# Detecting Malicious Behavior In Cooperative Autonomous RC Cars

G. Tombakaitė

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

# Detecting Malicious Behavior In Cooperative Autonomous RC Cars

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

G. Tombakaitė

October 20, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology


Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Detecting Malicious Behavior In Cooperative Autonomous RC Cars

by

G. Tombakaitė

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: October 20, 2020

Supervisor(s):

Dr. R. Ferrari

Ir. T. Keijzer

Reader(s):

Dr. Ing. S. Wahls

# Abstract

Sensors are used all around us in various industries, for instance: agricultural, medical, aerospace and automotive. It is important for these industries to have reliable sensor data because the functionality of technologies depends on it. In this work, the industry of interest is automotive, specifically in the field of Cooperative Adaptive Cruise Control (CACC). The control of vehicles depends on measurement data from radars, which are carried by each vehicle in a platoon. If these measurements are faulty, it could affect CACC and cause a crash. This work models the Radio Control (RC) vehicle, implements CACC and aims to identify such faults in the radar measurement data before they could impact the behavior of the platoon. The results are obtained in simulation, comprising the mathematical model of the vehicles, the implemented CACC controller and a virtual radar exposed to 4 different faults, with which the chosen method for fault detection is evaluated. The results of the CACC operating in ideal conditions and with faulty measurement data are depicted. The work ends with analysing the results and concluding, whether the chosen method is capable of identifying the faulty measurement data.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

I would like to thank my supervisors Dr. R. Ferrari and Ir. T. Keijzer for the guidance, feedback and support throughout this work. I would also like to thank my brother Tomas Tombakas, whose financial support and constant encouragement made it possible for me to study at Delft University of Technology. I would also like to acknowledge my boyfriend Gürol Gezer, who always comforted me in times of need. Finally, I would like to express gratitude to my friends Paulo, Vishrut, Gouri and Maria, for always being there for me.

Delft, University of Technology
October 20, 2020

G. Tombakaitė

# Chapter 1

# Introduction

## 1-1 Problem Statement

With the constantly growing number of vehicles, the number of roadway crashes increases as well. The current traffic demands also cause congestion in urban areas, which increases fuel consumption and hours spent on the road. To tackle these issues, transportation agencies started implementing solutions such as Intelligent Transportation Systems (ITS) [12], which use technology to improve transportation management, for example, the use of cameras to enforce the traffic laws. The ITS of interest in this thesis are Advanced Driver-Assistance Systems (ADAS), which assist the driver in parking and driving tasks, and through human-machine interaction, improve the traffic safety by reducing the time it takes to respond to unexpected obstacles and driver errors. Adaptive Cruise Control (ACC) is an ADAS that assists the driver by keeping a predefined distance from a vehicle ahead, and with the implementation of inter-vehicle communication, it becomes Cooperative Adaptive Cruise Control (CACC) [8]. CACC trusts sensors to transfer correct position and velocity data for implementing a well functioning vehicle control in traffic. Each vehicle must receive the states and input of the preceding vehicle, and send its own data to the vehicle behind. This type of control enables vehicles to drive closer to each other to reduce fuel consumption and improve traffic flow.
A lot of ADAS rely on the driver as a backup in case of failure. However, CACC aims to reduce the headaway times to 0.5 seconds or smaller, which is below the reaction time of a driver and therefore, such backup is not viable [37]. The safety of the platoon relies on the computed acceleration inputs, which are passed to the string of vehicles. These inputs are dependent on sensor data, which in CACC is provided by the radar. Introducing faults to radar measurements could result in the faulty computation of inputs to follower cars. Since vehicles in CACC travel in close proximity, sudden changes in acceleration could cause a crash. Therefore, it is important to ensure that the radar measurements are correct before using them to control a platoon. This work will aim to tackle the problem by proposing a method for fault detection in radar measurement data and testing it on 4 different simulated faults, which are injected into the sensor one by one. An experimental setup is provided to perform experiments for determining whether the chosen method is capable of detecting these faults at the time they are introduced.

## 1-2 Research Questions

Based on the problem highlighted in the previous section, the main research question becomes:

Is it possible to come up with a precise fault detection method, which identifies the faulty radar measurements before they could impact the behavior of the platoon?

The main research question can then be divided into sub-questions, giving an overview of the most important steps, which need to be taken to achieve the goal.

- How to model and identify the provided experimental setup under CACC?
- Which fault detection method has the highest potential to correctly identify faults?
- Does the chosen fault detection method identify the simulated faults on the experimental setup accurately?

## 1-3 Thesis Structure

In this thesis, the problems stated above will be tackled. Chapter 2 explains the concept of platooning and its objectives. It is followed by the introduction to string stability and the mathematical model of a vehicle. The chapter ends by presenting the model of CACC. The third chapter introduces the available experimental setup: the controller, Radio Control (RC) car and software, used for experiments. Within the same chapter, an overview of the simulation is given, which explains how does the real CACC platoon work, which parts must be simulated and which ones are simulated in this work. In chapter 4, system identification is done, which aims to identify the mathematical model of the vehicle, presented in chapter 2. Within chapter 5, the concept of a virtual radar is presented and the reason for using it. Chapter 6 explores available methods for fault detection and chooses one for the implementation. The chosen method is elaborated on and fault scenarios, which will be explored in this work, are introduced. In chapter 7, the results of the presented fault scenarios are shown and explained. The chapter ends by analysing the results and discussing how they could be improved. Lastly, this thesis ends with conclusions based on obtained results and proposes ideas for further extending this research in the future.

# Chapter 2

# Platooning

## 2-1 Introduction

With the continuous development of motorized vehicles, traffic flow increases every day, together with issues such as traffic congestion and accidents, energy waste and pollution [17]. The latest technological developments led to the integration of communication capabilities with existing Advanced Driver-Assistance Systems (ADAS), in an attempt to solve these issues. Such systems allow transmission of measurements like velocities and derivatives of it, which are used to implement longitudinal and lateral vehicle control. Longitudinal and lateral control of vehicles, which ensures they follow a lead vehicle in close proximity, is called platooning [30], an illustration of which can be seen in Figure 2-1. Research has shown, this method can increase the highway capacity [35] and reduce traffic accidents caused by human error.

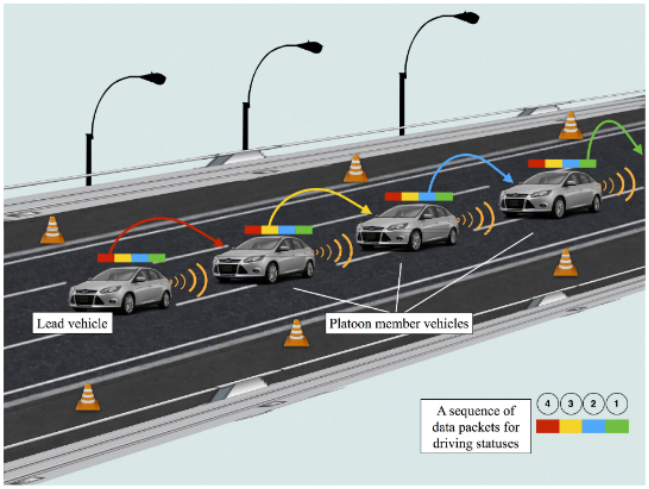



**Figure 2-1:** A platoon of vehicles [22].

In this thesis, ADAS, which enables longitudinal control, will be explored. These systems are based on headway distance and speed of a vehicle ahead. An example of ADAS is Adaptive Cruise Control (ACC), it keeps a predefined distance from a vehicle ahead, but it is not able to communicate with other vehicles. In the case where inter-vehicular communication is possible, it can be used to transmit and receive situational information, which could optimize longitudinal control and such control is known as Cooperative Adaptive Cruise Control (CACC) [30]. Vehicle platoons equipped with CACC also prove to reduce vehicular emission and travel times [23].
The design of intelligent cruise control systems must include a spacing policy design, which controls the speed of a vehicle such that it obtains a predefined distance from a vehicle ahead, and ensure string stability of the platoon [11]. The spacing policy will be part of the controller design. String stability is discussed in the next section.

## 2-2 String Stability

In platooning, it is important that vehicles are individually stable and the string of vehicles is stable as well. Individual stability simply describes vehicles converging to given trajectories [14]. For a string of vehicles equipped with intelligent cruise control systems, it is important that distance, velocity or acceleration errors do not get amplified upstream from vehicle to vehicle to ensure safety. This concept is called *string stability*, and it has multiple equivalent definitions [34]. The definition in [29] is as follows:

**Definition 2-2.1.** *A string of $m \in \mathbb{N}$ interconnected vehicles is being considered. Such a system is string stable if and only if*

$$\|z_i(t)\|_{\mathcal{L}_p} \leq \|z_{i-1}(t)\|_{\mathcal{L}_p}, \quad \forall t \geq 0, 2 \leq i \leq m \tag{2-1}$$

where $z_i(t)$ can either be position error $e_i(t)$ (defined in Eq. (2-8)), velocity $v_i(t)$ or acceleration $a_i(t)$ of the $i^{th}$ vehicle. $\|\cdot\|_{\mathcal{L}_p}$ is the $p$-norm, which is defined as $\|x\|_{\mathcal{L}_p} = (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{1/p}$, of a signal $x$. Vehicles are enumerated $i = 1, \ldots, m$, vehicle $i = 1$ being the leader. Definition of vehicle string stability therefore states that $\|z_i(t)\|_{\mathcal{L}_p}$ must decrease in the upstream direction [29].

