filter is only fed with an altered measurement vector. Not only does the EKF output the updated estimated system state, which is then used again by the dynamics model to predict the next system state, another output is the innovation.

The innovation, which is used in the third part, the fault detection mechanism. Through a statistical evaluation the fault detection mechanism decides whether the system is behaving correctly or not.

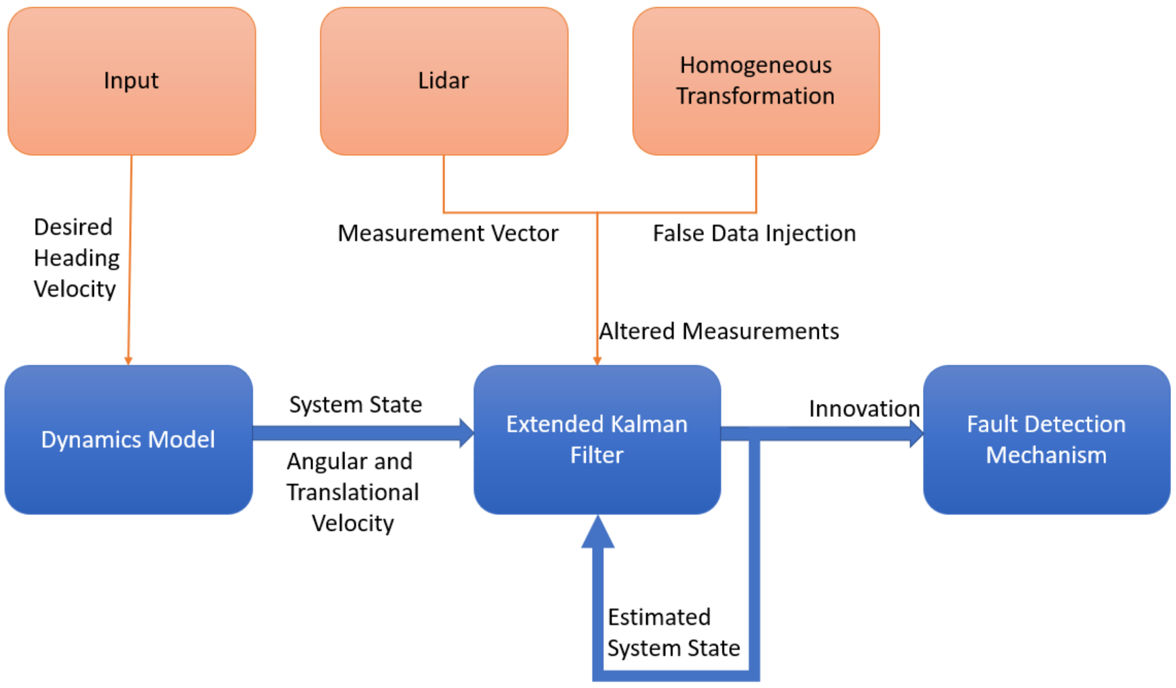

Figure 14: Illustration of the workflow of the Simulation

A numerical solver in form of a Euler Integrator is also implemented. The Euler method is a stepwise integration of the differential equations to derive the system states. Furthermore, it integrates the derivatives used in the update step of the EKF.

After the desired heading and velocity is fed into the dynamics model, the necessary DC-motor control voltages are calculated. Therefore, the forward kinematics are used to compute the actual velocity of the robot with Equation 2.1.2

Using the inverse kinematics model, the necessary wheel speeds for the next timestep are computed.

$$\begin{bmatrix}\omega_{ld}\\ \omega_{rd}\end{bmatrix} = \begin{bmatrix} l_r & l_l \\ -1 & 1\end{bmatrix}^{-1} \cdot \begin{bmatrix}\frac{v \cdot p_g \cdot (l_r + l_l)}{r} \\ \frac{\omega \cdot p_g \cdot (l_r + l_l)}{r}\end{bmatrix} \tag{3.2.1}$$

Here, p_g is the gearbox ratio.

From there on, the DC-motor control voltages are calculated with the associated gains K_i and K_p the differences of the revolutions of each wheel from the previous timestep to the desired next timestep and the current of each motor.

$$U_l = K_p \cdot (\omega_{ld} - \omega_l) + K_i \cdot I_l \tag{3.2.2}$$

$$U_r = K_p \cdot (\omega_{rd} - \omega_r) + K_i \cdot I_r \tag{3.2.3}$$

Finally, the robot dynamics are calculated for the next step using Equation 2.1.1.

The output that is then fed into the EKF is the current angular and translational velocity. Furthermore, the estimated previous position and heading or—in the first iteration—the initial conditions of position and heading are used to calculate the next predicted system state using the transition model. Since most of the equations and general steps have already been illustrated and explained only a rough re-explanation and details not yet mentioned are provided. [5]

The EKF estimate is updated every 0.01 seconds (T) and the measurement update occurs every second (T_U). Based on these time periods, the control measurement covariance (Q_u) with $q_v \left[\frac{m}{s}\right]$ and $q_\omega[\frac{rad}{s}]$ and the factors for the measurement noise ($q_r\ [m], q_\phi[rad]$) are derived.

$$Q_u = \begin{bmatrix} q_v^2 & 0 \\ 0 & q_\omega^2 \end{bmatrix} \tag{3.2.4}$$

$$q_v = \frac{1}{\sqrt{T}} \cdot \sqrt{0.001} \left[\frac{m}{s}\right] \tag{3.2.5}$$

$$q_\omega = \frac{1}{\sqrt{T}} \cdot \sqrt{0.012 \cdot \frac{\pi}{180}}\ [\frac{rad}{s}] \tag{3.2.6}$$

$$q_r = \frac{1}{\sqrt{T_U}} \cdot \sqrt{0.01}\ [m] \tag{3.2.7}$$

$$q_\phi = \frac{1}{\sqrt{T_U}} \cdot \sqrt{0.25 \cdot \frac{\pi}{180}}\ [rad] \tag{3.2.8}$$

From there on the process noise is derived, which is used in the transition model. The noise is simulated by multiplying the respective factor q with a random scalar x_{random} drawn from the standard normal distribution with zero mean.

$$dv = \sqrt{q_v} \cdot x_{random} \tag{3.2.9}$$

$$d\omega = \sqrt{q_\omega} \cdot x_{random} \tag{3.2.10}$$

The transition model then makes a prediction about the current system state. In addition, the covariance (P) of the system state is updated, which has already been discussed in section 2.2. After updating the covariance two checks are performed to see if there are negative elements on the main diagonal of P or if P diverges. If one of these events occurs, the simulation is stopped, and an error is displayed.

For the setup of the measurement update, a set of landmarks is created, as discussed earlier. Depending on the scenario, these landmarks positions are then left untouched or—in the event of false data injection—altered, using homogeneous transformation. The actual robot position is then fed into a distance-to-point function to determine the measurement vector to emulate the LiDAR sensor. This function calculates the distance (r) and bearing (ϕ) to each landmark from the position and orientation of the robot resulting in a vector of 82 entries for 41 landmarks. Afterwards noise is added to the measurements using the following equations:

$$r_{noisy} = q_r \cdot x_{random} \tag{3.2.11}$$

$$\phi_{noisy} = q_{\phi} \cdot x_{random} \qquad \textbf{(3.2.12)}$$

To calculate the innovation, a second measurement vector has to be built. The second vector is based on the position that has been predicted by the measurement model, which is already subject to inaccuracies. From there on, the process of building the measurement vector is the same. These two measurement vectors are then compared to derive the innovation. [6]

$$Innovation_k = \nu_k = z_{actual_k} - z_{pred_k} \qquad \textbf{(3.2.13)}$$

In a real-life application, the predicted measurement vector has to be derived through the use of a measurement model as it has been stated in section 2.2. The LiDAR can only measure the distances and bearings from the actual position of the robot (actual measurement). For the derivation of the innovation the predicted measurement has to be calculated based on the predicted position of the robot and cannot be measured directly because the predicted and actual position of the robot differ.

Since the innovation angle and distance have different magnitudes they are normalized using

$$\nu_{k,norm} = chol(S^{-1}) \cdot \nu_k \qquad \textbf{(3.2.14)}$$

where S is the covariance of the innovation, which has already been mentioned in section 2.2 This normalized innovation is then used to calculate ξ for the χ^2 test. However, ξ is not compared to a pre-defined critical value derived from a χ^2 table or the respective function. Rather a threshold is derived from running the nominal simulation numerous times and observing the outcome of the χ^2 test.

4 Simulation

The following chapter presents three different scenarios of the operation of the robot. A nominal scenario, in which the robot operates undisturbed is used to establish baseline performance. Two spoofed scenarios with altered LiDAR measurements are used to illustrate the ability to influence the system trajectory without detection.

4.1 Nominal case

In the first scenario the robot is operating normally and has access to unaltered LiDAR measurements, which are still subject to noise.

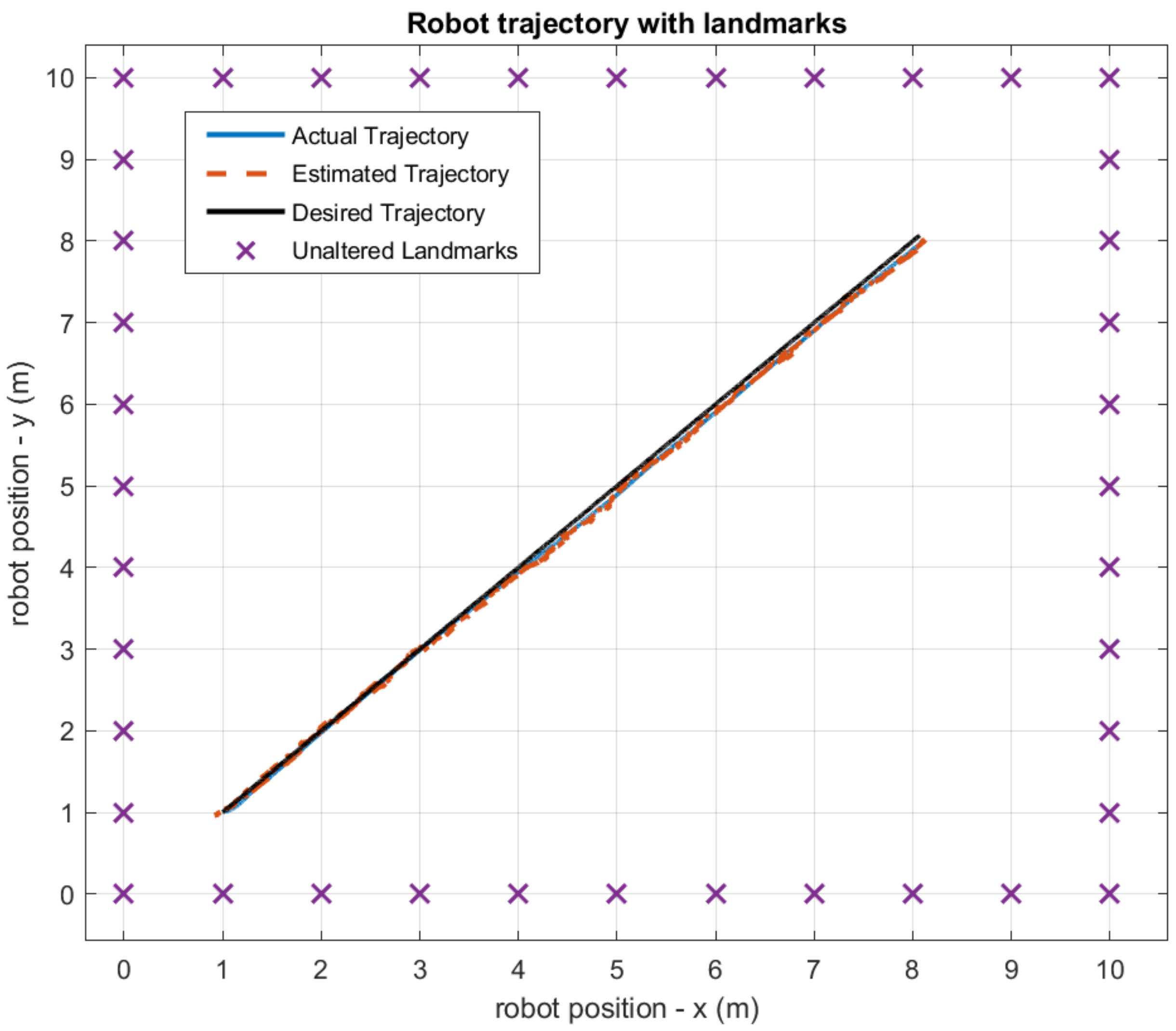

Figure 15: Nominal robot trajectory with landmarks for localization

The trajectory of the nominal robot is illustrated in Figure 15. The actual trajectory and the trajectory that has been estimated by the EKF are almost identical to the desired trajectory, although there are some fluctuations in the estimation and the actual trajectory. This indicates the correct function of the simulation program. The robot moves across the workspace at a 45-degree angle starting at the position (1,1) and ending at the position (8,8). Surrounding the robot is the set of 41 landmarks that represent the boundaries of the workspace.

Figure 16 shows the normalized innovation for this 200-second-long scenario. Each color corresponds to the innovation regarding one landmark and comprises an angle and a distance innovation. Although the duration of a timestep is 0.01 seconds, the innovation is only updated each second, since a measurement update—which is needed for the calculation of the innovation—only occurs every second resulting in a total of 200 innovation updates. Due to the normalization, the actual values of the innovation are not portrayed. Meaning the actual values of distance and angle innovation are smaller than the normalized innovation values. The scatterplot can be described as normally distributed with zero mean and $\sigma \approx 1.5$, which is to be expected for the normalized data.

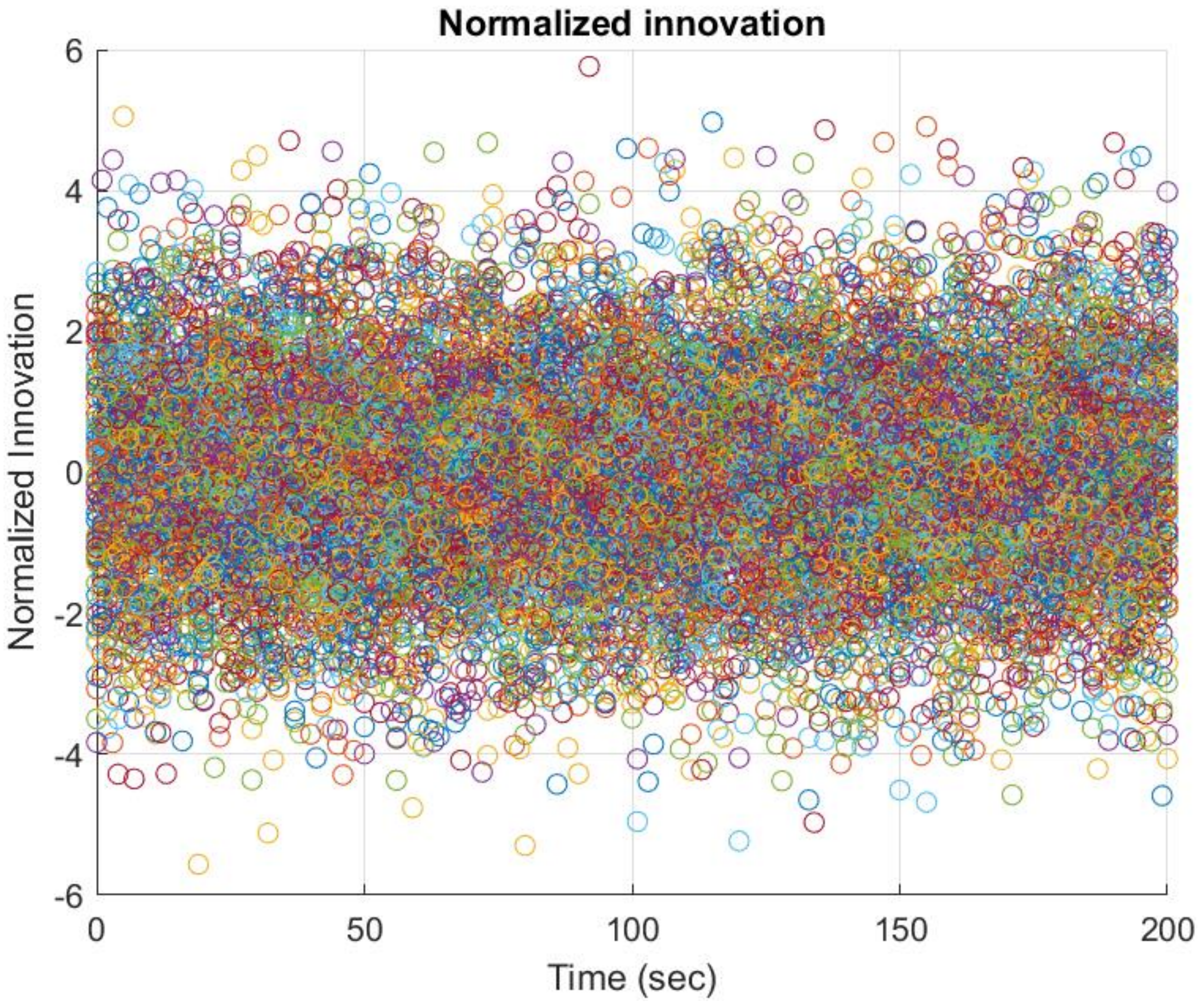

Figure 16: Normalized innovation for nominal simulation

Illustrated in Figure 17 are the χ^2-random-variables ξ corresponding to the normalized innovations from Figure 16. Each value uses a whole innovation vector (82 entries for 41 landmarks) for each timestep of a measurement update. All the values are in a range of 120-240 with two values being above 240.

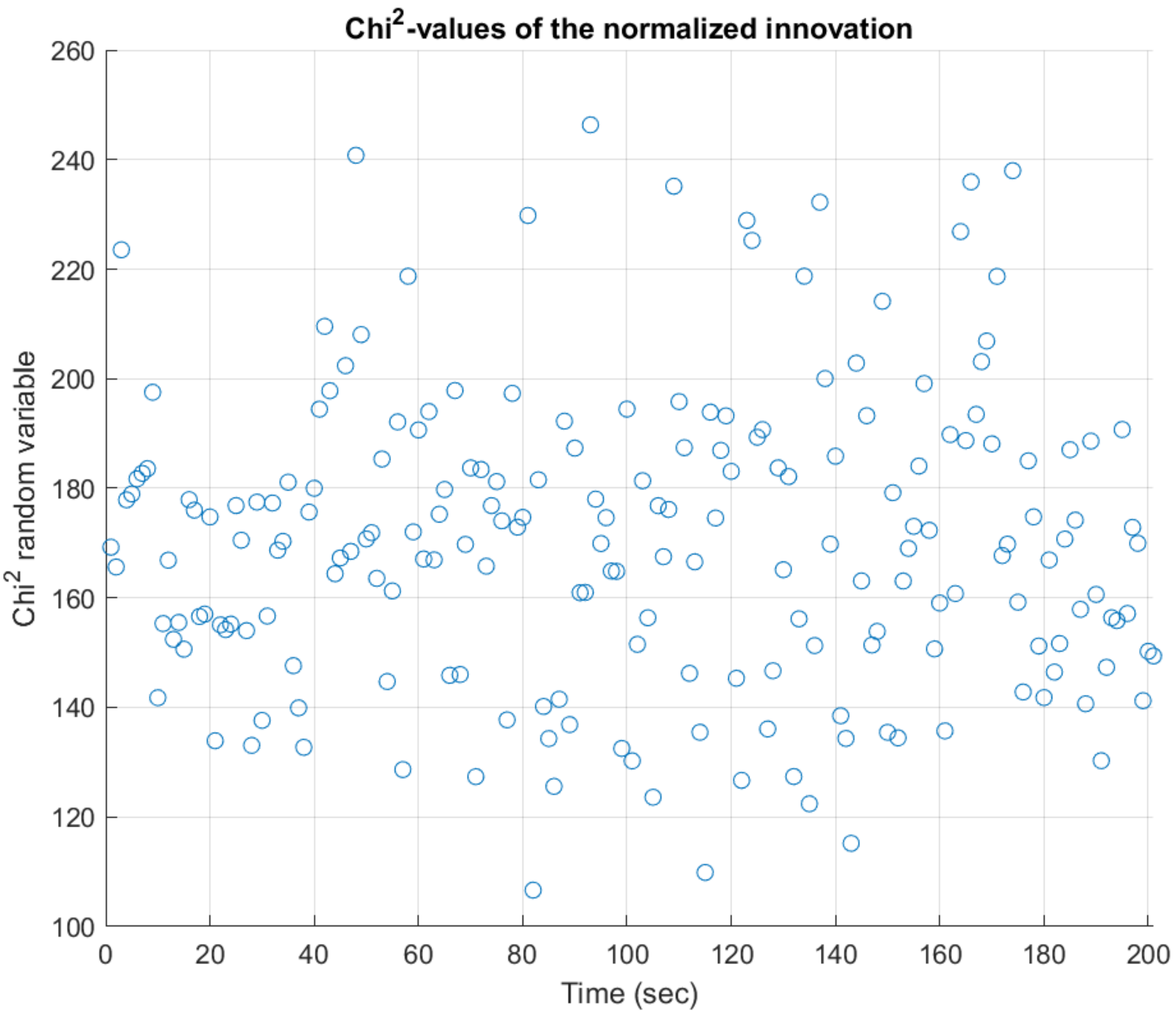

Figure 17: Chi-square-values for nominal simulation

The scenario has been run five times. For each simulation a moving average of the χ^2-random-variables has been made, which is illustrated in Figure 18. The average is calculated over a sliding window of length 20 across neighboring elements of the value under consideration. A smaller window results in higher fluctuations of the moving average and makes it more susceptible to noise. A larger window smooths the graph. The length of 20 has been chosen as a compromise between the two options. The moving average stays between approximately 150 and 190, with a minimum and maximum at 153 and 192. Over the course of each simulation, the average is roughly constant with some fluctuations.

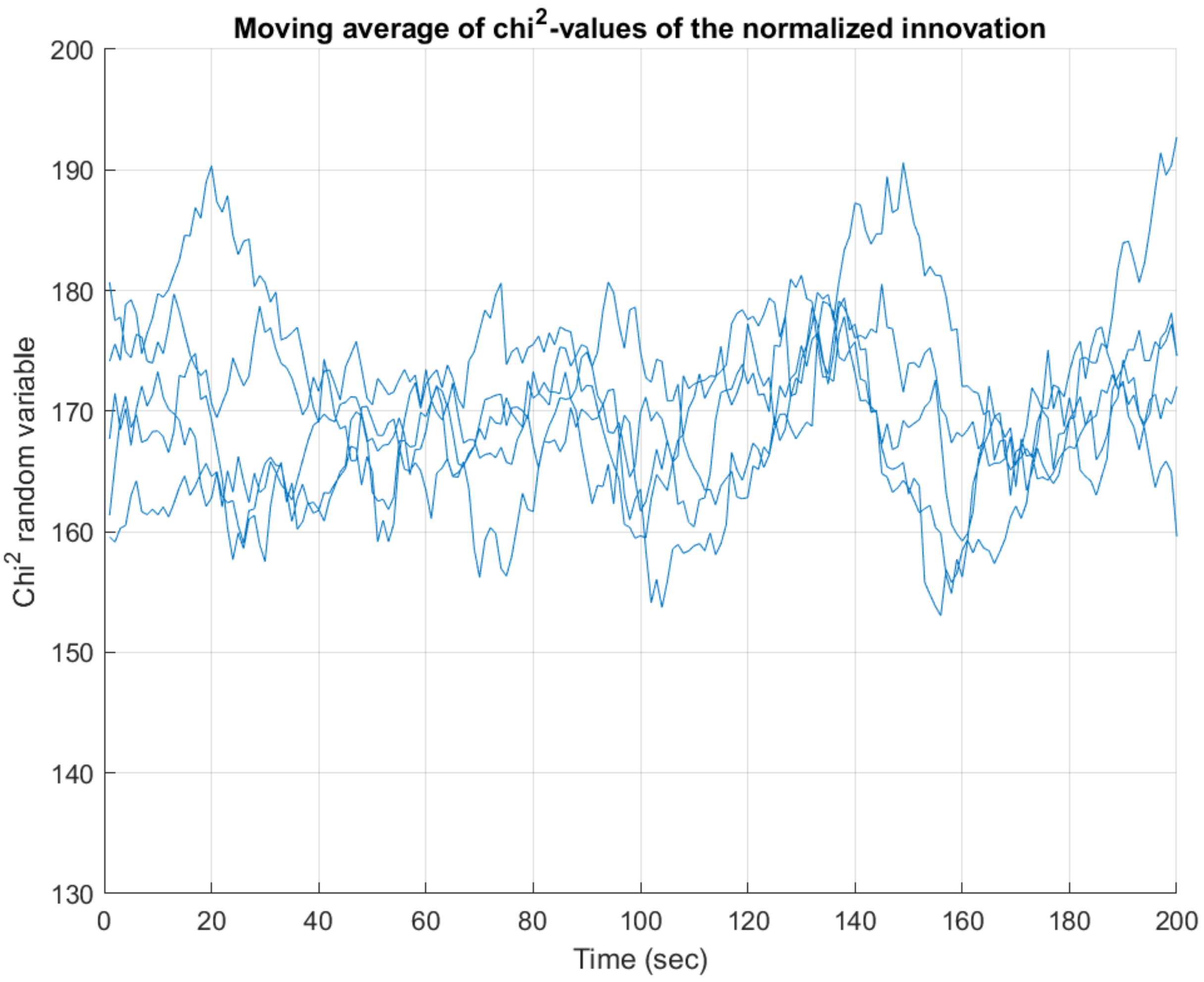

Figure 18: Moving average of chi-square-values of 5 nominal simulations

In Figure 19 the actual heading angle of the robot and the heading angle that has been estimated by the EKF over the course of the simulation are illustrated. After starting with the initial condition of a heading of 25°, the robot rotates towards the desired heading in approximately five seconds. From there on the actual heading stays relatively constant with fluctuations of $\pm 3°$ around 45°. The estimated heading fluctuates with approx. $\pm 7°$ around 45°.