## 2-3 Model Description

Following from the previous sections, the longitudinal dynamics of a platoon will be modeled. Just like before, vehicles are enumerated $i = 1, \ldots, m$ and vehicle $i = 1$ is the leader. In a close formation platoon, the following nonlinear third-order model of the $i$-th vehicle is considered in [33]:

$$\begin{aligned} \dot{d}_i &= v_{i-1} - v_i \\ \dot{v}_i &= a_i \\ \dot{a}_i &= f_i(v_i, a_i) + g_i(v_i)\eta_i \end{aligned} \tag{2-2}$$

where $d_i = p_{i-1} - p_i$ is the distance between two consecutive vehicles, $p_{i-1}$ and $p_i$ being their positions, $v_i$ is the velocity, $a_i$ is the acceleration and $\eta_i$ is the engine input. Functions $f_i(v_i, a_i)$ and $g_i(v_i)$ are given by

$$f_i(v_i, a_i) = -\frac{2K_{di}v_i a_i}{m_i} - \frac{1}{\tau_i(v_i)}\left[a_i + \frac{K_{di}v_i^2}{m_i} + \frac{d_{mi}}{m_i}\right] \tag{2-3}$$
$$g_i(v_i) = \frac{1}{m_i\tau_i(v_i)}$$

where $m_i$ is the vehicle mass, $\tau_i$ the time constant of its engine, $K_{di}$ is the aerodynamic drag coefficient and $d_{mi}$ is the mechanical drag. The linear model of Eq. (2-2) can be accomplished by introducing the following control law, which achieves feedback linearization

$$\eta_i = m_i u_i + K_{di}v_i^2 + d_{mi} + 2\tau_i K_{di} v_i a_i \tag{2-4}$$

where $u_i$ is the input, which is chosen such that it satisfies a required system performance criteria of the closed-loop model. Assuming that $\tau_i$ is constant, after introducing Eq. (2-4) to Eq. (2-2) the model in Eq. (2-2) becomes

$$\begin{bmatrix}\dot{p}_i(t)\\ \dot{v}_i(t)\\ \dot{a}_i(t)\end{bmatrix} = \begin{bmatrix} v_i(t)\\ a_i(t)\\ -\frac{1}{\tau}a_i(t) + \frac{1}{\tau}u_i(t)\end{bmatrix}, \quad 2 \leq i \leq m \tag{2-5}$$

The following platoon objectives in [10] define the existing platoon control approaches:

1. The closed-loop system model must be asymptotically stable, which means that every vehicle in the platoon is stable.
2. The deviation from the desired inter-vehicle distance, which is the spacing error, must be approximately zero of all vehicles.
3. The platoon must be string stable.

The platoon control approach, which satisfies these objectives will be explored in the next section.

## 2-4 Cruise Control

With the increasing number of manufactured cars in the world, highway traffic becomes a common problem. This has a negative impact on air pollution, fuel consumption and safety in traffic. New technologies such as ADAS are becoming a valuable asset for drivers to potentially increase their safety on the road. ADAS include systems such as collision-avoidance, automatic parking and lane-keeping to name a few. An ADAS system which was introduced by the autonomous industry is ACC. Without the driver's input, it automatically adjusts the speed of a vehicle to keep a safe predefined distance from a vehicle ahead [36]. Such control increases traffic safety since its reaction time is faster than that of a driver [16]. When a string of vehicles is equipped with ACC, but has no vehicle-to-vehicle communication,

the information about the leader car must be processed, control law has to be computed and the ACC equipped vehicle must respond to it. Only then, the third vehicle in the string can do the same and respond to changes in the vehicle ahead. The detection and response delay from the leader vehicle to the downstream vehicles is cumulative in the autonomous ACC. Moreover, sensor measurement errors will be amplified in this delay process and this challenges the string stability in longer streams of autonomous vehicles following each other [31].
Enabling Vehicle-to-Vehicle (V2V) communication further advances ACC. Such a system is called CACC and it ensures string stability for smaller inter-vehicle distances, which is not the case for ACC [26]. This thesis will focus on practical implementation of CACC on Radio Control (RC) cars and fault detection in the sensor data. In [29], Ploeg introduces the error

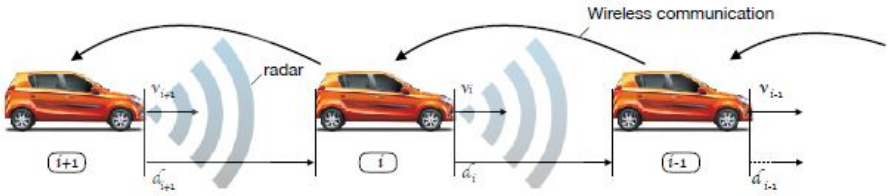



**Figure 2-2:** CACC equipped string of vehicles with V2V communication [19].

dynamics for a homogeneous string of vehicles. A string of $m$ vehicles is depicted in Figure 2-2 with $d_i$ being an inter-vehicle distance between the $i$-th and $(i-1)$-th (or preceding) vehicle, and $v_i$ being the velocity of the $i$-th vehicle. The objective of the CACC is to follow a leader vehicle at a desired distance $d_{r,i}$, which is given by

$$d_{r,i}(t) = r_i + hv_i(t) \tag{2-6}$$

where $r_i$ is a constant, which accounts for a standstill distance between the consecutive vehicles, $h$ is the headaway-time constant, measuring the time it takes for the $i$-th vehicle to arrive at the same position as the preceding vehicle when $r_i = 0$. The actual distance between the vehicles $i$ and $i-1$ is given by

$$d_i(t) = p_{i-1}(t) - p_i(t) - L_i \tag{2-7}$$

where $p_i$ is the position of the $i$-th vehicle and $L_i$ is the length of the $i$-th vehicle. Then the spacing error can be defined as follows:

$$e_i(t) = d_i(t) - d_{r,i}(t) \tag{2-8}$$

The spacing error derivatives are noted as follows:

$$\begin{bmatrix} e_{1,i}(t) \\ e_{2,i}(t) \\ e_{3,i}(t) \end{bmatrix} = \begin{bmatrix} e_i(t) \\ \dot{e}_i(t) \\ \ddot{e}_i(t) \end{bmatrix} \tag{2-9}$$

after which the control input to be fed to the follower is computed as:

$$\dot{u}_i(t) = \frac{1}{h}[-u_i(t) + (k_p e_{1,i}(t) + k_d e_{i,2}(t)) + u_{i-1}(t)] \tag{2-10}$$

As can be seen in Eq. (2-10), the control law is dependent on the sensor measurements. In an ideal situation, each car carries a radar measuring the relative position and velocity. However, in the practical setup the position measurements of each vehicle is provided by the Motion Capture (MoCap) system and the velocity is derived from them. The radar is simulated by subtracting the measured position of the follower vehicle from the position of the leader.
Introducing Eq. (2-10) into Eq. (2-9) finally yields the following closed-loop model:

$$\begin{bmatrix} \dot{e}_{1,i}(t) \\ \dot{e}_{2,i}(t) \\ \dot{e}_{3,i}(t) \\ \dot{u}_i(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\frac{k_p}{\tau} & -\frac{k_d}{\tau} & -\frac{1}{\tau} & 0 \\ \frac{k_p}{h} & \frac{k_d}{h} & 0 & -\frac{1}{h} \end{bmatrix} \begin{bmatrix} e_{1,i}(t) \\ e_{2,i}(t) \\ e_{3,i}(t) \\ u_i(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{h} \end{bmatrix} u_{i-1}(t) \tag{2-11}$$

The stability of Eq. (2-11) can be checked by verifying the state matrix is Hurwitz, which means that all eigenvalues have a negative real part. This is true as long as $h > 0$, $k_p, k_d > 0$ and $k_d > k_p\tau$ [29]. In the same paper, the proof of string stability of CACC is provided, which states that CACC is string stable if no communication delay is present. This establishes that CACC satisfies individual vehicle stability criteria and string stability.

# Chapter 3

# Experimental setup

## 3-1 Laboratory

To achieve the goals of this thesis, an experimental setup was provided by Delft University of Technology (TU Delft). The available hardware and software is presented in the next subsection. The detailed description of the software structure is presented in the subsection after.

### 3-1-1 Available Resources

The derived Cooperative Adaptive Cruise Control (CACC) model will be applied on the Erle-Rover, which is a Radio Control (RC) car, depicted in Figure 3-1a, controlled by the Erle-Brain 3 controller, depicted in Figure 3-1b. The controller is running a Linux-based


**(a)** Erle-Rover.

**(b)** Erle-Brain 3.

**Figure 3-1:** Hardware.

operating system, which includes an open source autopilot software called the ArduPilot and an open source Robot Operating System (ROS), which does not replace, but rather works

alongside a traditional operating system [27]. ArduPilot and ROS are combined because ROS extends the capabilities of the ArduPilot. The autopilot allows the user to configure certain parameters and gains, however editing the software with custom functions is difficult without a clear knowledge of the software structure of the ArduPilot. ROS provides an offboard control option with various libraries, tools and device drivers, which simplify the process of creating new applications for robots.
The feature description of the Erle-Rover is provided in Table 3-1. The rover includes: battery, servo motor, power module, a motor with an Electronic Speed Controller (ESC) and Pulse Position Modulation (PPM) sum module [1] as can be seen in Figure 3-2.

| Feature | Description |
|---|---|
| Dimensions | 53X19.3X29.7 cm |
| Wheel base | 33.4 cm |
| Diameter of the wheels | 112 mm |
| Width of the wheel | 45 mm |
| Drive System | 2WD Rear |
| Gearing | 48dp |

**Table 3-1:** Features description of the Erle-Rover [1].

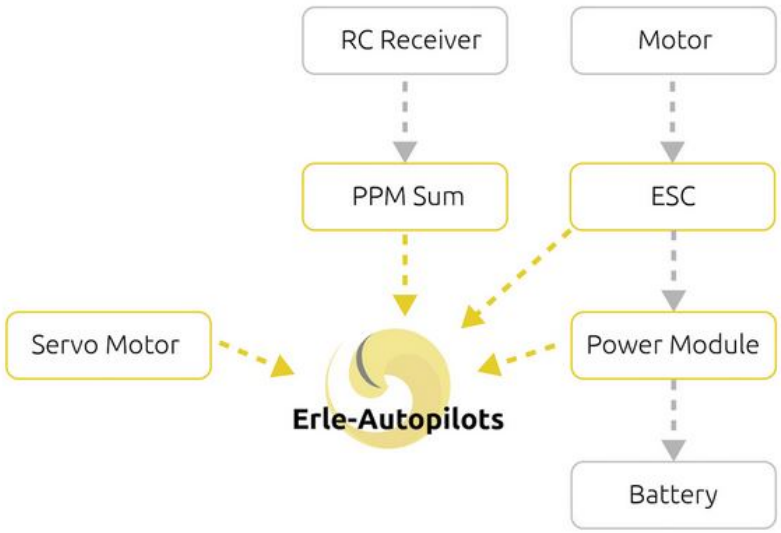



**Figure 3-2:** Connections of the Erle-Rover components [1].

The experiments will be performed in the Networked Embedded Robotics DCSC lab (NERDlab) using the OptiTrack Motion Capture (MoCap) system, which is a robot tracking system used for aerial and ground vehicles. Positional error of the MoCap system is less than 0.3 mm and rotational error is less than 0.05°, provided it is operating in ideal conditions [25].

The Erle-Rover together with the Erle-Brain are placed in the arena, as shown is Figure 3-3. The MoCap system sends the coordinates of the Erle-Rover to the computer, connected to the MoCap system. This data is then transferred to a Linux running computer, which is connected to the same WiFi network as the MoCap computer. The computer, running Linux, is also running ROS and Matlab, which are used to control the Erle-Rover.

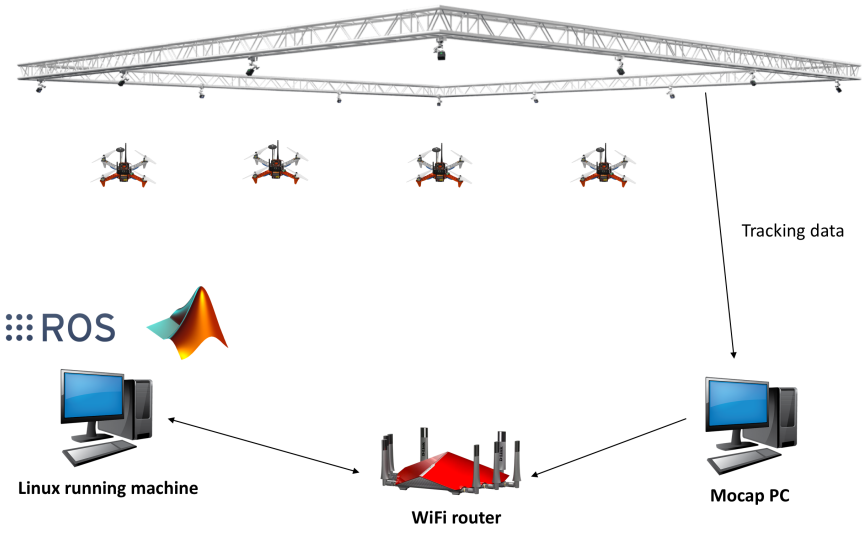



**Figure 3-3:** Experimental setup [2].

### 3-1-2 Software Structure

The Erle-Brain 3, also called the companion computer, is composed of a micro computer Raspberry Pi (RPi) 3 and an autopilot board mounted on top of RPi, made by the Erle Robotics. As was mentioned before, the Erle-Rover will be controlled by a personal computer, which will further be referred to as the Ground Station (GS). The Erle-Brain 3 and GS are both connected to the same WiFi network called *DCSC_Robot*, GS is running ROS *master*, the function of which is to enable communication between various ROS programmes, and a program which is responsible for the Erle-Rover control. The companion computer (Erle-Brain 3) of a vehicle is running a combination of ArduPilot and ROS. The communication protocol that ArduPilot is using is called Micro Air Vehicle Link (MAVLink) and ROS is required to have MAVLink libraries installed to allow communication between itself and the autopilot.
In Figure 3-4, the software architecture of the experimental setup is depicted. It can be seen that GS communicates to the vehicle using WiFi, those commands are received by the companion computer and passed onto the Autopilot, which ensure the rover responds to the newly sent instructions.

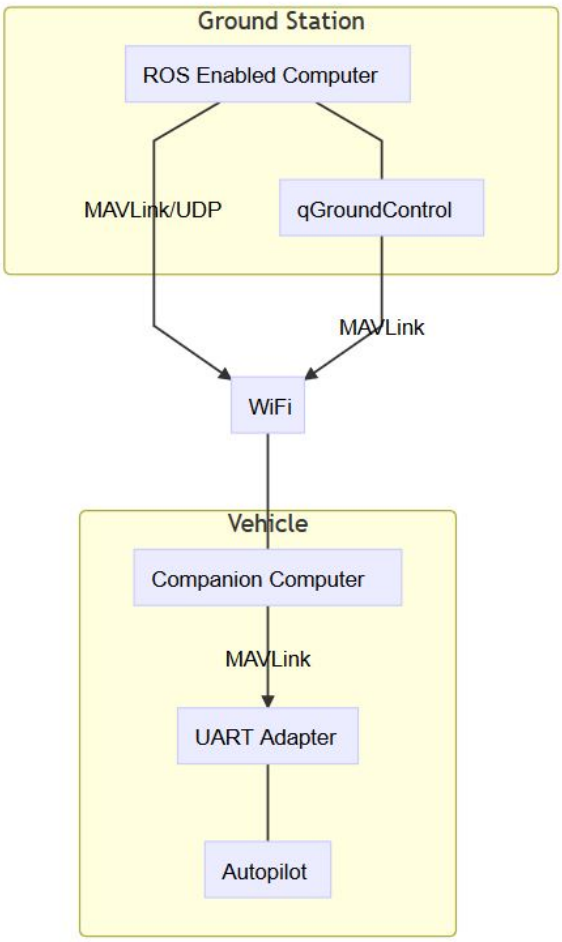



**Figure 3-4:** Software architecture of the experimental setup [3].

## 3-2 Simulation

The platoon, described in Section 2-4, virtual radar and radar sensor faults will be simulated in Simulink. The simulation schematics can be found in Figure 3-5, in which passed parameters from one block to another are depicted next to the arrows. The symbols have the following meaning: $p_L$, $v_L$, $a_L$ are position, velocity, acceleration of the leader car, $p_F$, $v_F$, $a_L$ are position, velocity, acceleration of the follower car, $u_L$ and $u_F$ correspond to inputs of leader and follower cars respectively, $F$ is the introduced fault to the measurement data, $e_1, e_2$ are relative position and velocity errors respectively. It can be noted that leader and follower cars are not shown to transmit any data to the MoCap. It is so because cars do not directly communicate any data, instead, MoCap tracks the reflective markers on the Erle-Rover and determines the coordinates of the vehicle.
In the beginning of the simulation an acceleration input is given to the leader car, where both rovers, leader and follower, are modeled after Eq. (2-5). In Figure 3-5, inside the green block, experimental setup in the NERDlab is simulated. Due to COVID-19 the NERDlab became inaccessible and the experimental setup had to be simulated in Simulink. In the

NERDlab, the MoCap system is constantly monitoring vehicles and sending their $x$, $y$, $z$ coordinates together with timestamps to the GS. The MoCap data is noisy and to mimic it, white Gaussian noise is added to the simulated measurement data so it would correspond to the measurement data received in the NERDlab. Within the orange block, virtual radar and sensor faults are simulated. The goal of the virtual radar is to mimic the radar in a real follower car in CACC platoon. Part of the virtual radar block is the MoCap Kalman Filter (KF), which filters out noisy MoCap data. Then, the radar measurements are simplified to the difference in position, $p_{leader} - p_{follower}$, and velocity, $v_{leader} - v_{follower}$. Within the same orange block, fault and switch blocks are presented as well. The fault to the radar data is introduced via a switch. If no fault is present, measurements of the simulated radar are directly passed to the CACC block. However, if the fault is activated, the switch is activated as well and one of the four presented fault scenarios are passed to the CACC block. The blue block represents the components that would be present in the real CACC platoon. Within the purple block in Figure 3-5, the input to the follower vehicle is computed and fault detection is done. CACC block passes the computed error dynamics to the fault detection block, where the data is assessed and then used to compute the control law. The input is computed to the follower car independently of whether the fault is introduced or not. The purple block represents the actions, which are always simulated. In a real CACC, it would be running on an onboard computer.