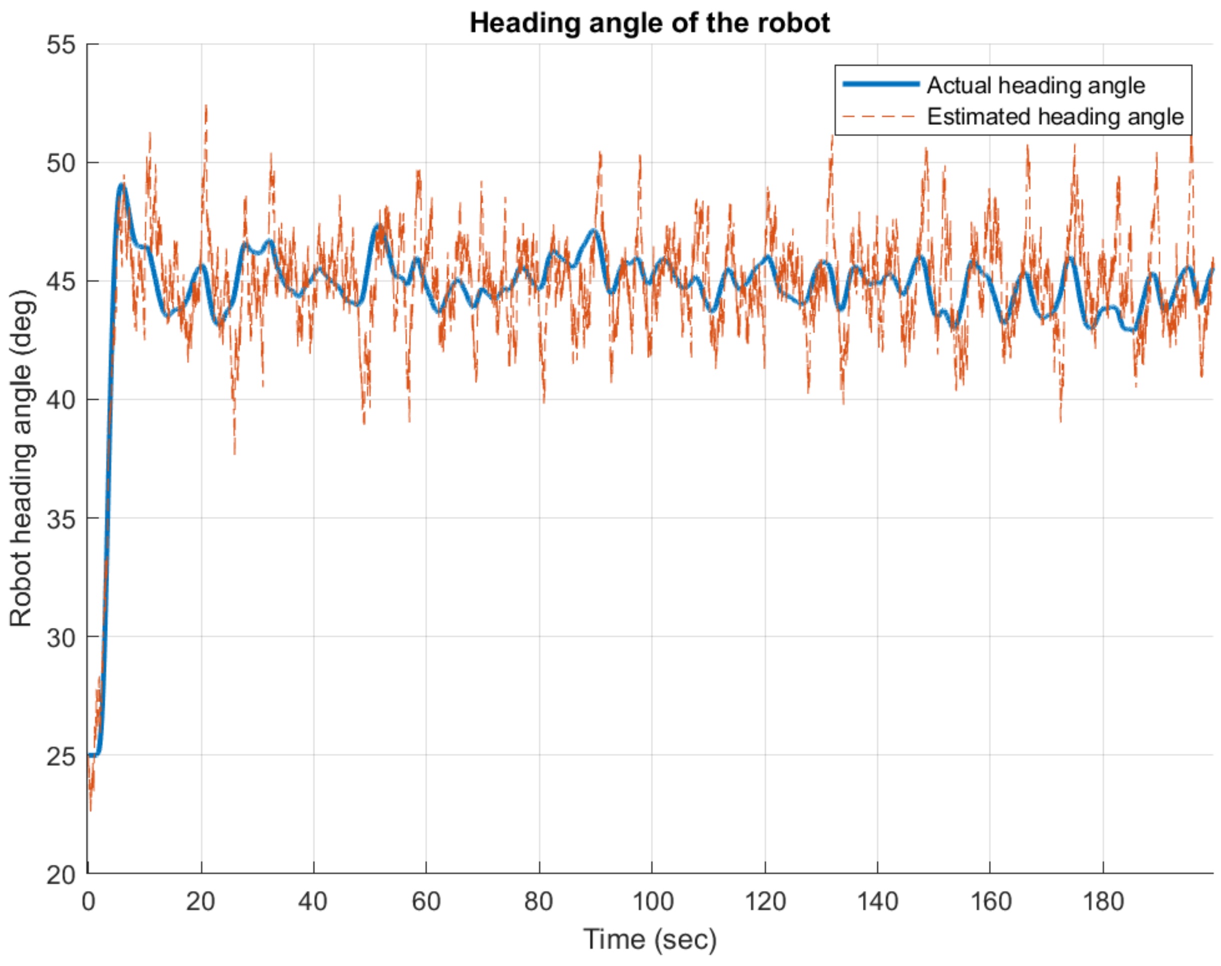

Figure 19: Heading angle of the robot for nominal simulation

4.2 Spoofed measurements with large sudden changes

The next simulation considers the rotation of the landmarks to alter the measured position and heading. This is a false data injection with large and sudden changes in the injected data Figure 20 shows the simulation environment with the unaltered and initial position of the landmarks marked with purple data points. Over the course of the simulation, the set of landmarks is rotated through four different angles around the origin. First, the set is rotated by 5° (blue data points) at zero seconds followed by another rotation of 7° (red data points) at 50 seconds. For the third step, the set is rotated by 13° (black data points) at 100 seconds and the final step is a rotation by 20° (green data points) at 150 seconds. This corresponds to a total rotation of 45°. The final position of the landmarks is indicated with green data points. The desired trajectory, is displayed as a black line and runs in a 45° across the plane. The estimated trajectory is shown as an orange dashed line. At every rotation of the landmarks the estimated trajectory jumps stepwise from the desired trajectory to the left. However, after each rotation step, the EKF estimated heading of the robot stays at 45°. This is the trajectory that is the final output of the EKF. The third trajectory is depicted in blue and marks the actual trajectory of the robot. It diverges from the desired trajectory to the right. Compared to the estimation there are no sudden changes in the trajectory, but the heading angle changes during each rotation step. The actual trajectory cannot be seen by the operator, who relies on a monitoring system to report changes in the operation of the system. The representation of the x and y-coordinate over time has been omitted, since they are very similar to the trajectories.

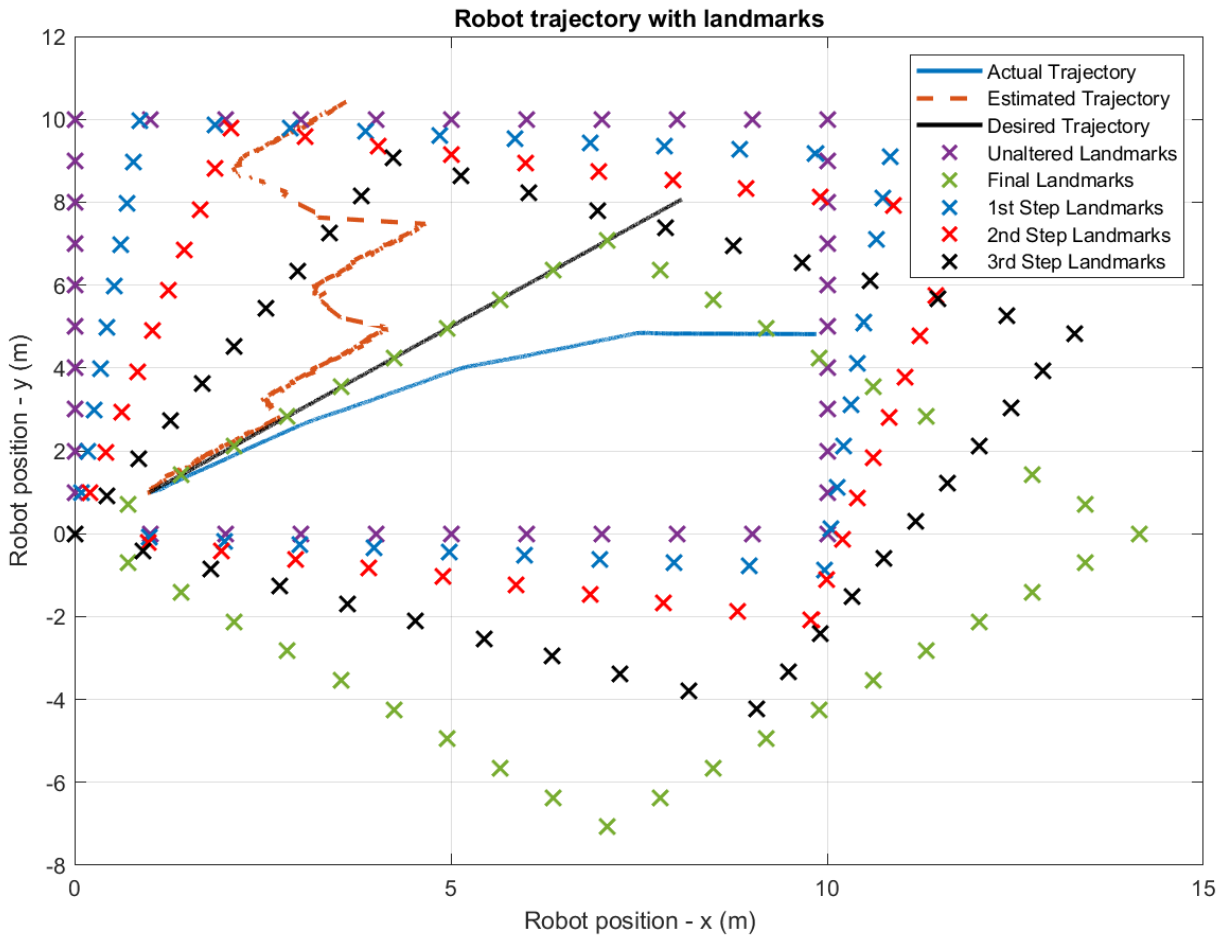

Figure 20: Robot trajectory with spoofed landmarks (spoofing occurring in 4 steps)

Figure 21 shows the normalized innovation of this scenario. For most of the time, the innovation values vary around ±3.8. However, at the instant of every rotation step, the innovation spikes with increasing magnitude. The first spike at zero seconds, which is induced by a rotation of 5°, is hardly noticeable. The second spike at 50 seconds (rotation by 7°) has a maximum of approximately 6.7 and a minimum of -5.9. During the third rotation of 13° at 100 seconds, the innovation spikes up to 13.2 and -15.6. The last spike at 150 seconds (rotation by 20°) has a maximum and minimum of approx. 25.9 and -17.2.

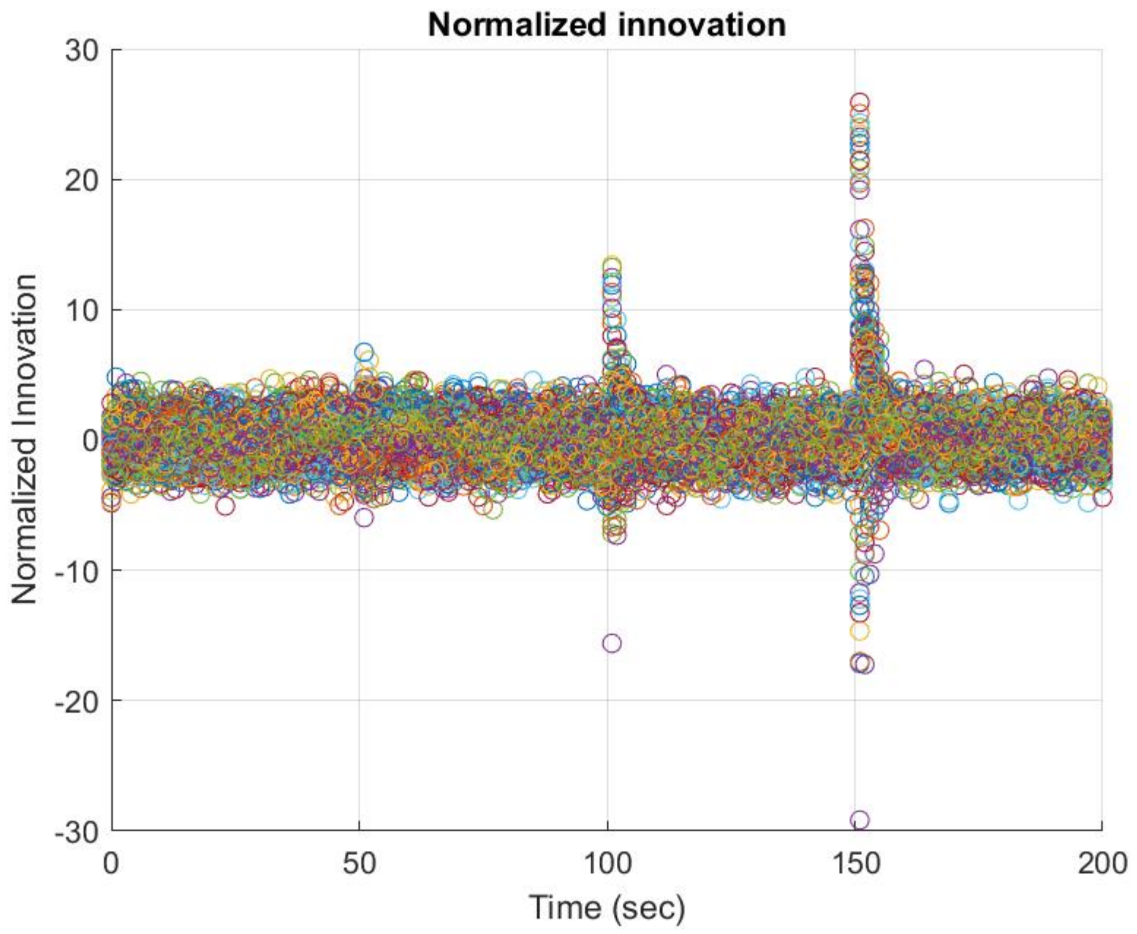

Figure 21: Normalized innovation for spoofing with large and sudden changes

In each case after the initial deviation, the innovation values decrease to the normal variation over the course of 5 seconds at the largest disturbance. The convergence of the innovation is illustrated in Figure 22, which is the enlarged at around 150 seconds, when the 20° bias was added to the measurement. At the third spike it takes the innovation about 4 seconds to reduce back to the expected range.

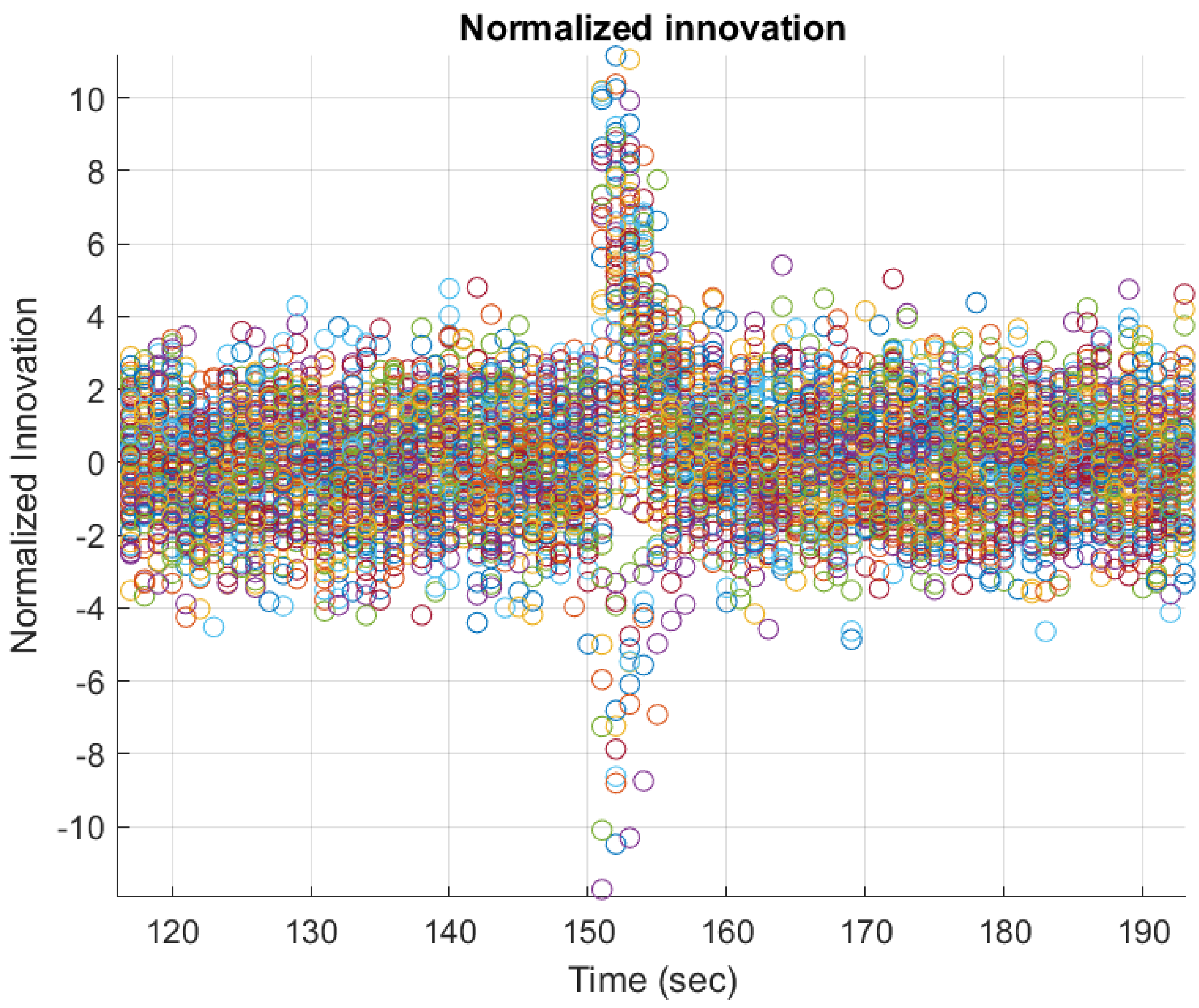

Figure 22: Zoomed innovation plot for final false measurement injection

The χ^2-random-variables that have been calculated from the normalized innovation are shown in Figure 23. As in the normalized innovation plot, the χ^2-random-variables can be seen spiking at each false data injection. In this illustration a small spike at zero seconds can be seen as well. Between the spikes, the χ^2 distribution is the same as in Figure 17. The first spike has a maximum of approx. 286. The second of 443. The third and fourth spike go up to 2541 and 12340 respectively. The gradual decrease already mentioned regarding the innovation plots can also be seen.