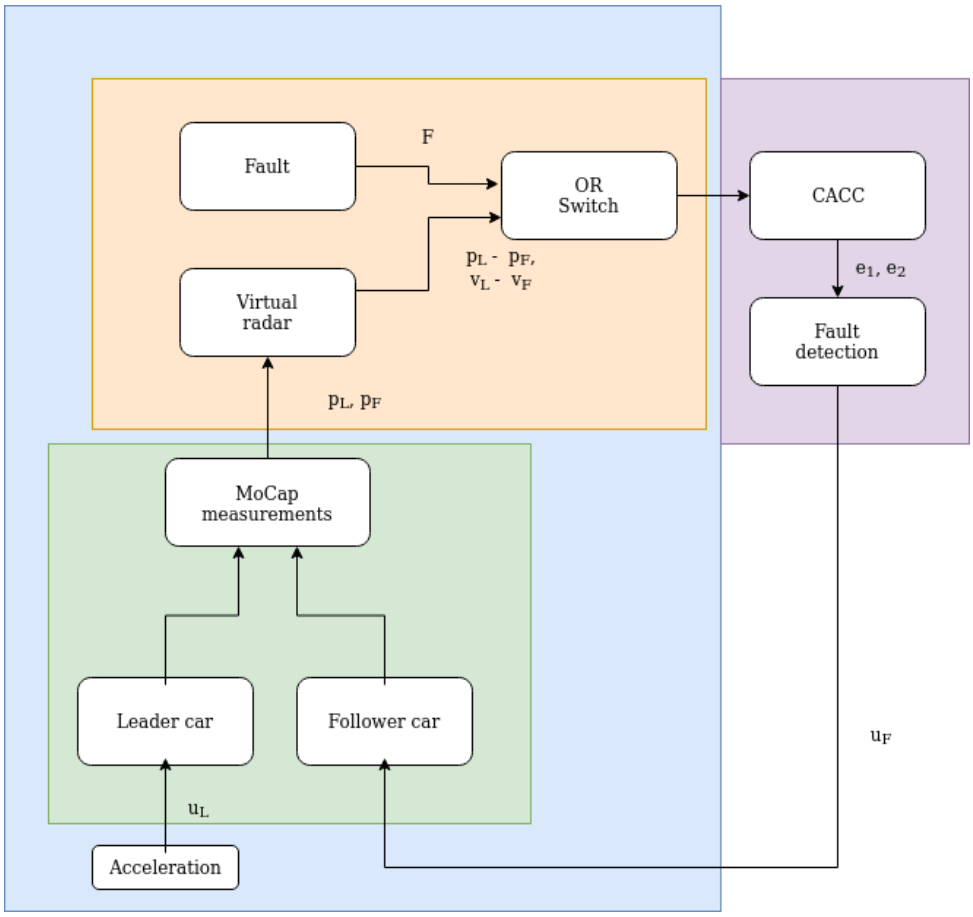



**Figure 3-5:** Simulation diagram.

# Chapter 4

# System identification

## 4-1 Goal

The goal of this section is to identify the mathematical model of the Erle-Rover. The vehicle model was introduced in Eq. (2-5). It can be seen that the model depends on position $p$, velocity $v$, acceleration $a$, time delay $\tau$ and input $u$. The position of the Erle-Rover can be measured and velocity and acceleration can be derived from it; the input to the vehicle must be desired acceleration and the time delay must be identified. This poses two challenges:

1. Identifying inputs to the Erle-Rover.
2. Correctly identifying time delay $\tau$ based on the measurements and input.

## 4-2 Practical Issues

In order to perform system identification, cars must be able to drive on a predefined path and their input must be known. Since there are no longer any manuals or example codes provided by Erle Robotics, new code had to be written from scratch.
The first step towards controlling the vehicle is connecting to the controller: Erle-Brain 3 and the Ground Station (GS) computer must be connected to the same WiFi network and their Internet Protocol (IP) addresses must be known; Then, the GS can connect to the controller using the Secure Shell (SSH) by running `ssh erle@erle-brain-3.local`. The IP addresses are used to initialize Robot Operating System (ROS) master and ROS slave in the *hosts* and *.bashrc* files. Once the files are updated, ROS master is initialized by running the `roscore` command on GS. The Erle-Brain is connected to the master by running the *setup-mavros.bash* file on the controller.
After connecting to the controller from the GS, available *rostopics* were explored. ROS *nodes* (programs) communicate with each other by sending messages, which are organized in topics. A node that shares information, publishes on a specific topic; one that receives

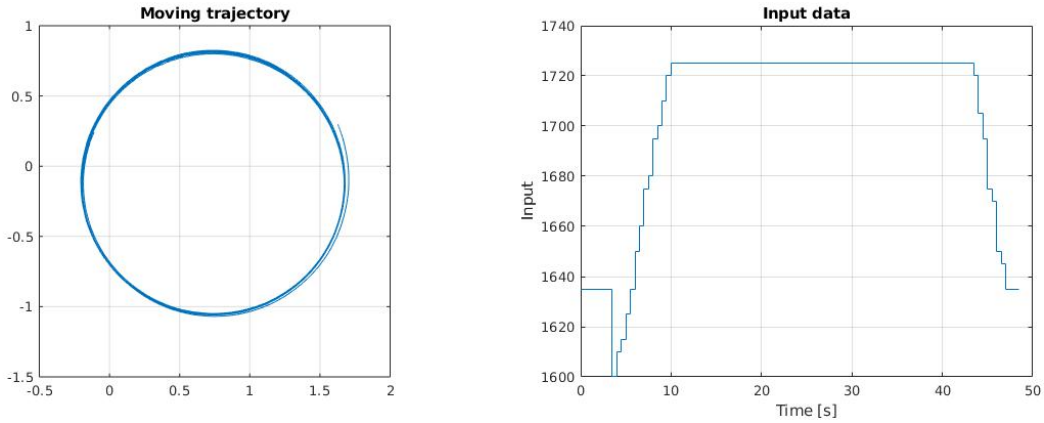
information, subscribes to a specific topic. The list of the Erle-Brain 3 rostopics can be found in Appendix A-2. It was found that the only topic to which the Erle-Rover responds to is `/mavros/rc/override`, which is the velocity input to the rover. As the name suggests, it overrides the current `rc/in` values. The topic has 8 inputs in total that correspond to 8 channels. The Erle-Brain 3 has several configurations, which include copter, rover and boat, and each of these configurations requires different numbers of servos and motors. The channels to which these devices are connected must be overwritten with desired inputs. In the rover configuration, channel 1 is responsible for controlling the servo motor, which does the steering, and channel 3 controls the brushless motor, which is the throttle of the rover. The custom node written for the rover control can be found in the appendix A-1.
The input to the Radio Control (RC) car is velocity, which is published by the ROS topic `mavros/rc/override`. The values the message sends are within the range 1000-2000, which hold no immediate physical meaning and therefore the velocity mapping to $m/s$ needs to be computed. In order to identify the model in Eq. (2-5), the velocity input to the Erle-Rover must be approximated as the acceleration input. The approximation will be assumed to be the following:

$$u = \frac{v_{ref} - v}{\tilde{t}} \tag{4-1}$$

where $v_{ref}$ is the input velocity, $v$ is the measured velocity of the rover and $\tilde{t}$ is the time in which the desired velocity should be achieved.
For the implementation on the experimental setup, the input must be integrated, which results in velocity in $m/s$. Then, these values must be mapped to Erle-Rover values and only then the inputs to the RC cars can be sent.

## 4-3 Velocity Mapping

In order to identify inputs to the Erle-Rover, the Motion Capture (MoCap) system OptiTrack in the Networked Embedded Robotics DCSC lab (NERDlab) was used together with ROS running on a GS computer. A custom ROS node *rover_control* was programmed to drive the Erle-Rover car approximately in a circle with a 1 meter radius, as can be seen in Figure 4-1a. In order to collect the data, *mocap_optitrack* node must be running at the same time, collecting the position data from the OptiTrack and saving it on the GS computer.
Experiments were conducted only for inputs 1560-1770, because of the lack of space in the NERDlab: extremely high speeds are hard to control and the Erle-Rover can easily end up crashing into an obstacle damaging the car. The input to the system is a staircase input with steps of 5 or 10, as can be seen in Figure 4-1b. These amplitudes were chosen because the Erle-Rover fails to respond to higher amplitude steps. If the initial input to the Erle-Rover goes from 1600 to 1750 in one step, the Erle-Rover will stand still in its initial position. Step of 10 proved to be the largest safe step the Erle-Rover would never fail to respond to.

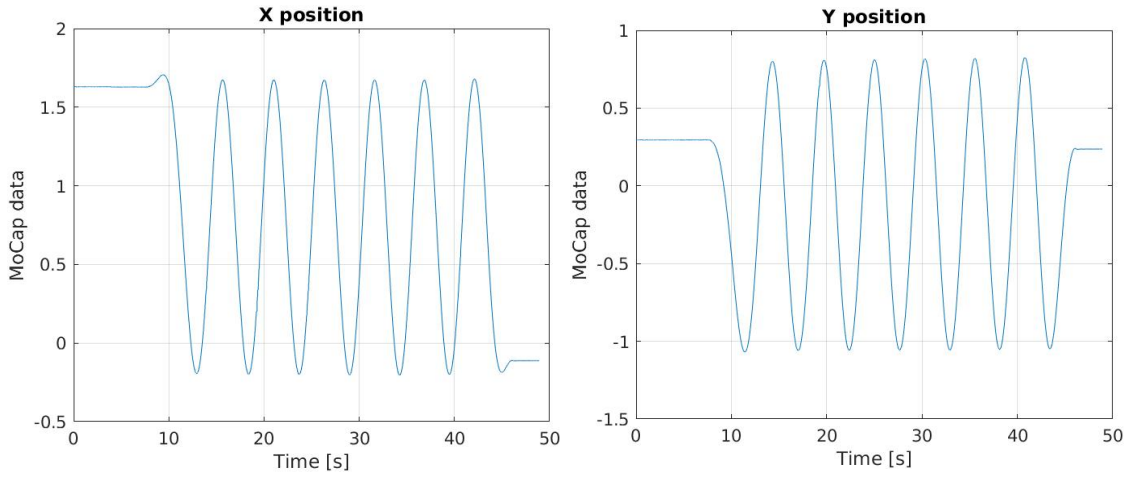



**(a)** Erle-Rover moving trajectory

**(b)** Input to the Erle-Rover.

**Figure 4-1:** Input and output of the system

The positional error of OptiTrack is less than 0.3 $mm$ and rotational error is less than 0.5° [25]. However, this is only true provided that it is being used in ideal conditions, which include: minimal ambient light (sunlight or other infrared source), all obstacles are removed from the capture area and reflective objects are covered [6]. The last two conditions were satisfied, but because of sunlight in the NERDlab, a slight error is visible in the collected OptiTrack data. In Figure 4-2, raw $X$ and $Y$ position data can be seen. Then, the time derivatives were taken to calculate velocities based on $X$ and $Y$ position data, Figure 4-3. Finally, the Euclidean norm is taken of each velocity component and the total car speed is computed, Figure 4-4. In total 19 experiments were performed with an initial input value of 1600 climbing up to 1770 and back down to 1560. Velocities are computed for each value and then, by taking the mean value of the computed velocity after it reaches its peak value and before the car starts braking, the velocity mapping is computed, Figure 4-5.

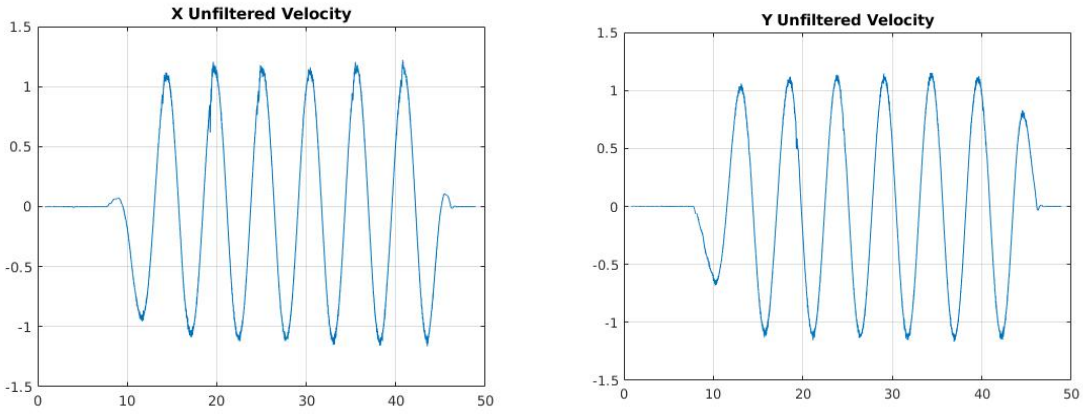



**(a)** X position data.

**(b)** Y position data.

**Figure 4-2:** Raw position data.

For the implementation on the experimental setup, a velocity look-up table is made so that




**(a)** X velocity.

**(b)** Y velocity.

**Figure 4-3:** Velocity based on the time derivatives.

the controller could translate the received velocity values in $m/s$ to the Erle-Rover expected values.

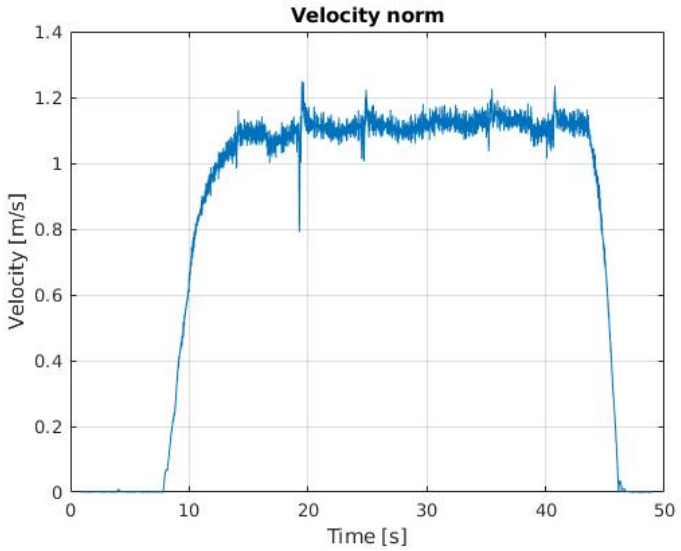



**Figure 4-4:** Velocity norm.

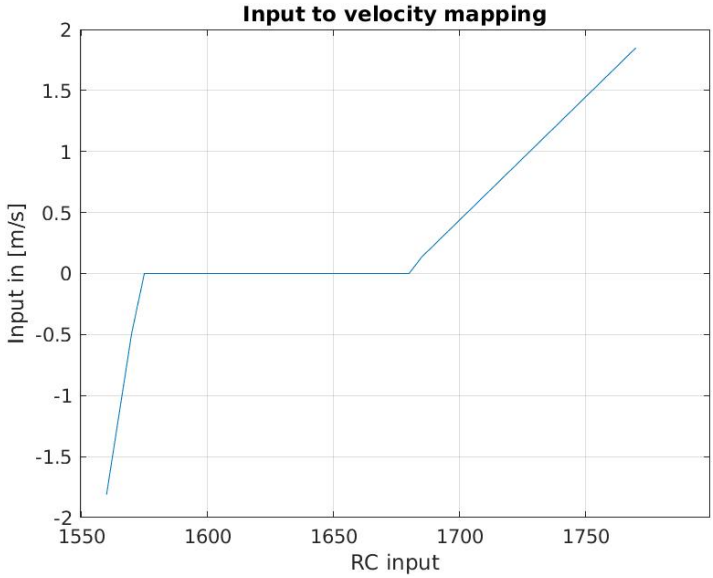



**Figure 4-5:** Velocity mapping.

## 4-4 Simulation Input and Output Practical Considerations

The leader and follower cars are initialized at different positions. The input to the leader car, according to Eq. (2-5), must be acceleration, which is depicted in Figure 4-6. However, the

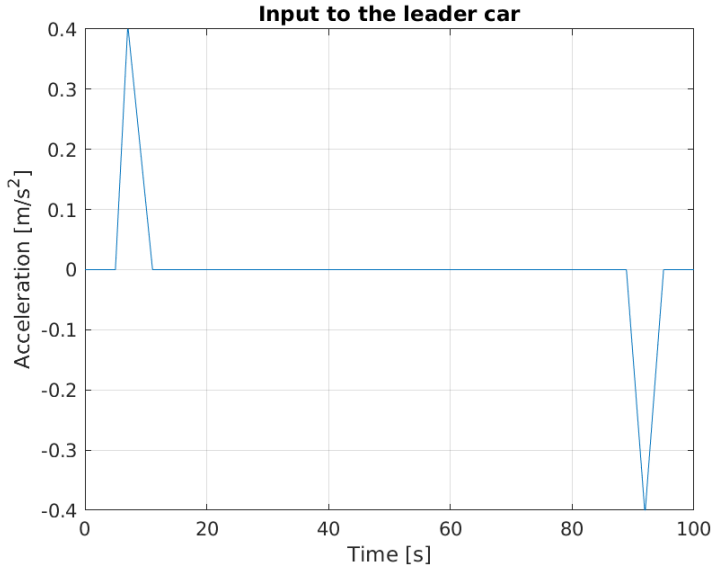
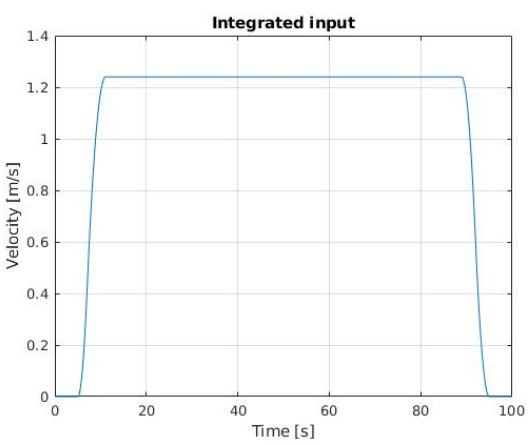
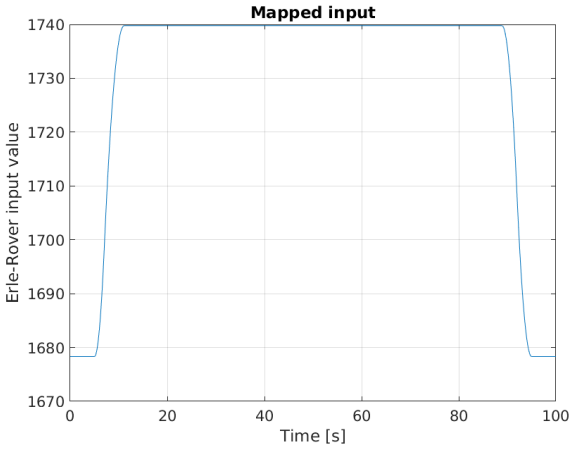



**Figure 4-6:** Input to the leader car.

Erle-Rover only accepts velocity inputs, hence, for the implementation on the experimental setup, the acceleration input must be integrated and mapped to the Erle-Rover values depicted in Figure 4-5.