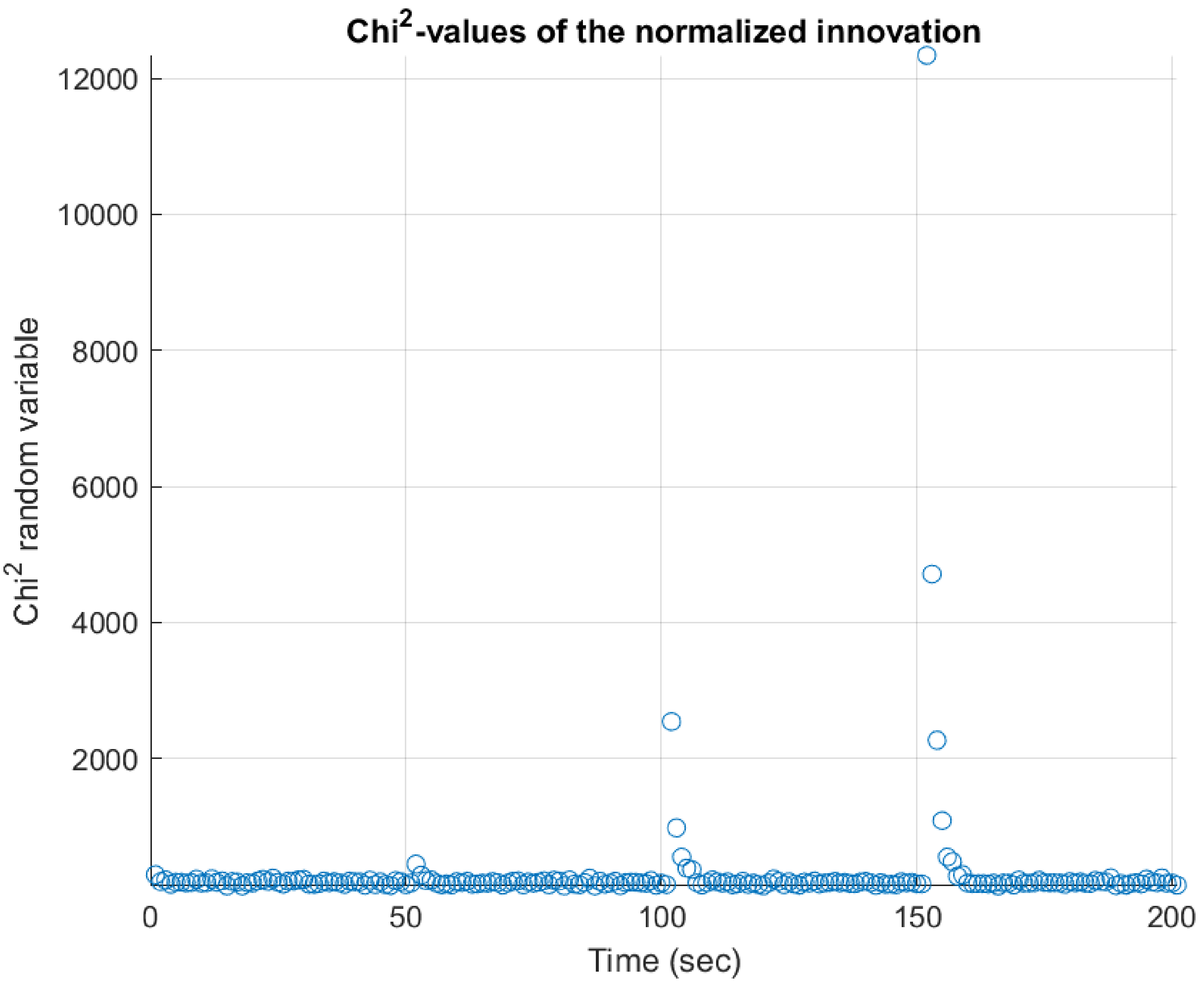

Figure 23: Chi-square-values for spoofing with large and sudden changes

Figure 24 shows the moving average of the χ^2-values of five different simulations. The moving average consists of a sliding window of 20 timesteps centered around the value considered and starts at approximately 195 and decreases after the first 10 timesteps noticeably to approx. 175. The second rotation can be as a small bump between 44 and 63 seconds at roughly 195. The following two spikes can be seen more clearly between 95 and 112 seconds at 395 and between 146 and 162 seconds at about 1180. The behavior during each simulation is roughly the same.

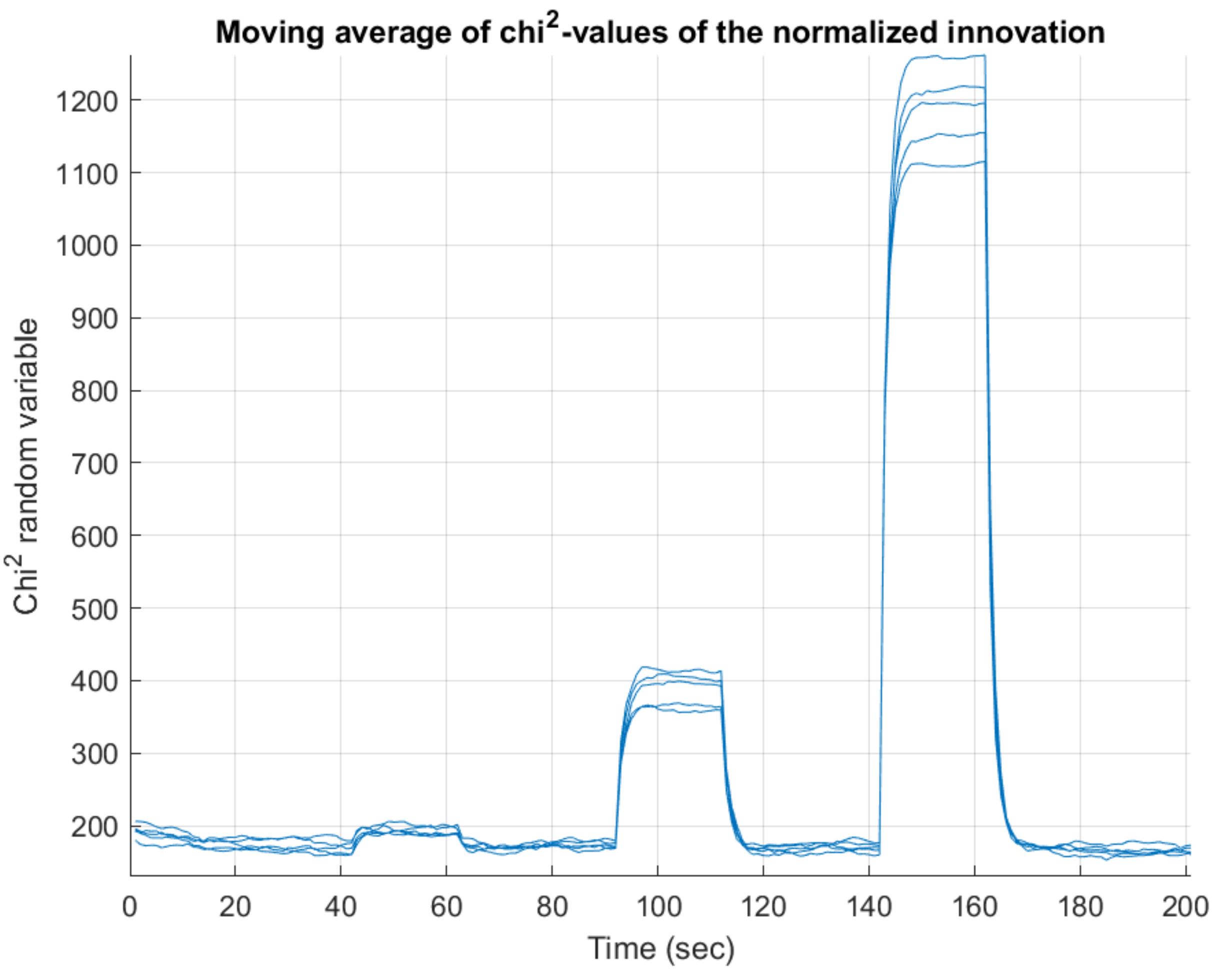

Figure 24: Moving average of chi-square-values of 5 for spoofing with large and sudden changes

In Figure 25 the estimated and actual heading angle of the robot is illustrated. After starting with an initial heading at zero seconds, the actual heading increases to approx. 40°. At 52 seconds, the heading decreases to 33°. The heading angle then jumps two more times at 102 seconds and 152 seconds to 20° and finally to 0°. Over the course of the whole simulation, the actual heading angle is subject to fluctuations of approx. ±3°. The estimated heading angle stays roughly constant over the course of the simulation with fluctuations of ±5° around the 45°. At 102 and 152 seconds spikes up to 62° and 73° can be seen. Potential spikes at the first and second rotation step can hardly be seen.

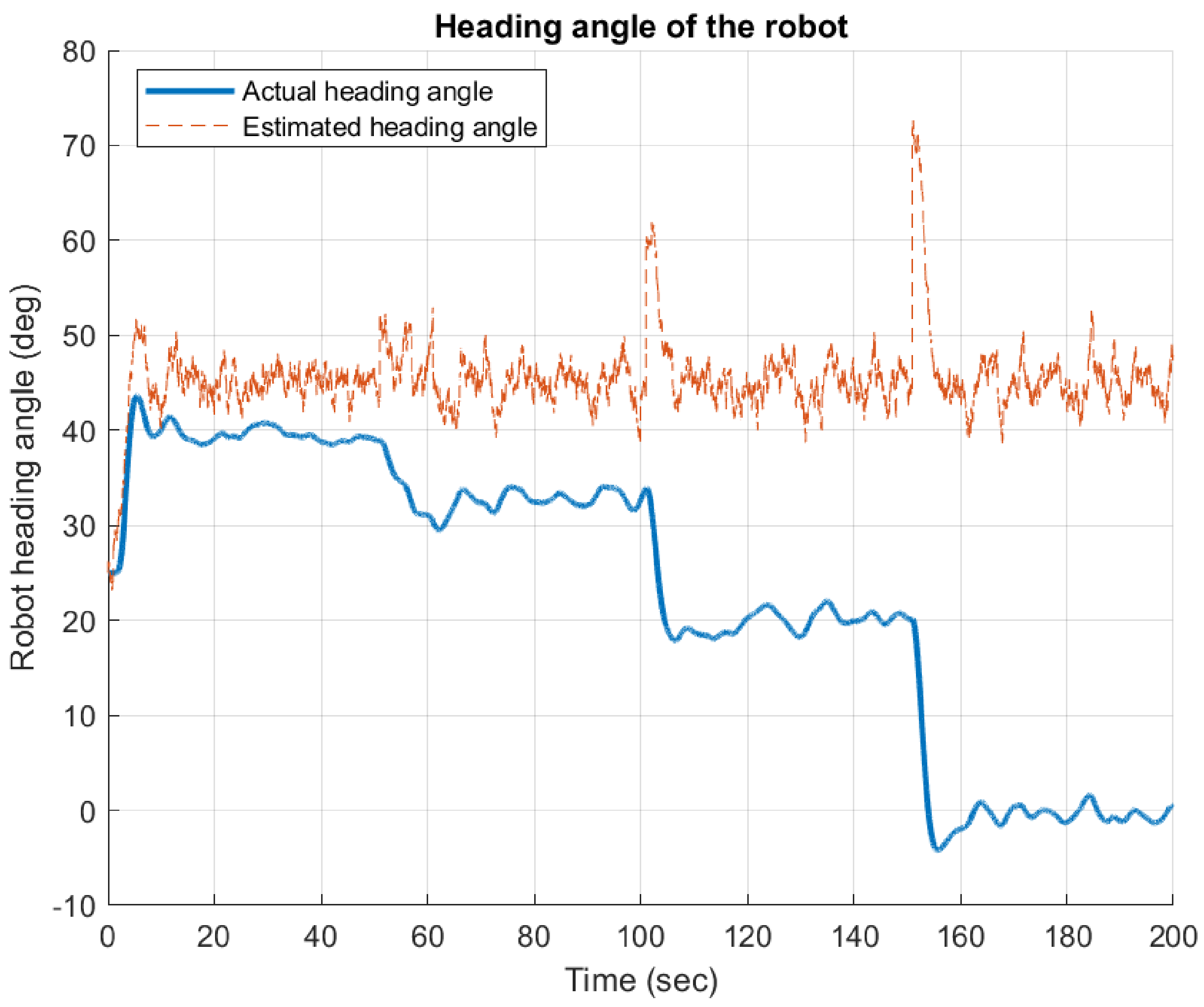

Figure 25: Heading angle of the robot for spoofing with large and sudden changes

4.3 Gradually spoofed measurements

The third scenario is a gradual rotation of the landmarks to alter the measured position and heading. The simulation environment is shown in Figure 26 with the unaltered and initial position of the landmarks marked with purple data points. Over the course of the simulation, the set of landmarks is rotated gradually around the origin until reaching 45° (marked with green data points). The rotation angle is calculated using the current timestep t.

$$\alpha_k = 45° \cdot \frac{t}{200} \tag{4.3.1}$$

Therefore, the rotation angle changes by 0.225° every measurement update.

As in the previous simulations, the desired trajectory, is a straight line at a 45° angle and is displayed as a black line. The estimated trajectory is shown as an orange dashed line and can be seen gradually diverging from the desired trajectory to the left. The third trajectory is depicted in blue and marks the actual trajectory of the robot. It gradually diverges from the desired trajectory to the right.

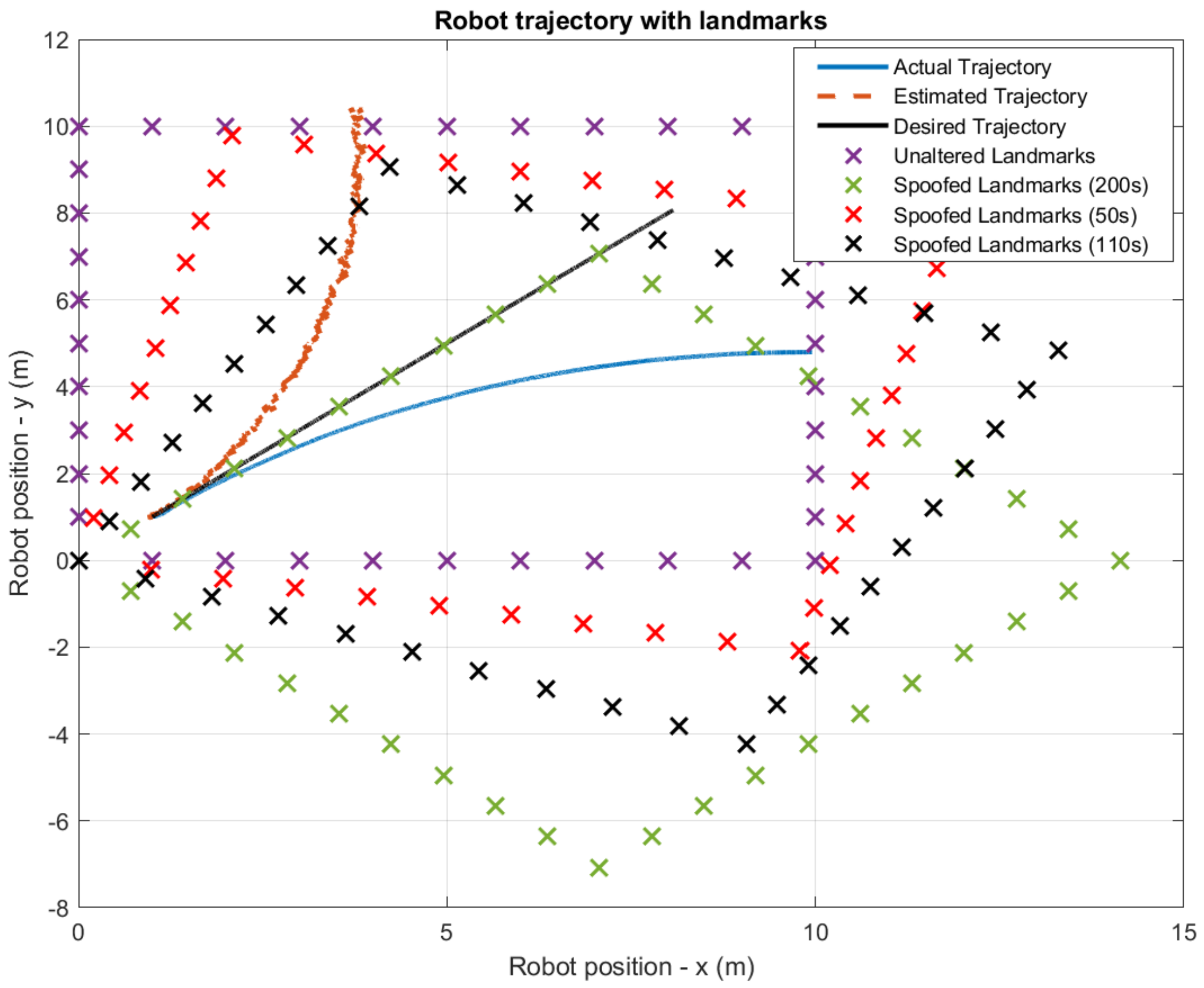

Figure 26: Gradually spoofed robot trajectory with landmarks

The normalized innovation as seen in Figure 27 ranges between -4 to 4 with a few innovation values being higher or lower. The density of the innovation value distribution increases when approaching zero as is expected for a normal distribution with zero mean. Over the course of the simulation, no big deflections can be seen and the distribution stays relatively constant.

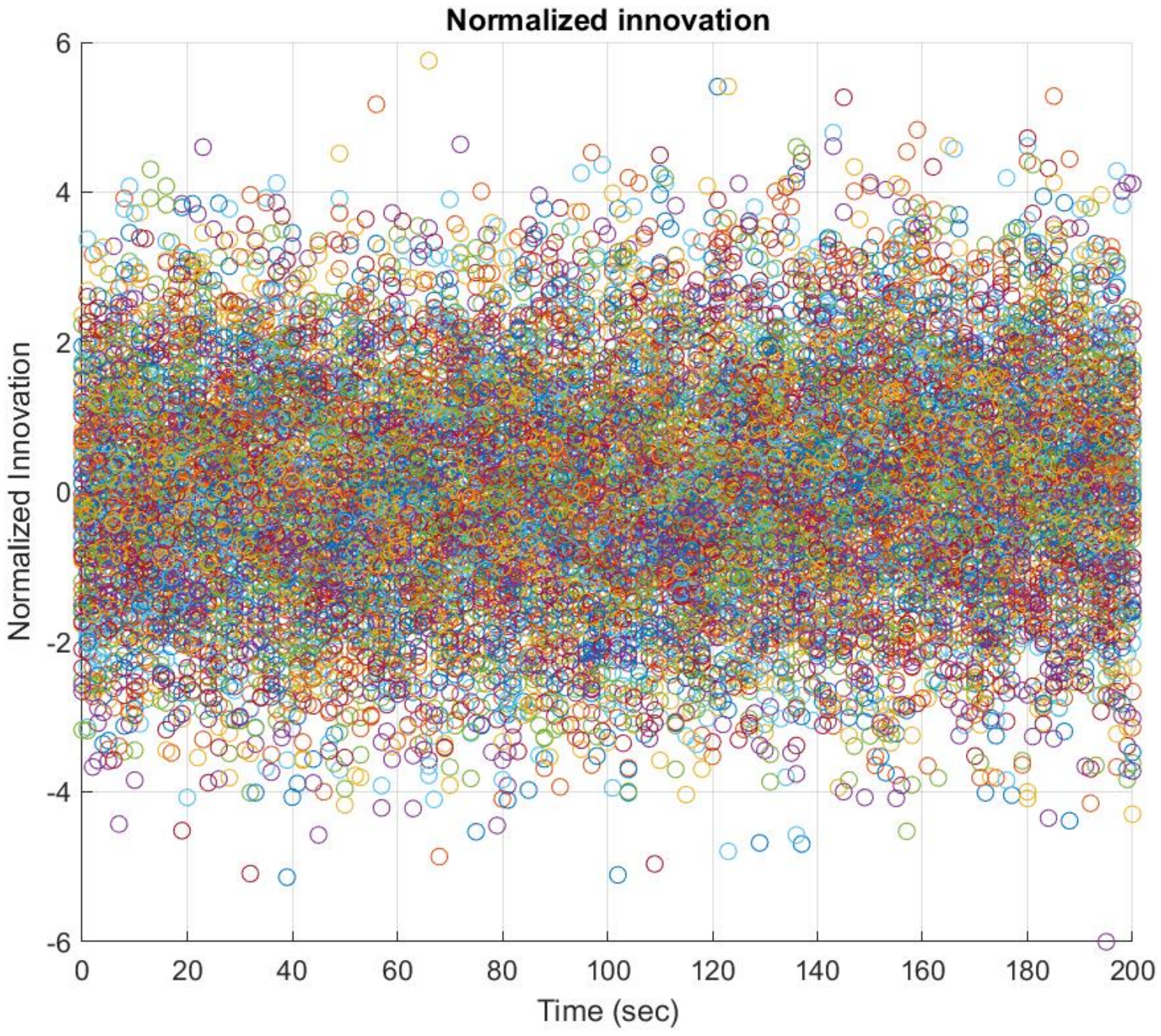

Figure 27: Normalized innovation for gradually spoofed simulation

The χ^2-random-variables of the normalized innovation can be seen in Figure 28. Most of the values lie in a range between 140 and 240 with a maximum of approximately 280 and a minimum of 110. Over the course of the simulation no discernible change in trend can is obvious.

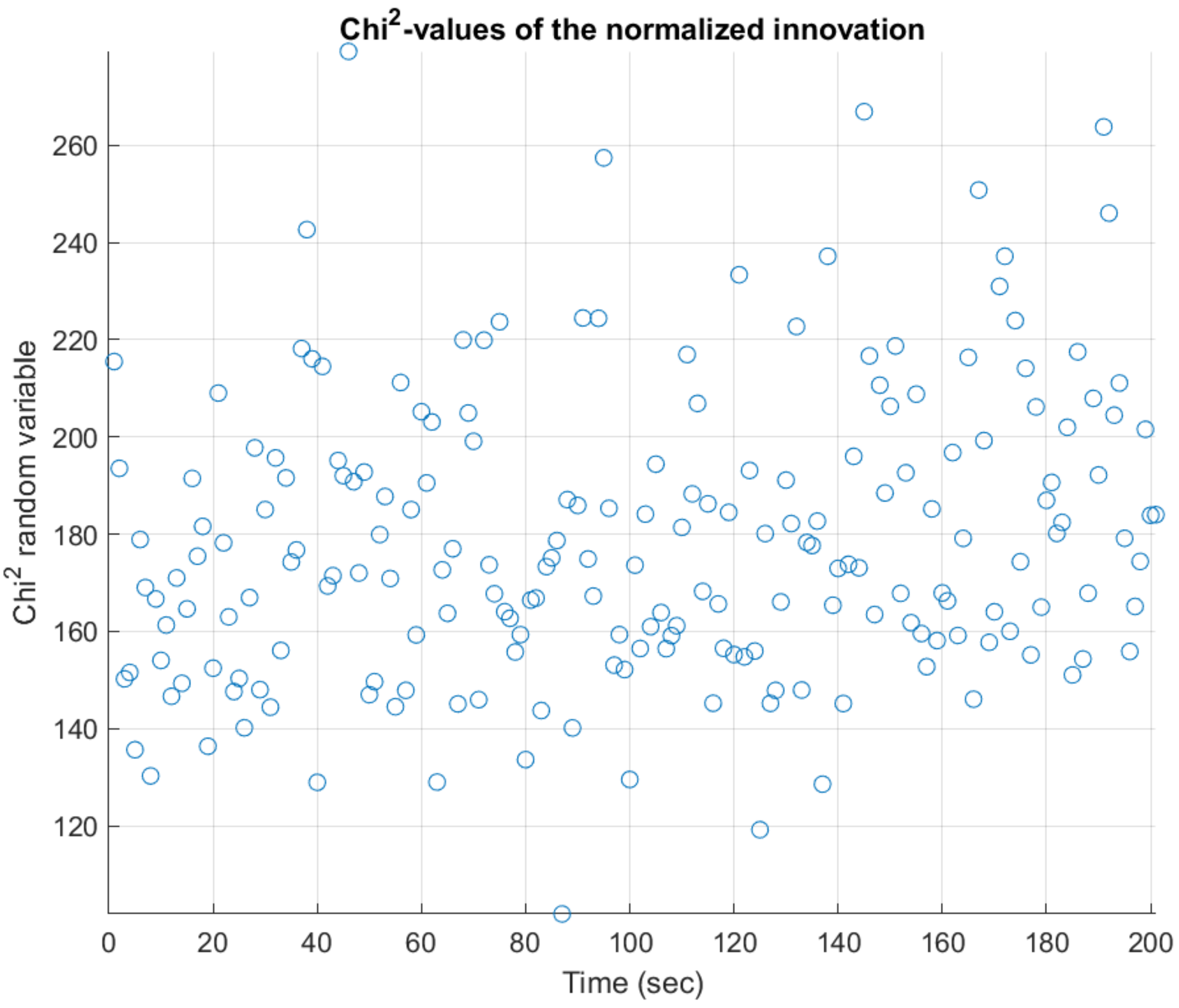

Figure 28: Chi-square-values for gradually spoofed simulation

The moving averages of the χ^2-random-variables of 5 simulations are illustrated in Figure 29. As before, the average was taken over the nearest 20 neighboring values of the corresponding value. Over the course of each of the simulations, the average stays roughly between 160 and 180. Overall there is a slight upward trend in the data. The moving average then stays roughly constant until it increases again at around 120 seconds until the end of the simulation.

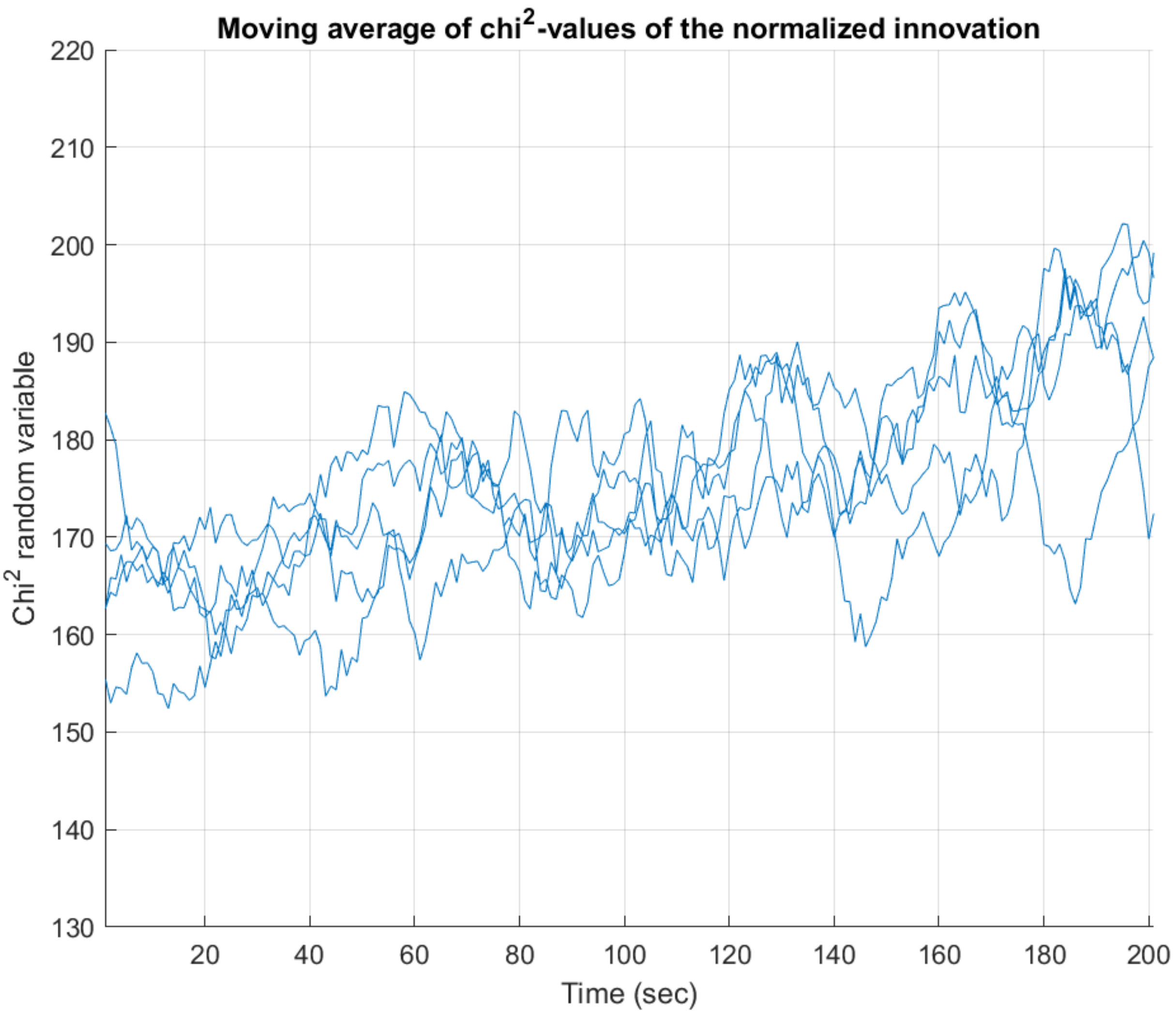

Figure 29: Moving average of chi-square-values of 5 gradually spoofed simulations

Figure 30 shows the actual heading angle (blue) and the estimated heading angle (orange, dashed). The estimated heading angle after increasing from the initial heading condition stays relatively constant over the course of the simulation at around 45° with fluctuation of ±7°. The actual heading angle increases from the initial condition to the desired 45° as well and overshoots to 49° at 6 seconds. After declining rapidly for 2 seconds to 43°, the heading angle gradually and linearly decreases with some fluctuations until the end of the simulation, reaching a heading angle of 2°.