**(a)** Integrated acceleration input.




**(b)** Mapped input to the Erle-Rover values.

**Figure 4-7:** Input to the Erle-Rover.

The only data received from the experimental setup is position data from the MoCap system in $x$, $y$, $z$ coordinates. The simulation is meant to replicate the available hardware and data at the laboratory and therefore, only position data and its derivatives will be used for control. Since in this thesis only one dimensional dynamics will be explored, $z$ data will be neglected and $x$ and $y$ data will be converted into the total travelled distance

$$d = \sqrt{(x(i) - x(i-1))^2 + (y(i) - y(i-1))^2} \tag{4-2}$$

Cooperative Adaptive Cruise Control (CACC) outputs an acceleration input to the follower car based on the error dynamics and tuned $PD$ gains used in the control law, Eq. (2-10). In order to implement such control on Erle-Rovers, the input must be integrated and converted to the Erle-Rover velocity values. That is achieved by using the look-up table, which contains the Erle-Rover values and measured $m/s$ values. As can be seen in Figure 4-9, $1.2m/s$ is equivalent to approximately 1740.

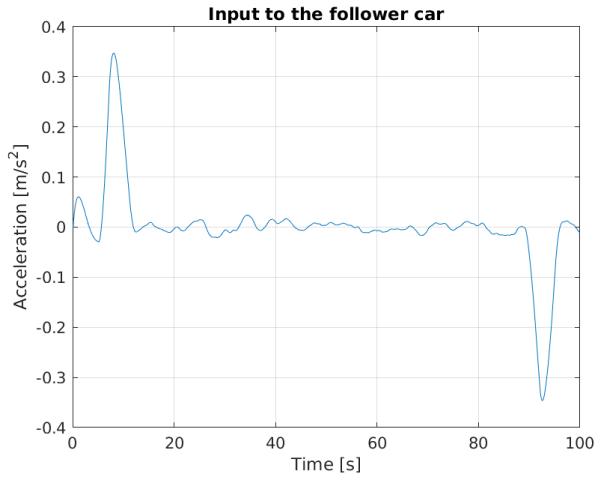
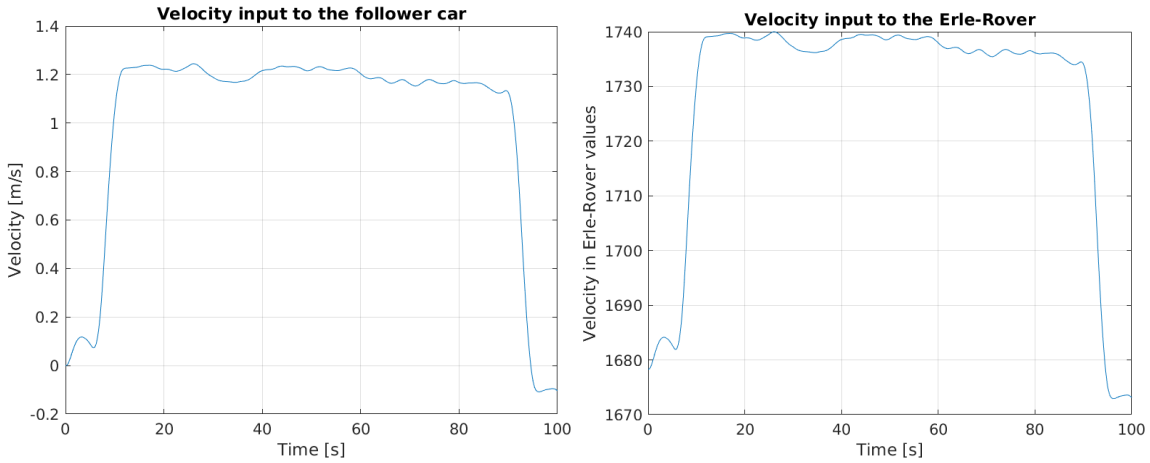



**Figure 4-8:** Input to the follower car.




**(a)** Integrated input to the follower car. **(b)** Mapped input to the Erle-Rover values.

**Figure 4-9:** Input to the Erle-Rover.

## 4-5 Identifying $\tau$

After having the velocity mapping calculated, the time constant $\tau$, which accounts for dynamics of an engine, can be identified, Eq. (2-5). As was mentioned before, the input to the Erle-Rover must be approximated, since in Eq. (2-5) it is the desired acceleration, but the Erle-Rover expects a velocity input. The assumption can be seen in Eq. (4-1), with which the system model becomes

$$\begin{bmatrix} \dot{p} \\ \dot{v} \\ \dot{a} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -\frac{1}{\tau\tilde{t}} & -\frac{1}{\tau} \end{bmatrix} \begin{bmatrix} p \\ v \\ a \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{\tau\tilde{t}} \end{bmatrix} v_{ref} \tag{4-3}$$

and its transfer function

$$y = \frac{1}{s(\tau\tilde{t}s^2 + \tilde{t}s + 1)} v_{ref} \tag{4-4}$$

Now, the system could be identified based on the collected position data from the MoCap system and the predicted output in Eq. (4-4). MATLAB is used to write an optimization function which compares the predicted system output with the measurement data and reduces the error by optimizing the parameters $\tau$ and $\tilde{t}$. The experiments conducted in Subsection 4-3 were saved in different files. The algorithm aims to minimize the error between the measurement and the estimate, as can be seen in Eq. (4-5).

$$\min_x \|f(x))\|_2^2 \tag{4-5}$$

where $x = y_{measured} - y_{predicted}$. The error is minimized by iterating over the optimization parameters $\tau$ and $\tilde{t}$. The implementation of the cost function included a simulation of the system, and was minimized with `lsqnonlin`. The simplified algorithm can be seen in Algorithm 1.

**Algorithm 1** Simplified system identification algorithm

1: **initialization:** $\tau = 0.3$, $\tilde{t} = 1$;
2: yMeasured = LOADALLDATA(experiments)
3: **function** APPROXOVERALLDATA($\tau$, $\tilde{t}$, data)
4: err = [yMeasured - yPredicted]
5: **end function**
6: $[\tau, \tilde{t}]$ = LSQNONLIN(err)

The results, achieved by following the steps mentioned above can be seen in Figure 4-10 to Figure 4-13, which depict the unfiltered MoCap measured data in blue and the estimate, based on the model derived in Eq. (4-4). When the rover is driving at a low speed, Figure 4-10, the measurement data is clear and the estimation error is relatively small, Figure 4-14a. However, at higher speeds the rover starts drifting and the inconsistencies in the measurement data are visible in Figure 4-11, Figure 4-13 and especially in Figure 4-12.