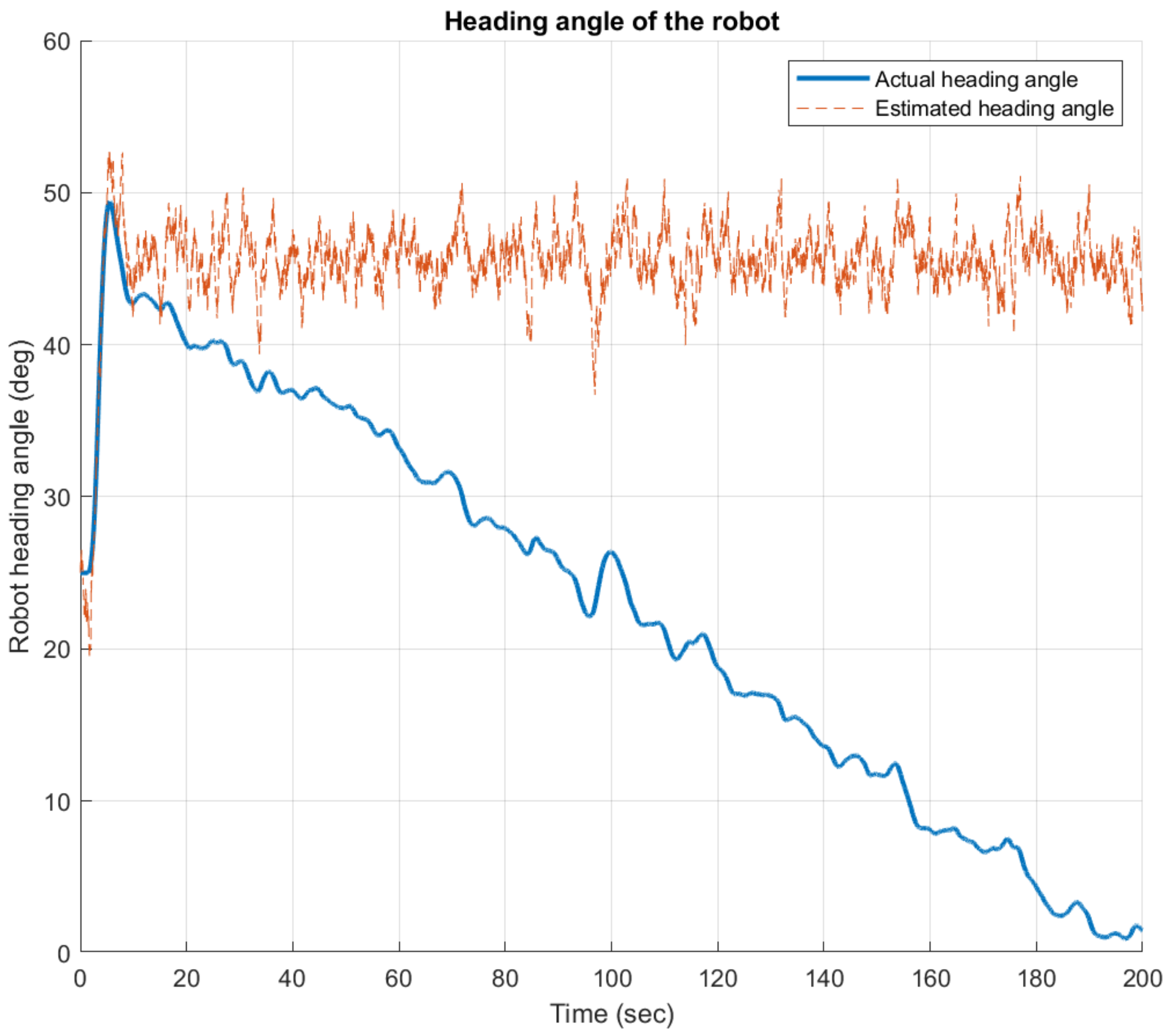

Figure 30: Heading angle of the robot for gradually spoofed simulation

5 Interpretation

In this chapter, results of the previously presented simulations will be interpreted. Based on the results conclusions will be drawn in the sense of the initial problem that was to be explored.

5.1 Nominal case

As expected, the robot is following its desired trajectory when the measurement device is not compromised. However, due to noise there are still some minor deviations that impact the prediction as well as the measurements. On the one hand, predictions are made on the assumption of a perfect environment. Since in a real-world application the surface may be uneven and the wheels might slip from time to time this has to be accounted for in the simulation as well by adding noise to the prediction model. The same applies of the measurements. A LiDAR and any other measurement device are subject to uncertainties that alter the measurements to a certain extent. The impact of all the uncertainties can be seen in the innovation and χ^2-plots. In a perfect environment, the expected innovation would be zero, since the predicted and the actual measurement would not be different from one another. However, as shown in Figure 16, the innovation looks more normally distributed, which can be expected since the noise models implemented are gaussian. The presented results are used as reference data for the following conclusions of the spoofed scenarios. After running approximately 100 simulations (only five have been shown for illustration purposes), the threshold has been chosen to be the moving average of the χ^2-random-variable of 195 as shown in Figure 31 (red horizontal line). The threshold serves the purpose of determining whether the robot is behaving correctly. If the χ^2-random-variable rises above this threshold, an alarm is triggered. Two simulations with the highest moving average have been included in the presented figure. Therefore, it is highly unlikely that the moving average in a nominal case would rise above the chosen threshold, which is well above the maximal averages. The lower boundary in this illustration is inconsequential, since the innovation values are squared. Therefore, lower χ^2-random-variables correspond to innovation values closer to zero and not high negative values.

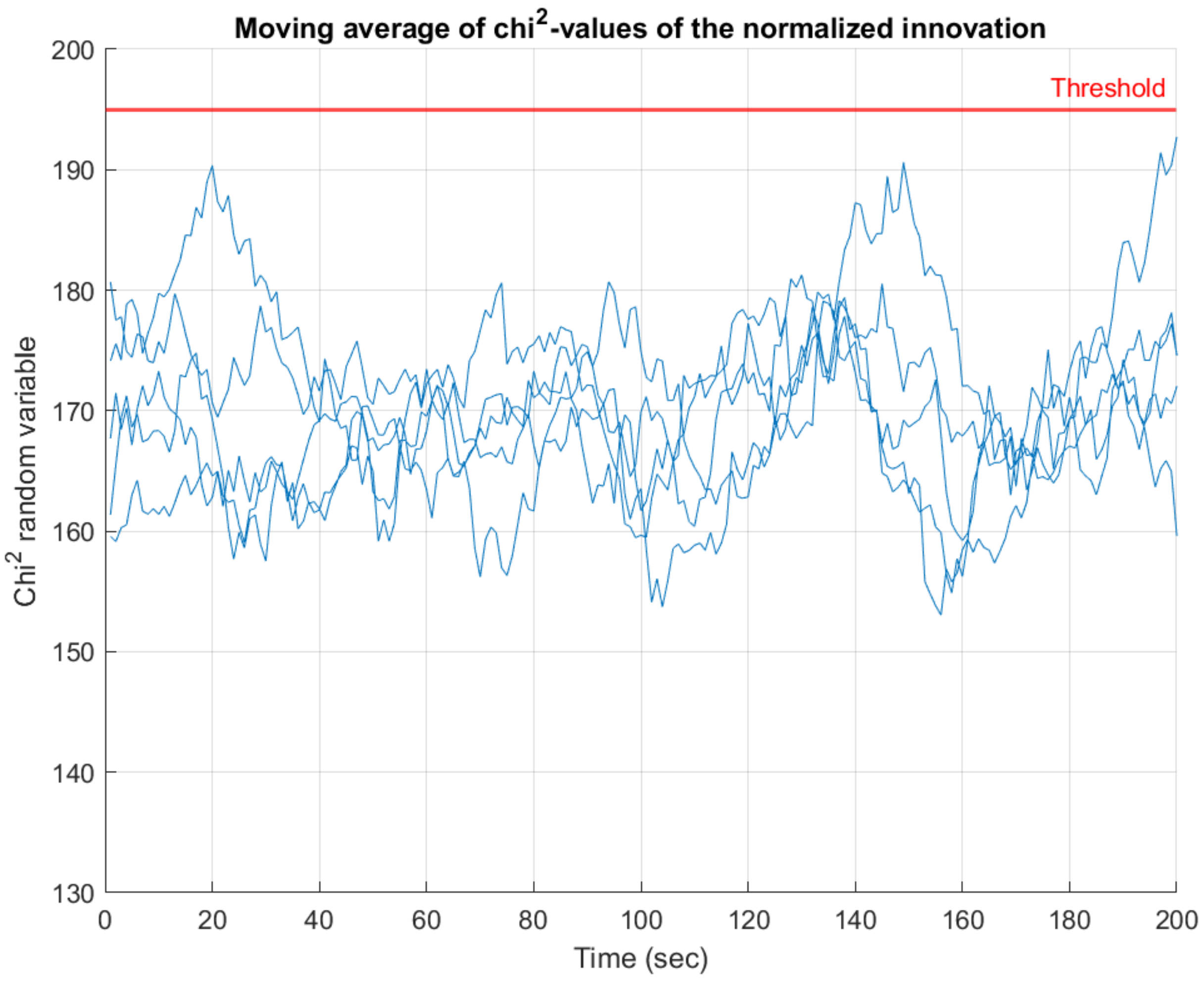

Figure 31: Moving average of the chi-square value with fault decision threshold

The heading angle as seen in Figure 19 expectedly fluctuates around the desired heading angle of 45°, which is due to noise. The estimated heading angle is seen fluctuating more than the actual heading angle, since the estimation is subject to noise and the actual heading—although driven by the estimation—is unaltered. This dependence of estimation and actual behavior of the robot can be seen in Figure 32, which is an enlargement of Figure 19. Once the estimated heading angle spikes up above 45° the robot accounts for that by lowering the heading angle and vice versa. This uncertainty produces an alternating motion around the 45° desired heading.

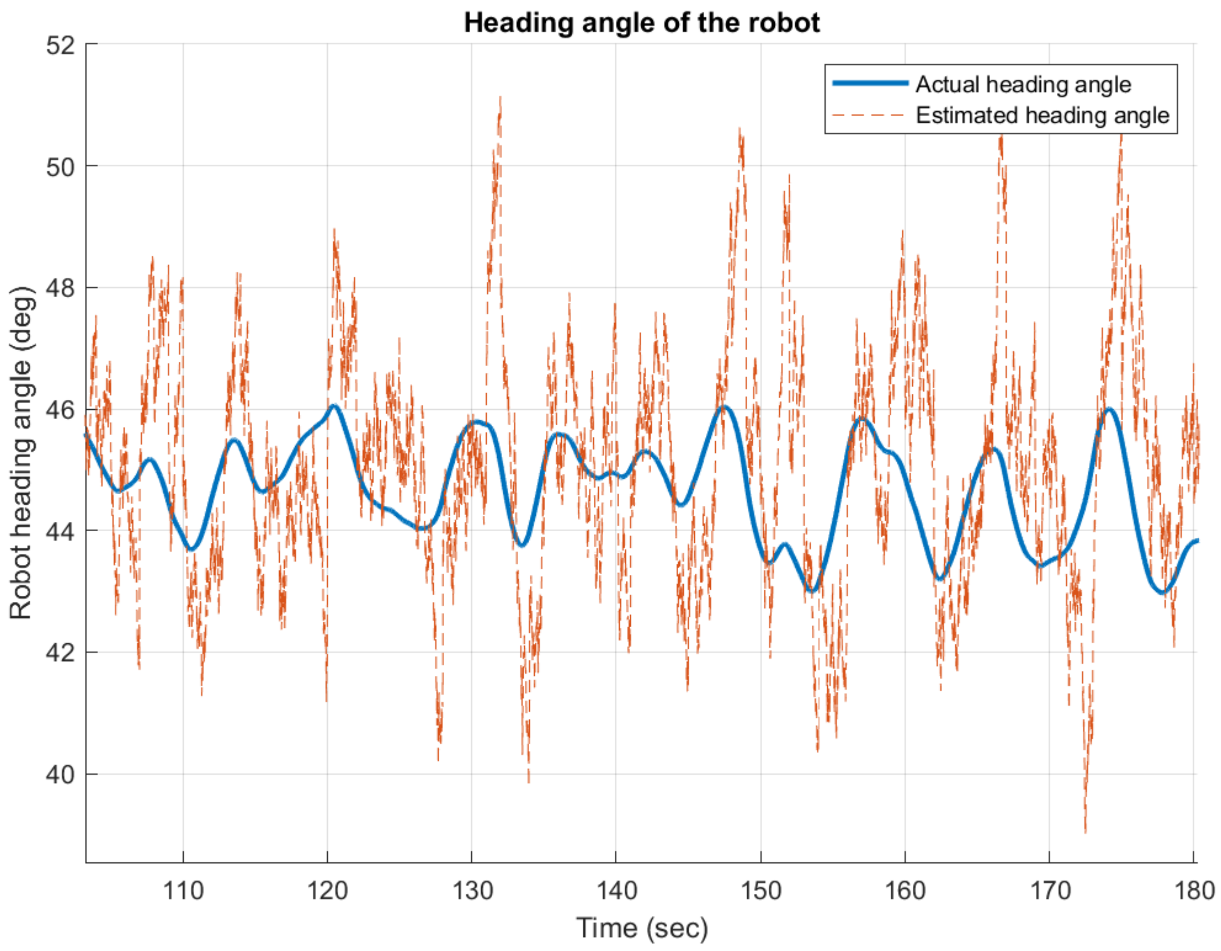

Figure 32: Enlargement of the heading angle for nominal simulation

5.2 Spoofed measurements with large sudden changes

The scenario with sudden measurement changes drastically differs from the nominal case, not only in terms of the trajectory but also regarding the innovation. In Figure 20 the estimated trajectory is seen jumping away from the desired trajectory during each rotation step. Due to the rotation around the origin the measured heading angle of the robot changes but also its position, which becomes clear when comparing the initial and spoofed landmarks. However, the estimated heading angle stays relatively constant at 45° between each step.

Note, that the steps marked red in Figure 33 occur just in a few measurement cycles, since the rotation of the landmarks is executed in just one timestep and the EKF then takes a few iterations to fully apply the changed measurement into the estimation process.

The actual trajectory differs from the estimated one due to the injection of false data. Since the only desired input, the operator is feeding into the system is the heading angle and the velocity, which is reflected by the robot's behavior, meaning the sudden change in heading causes the robot to turn in the opposite direction. However, the sudden difference in its measured position is not influencing the robot's behavior, since it is not a dictated system state. The actual heading of the robot changes each timestep to maintain 45° in the changed set of landmarks; therefore, the estimated heading stays roughly the same, since the heading based on the altered measurements during each timestep is still 45°.

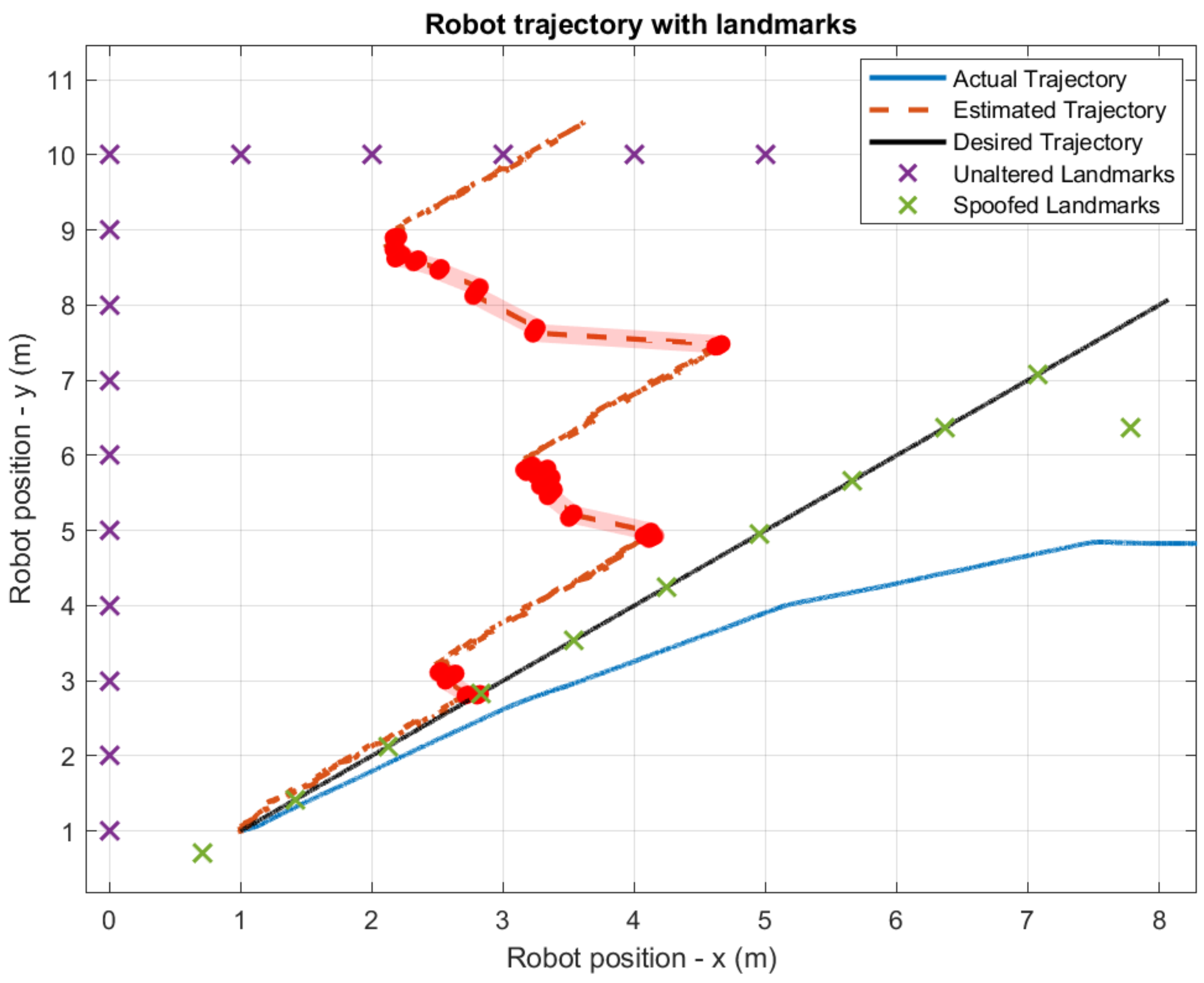

Figure 33: Marked rotation steps in trajectory

At each rotation the innovation and thus the χ^2-random-variables are distributed with increasing magnitude. This is due to the increasing change in heading, but can also be traced back to the change in distance because of the lengthening lever arm that gets bigger as the robot drives away from the origin.

Following each rotation, the innovation declines after approximately five seconds back to the level of the nominal performance. This is due to the nature of the Kalman Filter, the estimation of the system states usually converges after a few iterations up to a certain accuracy as long as the inputs don’t significantly change as seen in Figure 22. Consequently, once the altered measurement is introduced, the estimated measurement does not yet fully reflect that, since the prediction made with the transition model is also used to calculate the estimate. The new estimate, which is not consistent with the altered measurement is then used to calculate the next system state resulting in a prediction that still differs a lot from the measurement. That means that the sudden change of the measured positions is only taken fully into the estimation process after a few iteration (approximately five seconds).

The moving average of the χ^2-random-variables compared to the fault detection threshold marked in red paints a clear picture (Figure 34). Especially the average during the last two rotation steps rises significantly above the previously defined threshold. The second rotation does not cause the average to rise above the threshold in all of the displayed simulations, although it stands out compared to surrounding timesteps. Still the slight peak during the second rotation can be seen in Figure 23. Shrinking the window, the average is derived from, could resolve this issue on the one hand but would also make the detection mechanism more susceptible to false alarms due to noise induced fluctuations. This implies that if the spoofed measurements introduced by the false data injection are small enough, they may not be detectable. That is, it may be impossible to separate the spoofing inputs from the noise.

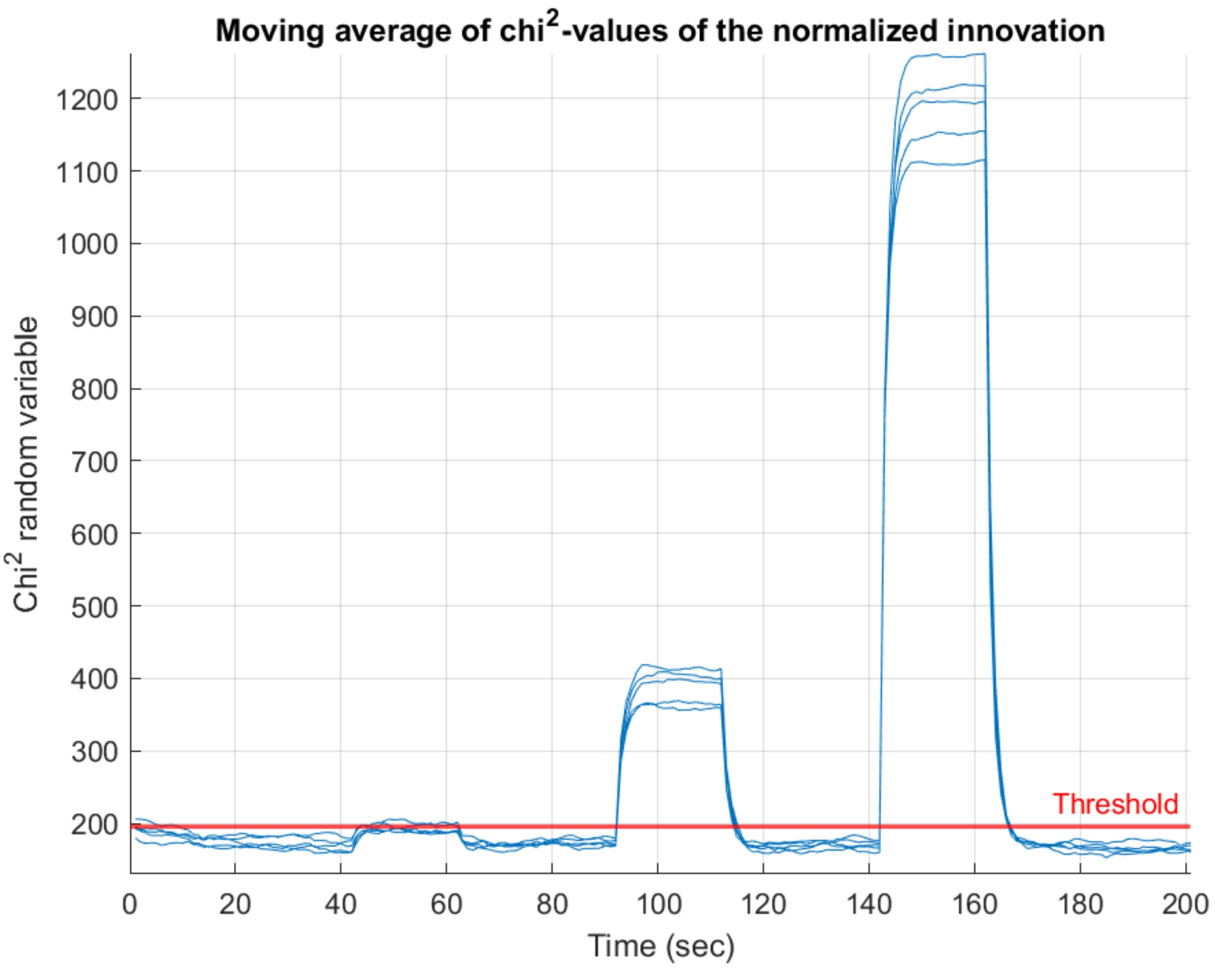

Figure 34: Moving average of the chi-square value with threshold for spoofing with large and sudden changes

The estimated and actual heading angle as displayed in Figure 25 shows stepwise inconsistencies. The first rotation takes place at the beginning of the simulation. After the initial condition is corrected, the estimated angle overshoots because of the first rotation of 5°. The robot immediately accounts for that, by adjusting its heading angle to 40°. The rotation spikes can be observed best at 100 and 150 seconds. At each rotation, the estimation of the heading angle spikes for a few measurement cycles. Since the operators input dictates a heading of 45°, the robot turns, causing the actual heading angle to decrease even lower and the estimated heading angle to decrease to the desired 45°. Once the estimated heading reaches 45° again, the robot stops turning and only makes small heading adjustments to compensate for the small fluctuations in heading caused by the noise.

5.3 Gradually spoofed measurements

Both the actual and estimated trajectory can be seen diverging from the desired one as seen in Figure 26. Because of the gradual rotation, the estimated heading angle in relation to the new and spoofed set of landmarks changes, causing the robot to accommodate by turning in the opposite direction. The impact on the robot is as severe as during the simulation with sudden measurement changes.

However, when looking at Figure 27, the normalized innovation looks similar to the nominal simulation and just by looking at the plot, no significant difference can be observed. The χ^2-random-variables in Figure 28 occasionally rise well above 246, which was the maximum during the nominal scenario, but mainly stay in the range between 120 and 240 as in the nominal experiment.