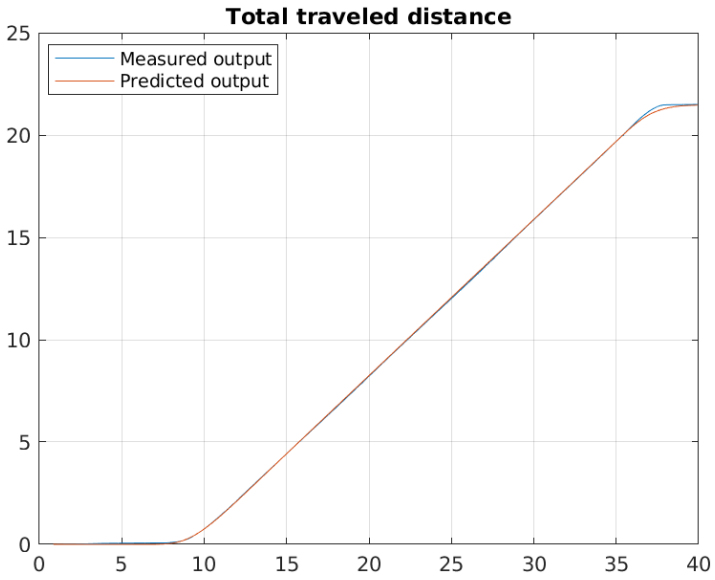



**Figure 4-10:** Input value of 1720.

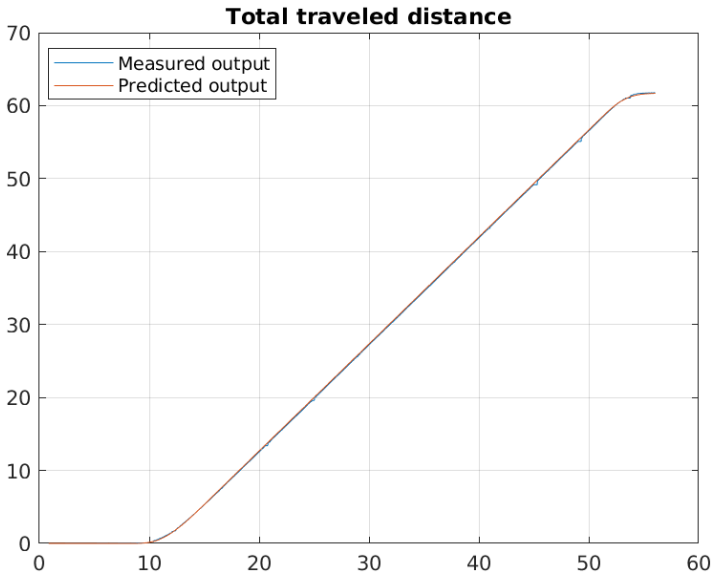



**Figure 4-11:** Input value of 1750.

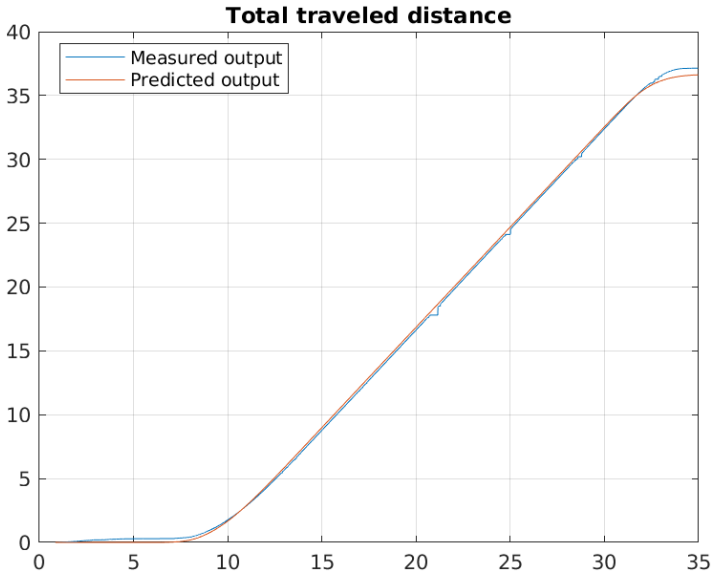



**Figure 4-12:** Input value of 1760.

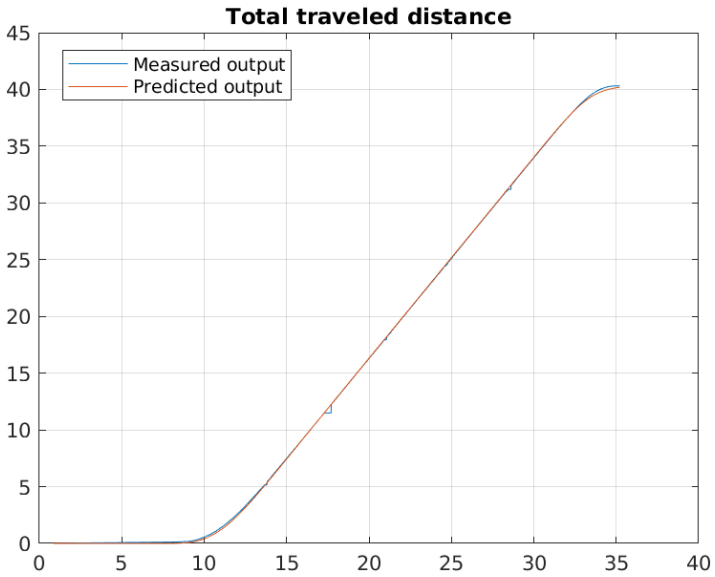
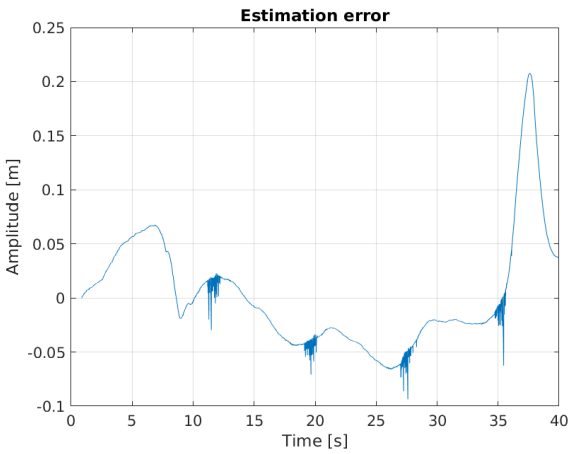
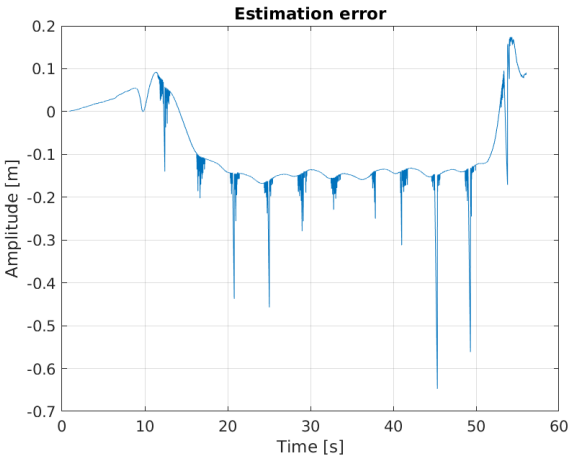



**Figure 4-13:** Input value of 1770.

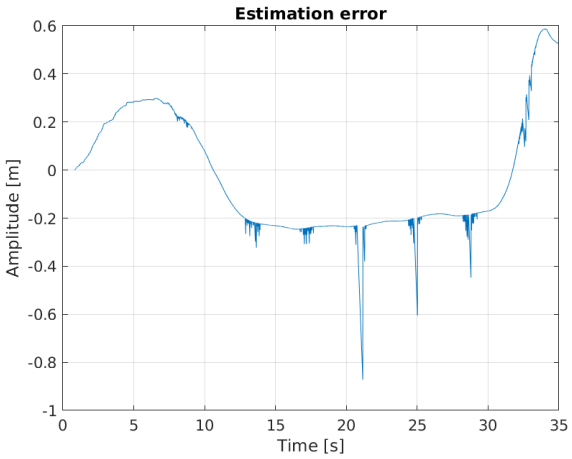
The predicted and measured traveled distance data match with a small error, values of which vary in the range of 0.005-0.2 meters. The noise in Figure 4-14 is due to the measurement noise of the MoCap data. The identified value of the time constant is $\tau = 0.61$. After the identification of the mathematical model of the Erle-rover, the access to the NERDlab was restricted and therefore, the rest of the work was completed in simulation.



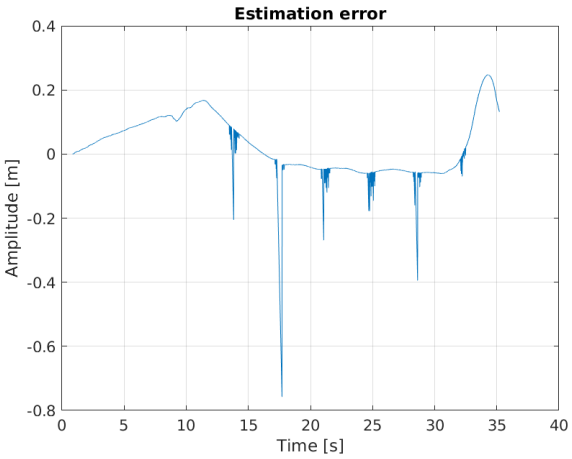

**(a)** Estimation error when the input is equal to 1720.




**(b)** Estimation error when the input is equal to 1750.




**(c)** Estimation error when the input is equal to 1760.




**(d)** Input value of 1770.

**Figure 4-14:** Estimation error when the input is equal to 1770.

# Chapter 5

# Virtual Radar

## 5-1 Introduction

Cooperative Adaptive Cruise Control (CACC) uses inter-vehicle distance, relative velocity and inter-vehicle communication to ensure the platoon of vehicles keeps a predefined inter-vehicle distance. In the real platoon, each vehicle would be equipped with a radar, which provides the required measurement data. However, the experimental setup does not include a radar and therefore it must be simulated. In the Networked Embedded Robotics DCSC lab (NERDlab) the position of each vehicle is measured, but not the inter-vehicle distance, and therefore it must be derived from the provided Motion Capture (MoCap) data. Within the simulation, the virtual radar must be simulated as well because the simulation aims to mimic the provided experimental setup. Thus, the MoCap measurements will be replicated and radar measurements will be derived from them, just like in the NERDlab.
In this thesis, the radar consists of the Kalman Filter (KF) and the radar itself. The KF is there to filter out the noisy MoCap data and to derive velocity and acceleration from the provided position data. The radar is simplified to the difference in position of the follower and leader vehicles.