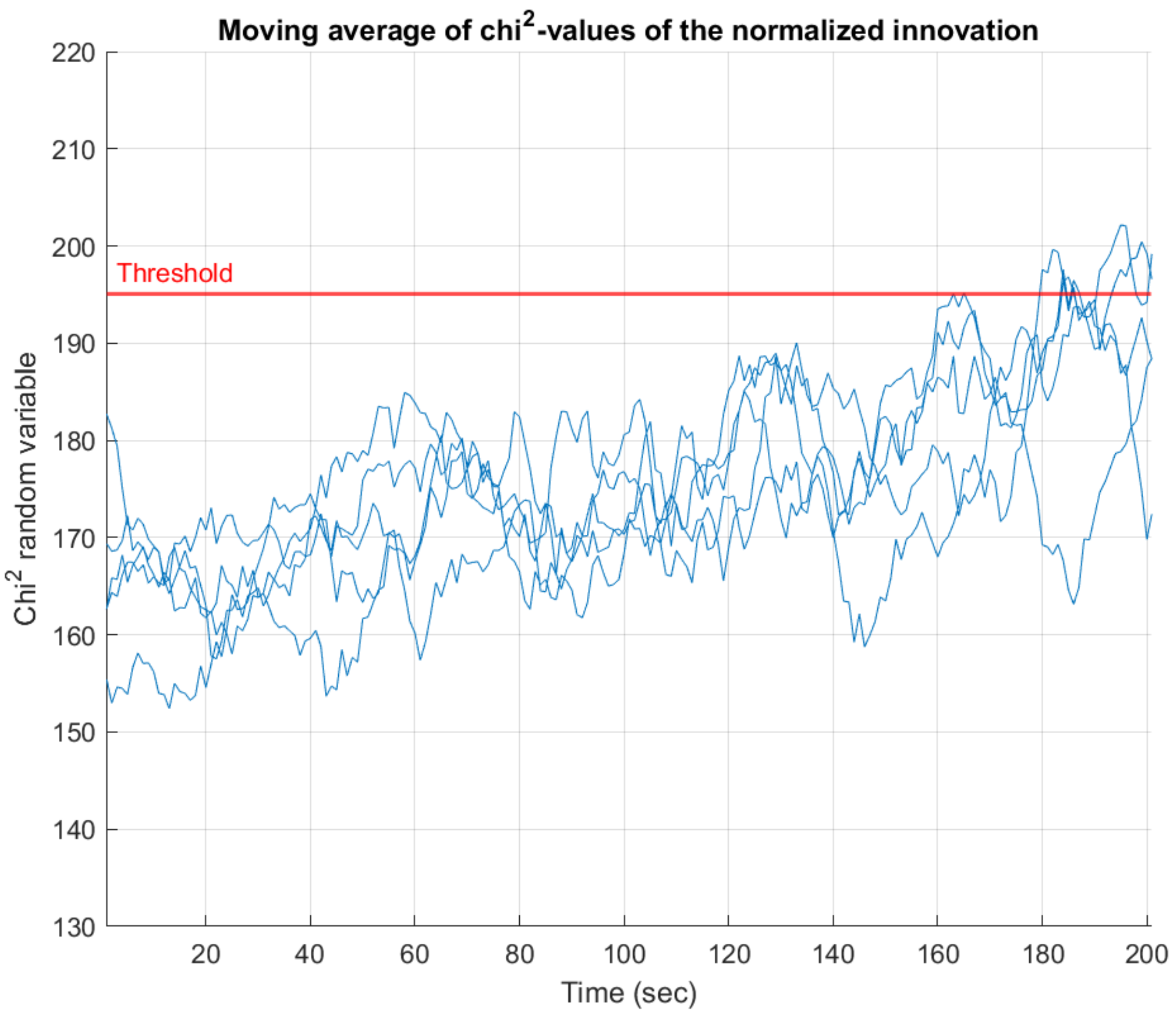

Figure 35: Moving average of the chi-square value with threshold for gradually spoofed simulation

Looking at the moving average with the defined threshold in Figure 35, the average can be seen rising above it, but only after the RV has moved far off the desired course. What is particularly noticeable are the increases at 40 seconds and 180-190 seconds. The second increase could be explained by the positioning of the landmarks seen in Figure 36. Shown is the landmark location at 190 seconds (green). The robot can be seen passing a landmark really close at the end of the trajectory (blue). While passing a landmark in very close proximity, the orientation angle of this specific landmark changes drastically over a short period of time, making fluctuations due to noise more impactful and thus the measurements and predictions less accurate.

The rise at 40 seconds, however, cannot be explained this way, since during this time period no landmark is in close proximity to the robot. Generally speaking a spike above the given threshold can always occur, when the gaussian noise of prediction model and measurements and the gradual spoofing constructively interfere with each other, to add up to a high difference in prediction and measurement. Still it is surprising that all of the simulations shown in Figure 35 behave in a relatively similar way.

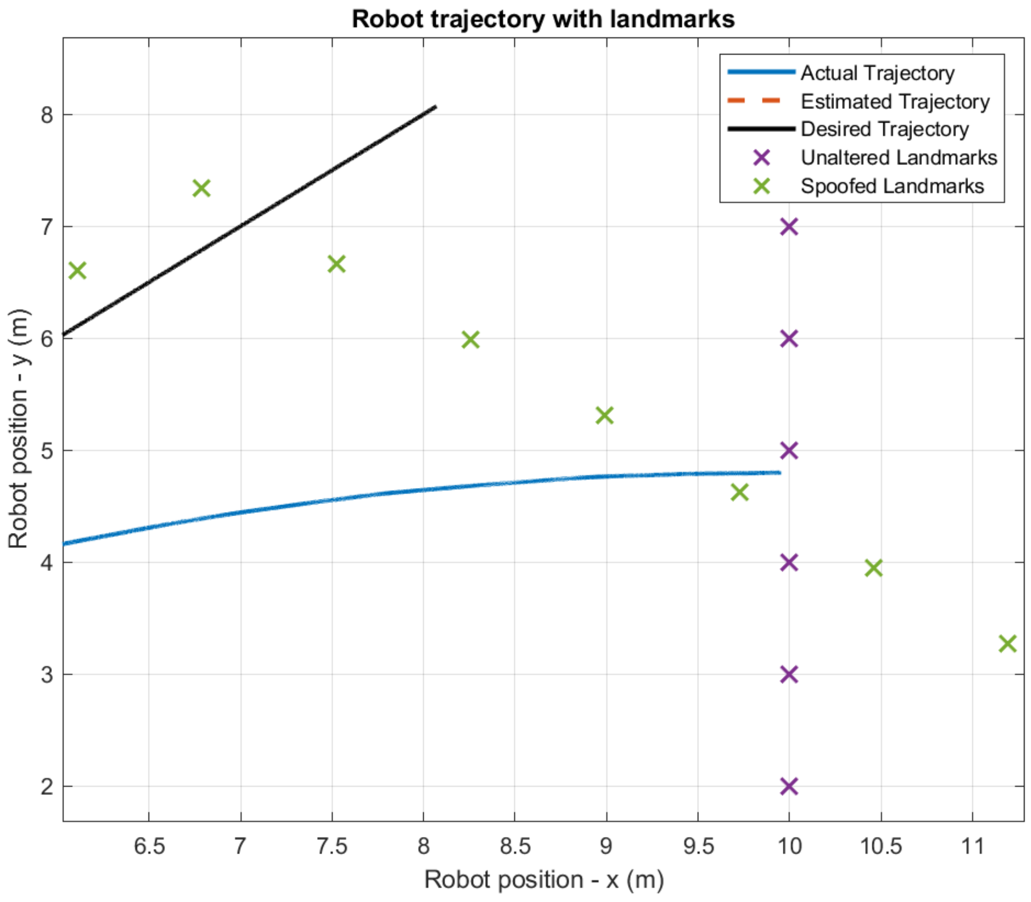

Figure 36: Gradually spoofed trajectory with spoofed landmarks at 190 seconds

The estimated and actual heading angle in Figure 30 are consistent with the observations made earlier. The actual heading angle gradually declines to account for the rotation of the landmarks thus maintaining the estimated heading angle at approx. 45°. The estimated trajectory can still be seen diverging, since the estimation is partially based on the measurement, which is based on the spoofed measurements.

Generally speaking, the gradual approach of injecting false data is really hard to detect. Although the moving average in each nominal simulation did not rise above the threshold, it cannot be said with complete certainty that it generally wouldn’t. Meaning that there is still a risk of triggering false alarms. On the other hand, the probability of detecting the false data attack is not nearly at 100%. In the selection of simulations shown in Figure 35, only four out of five attacks would have been detected. It is the user’s decision of either risking more false alarms or detecting less attacks. Still it has to be stated, that the attack can also be tuned by i.e. decreasing the rotation rate of the landmarks and thus making the attack even harder to detect.

It has been demonstrated that it is possible—with the appropriate procedure—to disturb the nominal performance of the robotic system in a way that could be quite difficult to detect.

6 Conclusion and outlook

To conclude the findings, it can be stated that it is generally possible to detect a false data injection attack on a RV’s measurements with an EKF and a fault detection mechanism, as long as the difference between prediction and measurement and thus the innovation is high enough. This has been the case when the measurement is changed through false data injection in a sudden and severe manner. To pick up the initial thesis it is certainly possible to construct an attack that is nearly undetectable with the measures introduced. Not only can the attack be considered stealthy, the impact on the robot’s behavior is also severe causing it to diverge significantly from its desired trajectory. As long as the difference between predicted and measured system state is small (in the order of magnitude of the noise) it may be difficult for a fault detection mechanism to distinguish intentionally altered measurements from noise. However, a subtle difference between the nominal case and the gradually spoofed simulations (small injected deviation each timestep) can still be observed. To increase the capabilities of detecting an attack the threshold has to be lowered. This can be achieved by using for example more sophisticated measuring devices or by improving the model used for the prediction. This way on average the innovation could be reduced resulting in lower χ^2-random-variables enabling the operator to lower the fault detection threshold. However, the attacker could in turn adapt to the lowered threshold by reducing the amount of deviation each timestep introduced into the system via false data injection. For example, by rotating the artificial, spoofed landmarks at a slower rate. Another approach could be to implement multiple different measuring devices, for example a GNSS receiver to determine the position. This would create another layer of redundancy but in the end these additional measuring devices could be attacked as well.

Future work could look at different alterations of the spoofed landmarks. The possibilities of the homogeneous transformation used to change the measurements have been discussed in section 2.4. Conceivable would be gradual stretching, sheering or translation. A constant rotation around the robot (and not the origin) would be interesting as well, although more difficult to implement. Another interesting subject could be the calculation of the probabilities of triggering false alarms. Since the noise implemented into the measurements and the model used for the prediction are normally distributed, a statement can be made on how high the innovation (which itself is normally distributed) and thus the χ^2-random-variables are with what probability. This can be compared to different fault detection thresholds. For each threshold a statement could be made on how likely a false alarm would be.

The natural next step, however, would be an implementation into a simulation environment for the *TurtleBot*. The manufacturer recommends the simulation environment from *Gazebo* using the *Robotic Operating System* (ROS) [3]. After that, an implementation into a physical *TurtleBot 3 ‘Burger’* or even other two-wheeled differentially steered robots could be done.

Sources

[1] K. Kim, J.S. Kim, S. Jeong, J.-H. Park, H.K. Kim, Cybersecurity for autonomous vehicles: Review of attacks and defense, Computers & Security 103 (2021) pp. 102–150. https://doi.org/10.1016/j.cose.2020.102150.

[2] D. Morris, G. Madzudzo, A. Garcia-Perez, Cybersecurity threats in the auto industry: Tensions in the knowledge environment, Technological Forecasting and Social Change 157 (2020) pp. 102–120. https://doi.org/10.1016/j.techfore.2020.120102.

[3] Open Source Team, TurtleBot 3 E-Manual. https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (accessed 13 July 2022).

[4] P. Dash, M. Karimibiuki, K. Pattabiraman, Stealthy Attacks against Robotic Vehicles Protected by Control-based Intrusion Detection Techniques, Digital Threats 2 (2021) pp. 1–25. https://doi.org/10.1145/3419474.

[5] F. Dušek, D. Honc, P. Rozsíval, Mathematical model of differentially steered mobile robot, in: 18th International Conference on Process Control 14. - 17.06.2011.

[6] R. Permana Saputra, Implementation 2D EKF-Based Simultaneous Localisation and Mapping for Mobile Robot. UNSW Australia Mechanical and Manufacturing Engineering Masterthesis, (2015).

[7] J. Čerkala, T. Klein, A. Jadlovska, Modeling and Control of Mobile Robot with Differential Chassis, Electrical Engineering and Informatics 6 proceedings of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice (2015) pp. 651–656.

[8] A.A. Housein, G. Xingyu, W. Li, Y. Huang, Extended Kalman Filter Sensor Fusion in Practice for Mobile Robot Localization, IJACSA 13 (2022). https://doi.org/10.14569/IJACSA.2022.0130204.

[9] S.J. Julier, J.K. Uhlmann, New extension of the Kalman filter to nonlinear systems, in: Signal Processing, Sensor Fusion, and Target Recognition VI, Orlando, FL, USA, SPIE, 1997, p. 182.

[10] R.E. Kalman, A New Approach to Linear Filtering and Prediction Problems, Journal of Basic Engineering 82 (1960) pp. 35–45. https://doi.org/10.1115/1.3662552.

[11] M.S. Grewal, A.P. Andrews, Kalman filtering: Theory and practice using MATLAB®, Fourth edition, Wiley, Hoboken, New Jersey, 2015.

[12] P. Kim, Kalman filter for beginners: With MATLAB examples, Createspace, s.l., 2011.

[13] D. Simon, Optimal state estimation: Kalman and nonlinear approaches, Wiley-Interscience, Hoboken, N.J., 2006.

[14] I. Reid, Lecture Notes Estimation 2, 2001. https://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf (accessed 25 July 2022).

[15] A. Becker, Kalman Filter in one dimension. https://www.kalmanfilter.net/kalman1d.html (accessed 24 August 2022).

[16] M. Fazekas, P. Gáspár, B. Németh, Calibration and Improvement of an Odometry Model with Dynamic Wheel and Lateral Dynamics Integration, Sensors (Basel) 21 (2021). https://doi.org/10.3390/s21020337.

[17] Eckhardt Optics, Light Source Measurement. https://www.eckop.com/applications/light-source-measurement/ (accessed 27 July 2022).

[18] C. Weitkamp, Lidar: Range-Resolved Optical Remote Sensing of the Atmosphere, Springer Science+Business Media Inc, [New York], 2005.

[19] R. van der Heijden, T. Lukaseder, F. Kargl, Analyzing attacks on cooperative adaptive cruise control (CACC), in: 2017 IEEE Vehicular Networking Conference (VNC), Torino, IEEE, 2017, pp. 45–52.

[20] D. Marsh, Applied geometry for computer graphics and CAD: With 27 figures, secondnd ed., Springer-Verlag, London, 2005.

[21] D.I. Urbina, J.A. Giraldo, A.A. Cardenas, N.O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, H. Sandberg, Limiting the Impact of Stealthy Attacks on Industrial Control Systems, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna Austria, ACM, New York, NY, USA, 2016, pp. 1092–1105.

[22] R. Da, Failure detection of dynamical systems with the state chi-square test, Journal of Guidance, Control, and Dynamics 17 (1994) pp. 271–277. https://doi.org/10.2514/3.21193.

[23] A. Ugoni, B.F. Walker, THE CHI SQUARE TEST: An Introduction, COMSIG Rev. 4 (1995) pp. 61–64.

[24] M. Saadeh, MATLAB based PID controller. https://www.mathworks.com/matlabcentral/fileexchange/35163-matlab-based-pid-controller (accessed 25 July 2022).

[25] M. Huba, D. Vrancic, Tuning of PID Control for the Double Integrator Plus Dead Time Model by Modified Real Dominant Pole and Performance Portrait Methods, Mathematics 10 (2022). https://doi.org/10.3390/math10060971.

List of Figures