## 5-2 State Estimation

Virtual radar measurements are used to compute the position, velocity and acceleration errors depicted in Eq. (2-11). These errors are needed to compute the control law in Eq. (2-10). The errors must be sufficiently accurate because the input to the follower car is based on them. The only data provided by the MoCap system is position and these measurements are noisy, velocity and acceleration must be estimated using the provided position data.
Given the stochastic nature of the measurement noise affecting the MoCap data assumed to be Gaussian white noise, the KF is a particularly apt choice for state estimation.

### 5-2-1 Kalman Filter

The KF was proposed by R.E. Kalman in the 1960s [18]. The goal of the KF is to find a minimum variance unbiased estimate of the state vector. A Linear Time-Invariant (LTI) model subject to two noise signals $w(k)$ and $v(k)$ will be considered:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) + w(k) \\ y(k) &= Cx(k) + v(k) \end{aligned} \tag{5-1}$$

where $w(k)$ and $v(k)$ are zero-mean Gaussian white process and measurement noise signals, respectively and their joint covariance matrix is

$$E\begin{bmatrix} w(k) \\ v(k) \end{bmatrix}\begin{bmatrix} w(j)^T & v(j)^T \end{bmatrix} = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix}\Delta(k-j) \tag{5-2}$$

such that

$$\begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \geq 0, \quad \text{and} \quad R > 0 \tag{5-3}$$

Due to the noise present in the measurements and process, the error between the estimate and the real state will not converge to zero. Instead, the KF minimizes the filtered and predicted state error covariance matrices and ensures the mean of the errors converges to zero. The KF comprises a 2 step process: prediction and correction. In the prediction step, it uses the system model to calculate an estimate of the state and error covariance. If the pair $(A, C)$ is observable and $(A, Q^{1/2})$ is reachable then at a time step $k-1$ an estimate of $x(k)$, denoted $\hat{x}(k|k-1)$, can be calculated, and has the following properties:

$$\begin{aligned} &P(k|k-1) = E[(x(k) - \hat{x}(k|k-1))((x(k) - \hat{x}(k|k-1))^T], \\ &\text{with } \hat{x}(k|k-1) = E[x(k)], \text{ satisfies} \\ &\lim_{k\to\infty} P(k|k-1) = P > 0 \end{aligned} \tag{5-4}$$

In Eq. (5-4) the estimate is uncorrelated with the noises $w(k)$ and $v(k)$ and $P$ satisfies the following discrete algebraic Riccati equation

$$P = APA^T + Q - (S + APC^T)(CPC^T + R)^{-1}(S + APC^T)^T \tag{5-5}$$

Eq. (5-5) has several solutions however only the positive-definite solution is unique and ensures the asymptotic stability of $A - KC$. The calculated $P$ is used to determine the Kalman-gain matrix $K$

$$K = (S + APC^T)(CPC^T + R)^{-1} \tag{5-6}$$

Once the estimate and the error covariance matrix are calculated, the KF uses the input $u(k)$ and measured output $y(k)$ data for the one-step-ahead prediction

$$\begin{aligned} \hat{x}(k+1) &= A\hat{x}(k) + Bu(k) + K(y(k) - C\hat{x}(k)) \\ &= (A - KC)\hat{x}(k) + Bu(k) + Ky(k) \\ \hat{y}(k) &= C\hat{x}(k) \end{aligned} \tag{5-7}$$

In Eq. (5-7) the term $i(k) = y(k) - C\hat{x}(k)$ is known as *innovation sequence* or *measurement residual* and covariance matrix of it is

$$E[i(k)i(k)^T] = CPC^T + R \tag{5-8}$$

The update of a state at a particular time instant is done at the same instant at which the input-output measurements are collected [38].

### 5-2-2 Kalman Filter Design

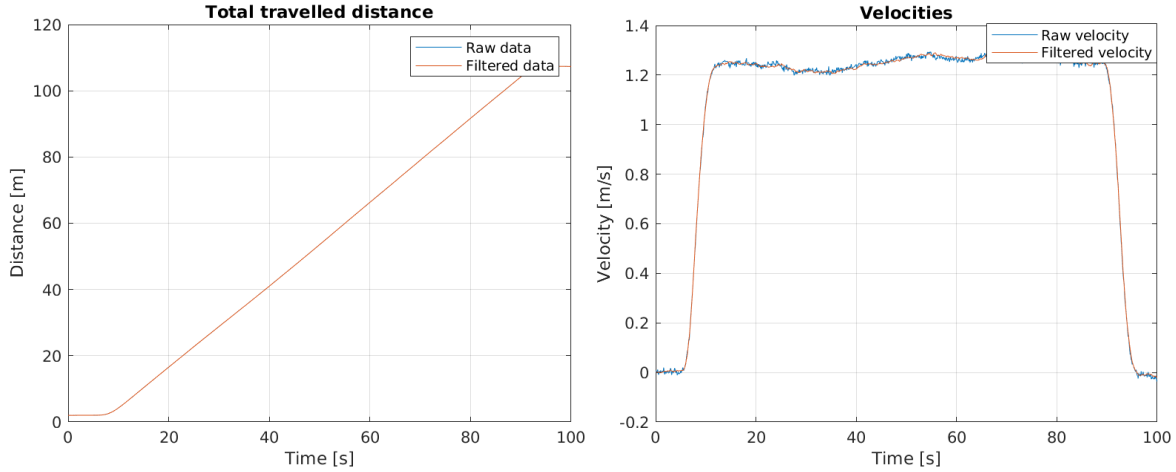
Before the KF can be designed Eq. (2-5) must be discretized. The discrete time-step $k$ is defined such that

$$T = kD \tag{5-9}$$

where $D$ is the sampling period and $T$ is the elapsed time of simulation. Usually, the plant is controlled by holding each control input for one sample interval. This method is known as Zero-Order Hold (ZOH). The discrete plant matrices can be computed as follows, according to [21]:

$$\phi = e^{AD} = I + AD + \frac{A^2D^2}{2!} + \frac{A^3D^3}{3!} + \dots \tag{5-10}$$
$$\Gamma = \int_0^D e^{As}Bds$$

Model Eq. (2-5) is discretized using the ZOH method and the KF gain can be easily designed for the derived model assuming knowledge of the noise and process covariances, which were manually added for simulation purposes, following the procedure of Section 5-2. The KF results can be seen in Figure 5-1. The MoCap KF is fed the acceleration inputs of the leader and follower vehicles, and the simulated position data of both vehicles, which is corrupted with process and measurement noise. The noise covariances are chosen such that it replicated the received data from the MoCap. Once the MoCap KF retrieves the filtered position data,




**(a)** Position measurements and estimation. **(b)** Velocities measurements and estimation.

**Figure 5-1:** Simulated MoCap data.

velocity and acceleration, the difference in position and velocity of the two vehicles is taken to simulate a simplified radar.

# Chapter 6

# Model-Based Fault Detection

## 6-1 Fault detection methods

With increasing development in automated vehicles, the requirements for safety increase as well. Faulty sensor measurement data could cause a traffic accident and therefore it is important to detect such faults on time. Communication networks and on-board computers make it possible to run various fault detection algorithms, which are categorised as follows according to [24]:

- Signal-based fault detection.
- Model-based fault detection.
- Knowledge-based fault detection.

Signal-based methods involve the limit checking method and trend checking method, which require upper and lower limits for a signal. If it exceeds the determined thresholds, the data is considered to be compromised. An advantage of this method is its simplicity, however the disadvantage is the fact that this approach is only able to react after a relatively large change of a feature [15]. Another signal-based fault detection method is spectrum analysis, which looks at amplitudes or amplitude densities within a certain bandwidth of a signal. When no fault is present the signal is expected to fall within a particular spectral range [24].
Model-based fault detection method is based on the deviation between the measured data and model estimated data. This method is classified into three categories: observer-based, parity space and parameter estimation. For the observer-based approach the system variables are estimated from the measurement data with the aid of an observer. For linear systems, a Luenberger observer can be used, but in stochastic cases, the Kalman Filter (KF) provides statistically optimal estimates of the system states. Measurement prediction errors are used as a residual for fault detection [4]. Parity space fault detection also uses a mathematical model of a system, but unlike the observer-based method, the error is not fed back to the

system. Parity space is an open-loop model. Fault detection is based on the difference between the measured data and model estimated data [13]. Residuals and parity vectors are synonyms in this case. The parity space constructs the parity vectors and analyses them [28]. Parameter estimation approach compares the estimated parameters of the model under nominal conditions and the ones estimated at each new time step. The fault is detected based on the difference between the estimated parameters [9].
Knowledge-based fault detection is used when understanding physical properties of a system is not possible or is too costly to obtain. It requires experienced engineers to come up with rules that define the action that must be taken when a specific symptom is observed [5].
In this work, the model-based approach for fault detection is selected because the mathematical model of the Erle-Rover was successfully identified in the previous chapter, which gives a lot of insight about the system behavior under nominal conditions. Since the Motion Capture (MoCap) data has measurement noise, the KF was chosen for state estimation to filter it out, which is further described in the next section. The fault detection method is based on the innovation sequence, which is further explained in the last section of this chapter.

## 6-2 Kalman Filter for Fault Detection

Fault detection will be based on the error dynamics, defined in Eq. (2-11). The virtual radar measures inter-vehicle distance and relative velocity. The measurements are used to calculate the error dynamics, which are passed to the fault detection KF. Because the noise to the position measurements is injected manually, the covariances of the error dynamics can be derived and an accurate KF can be designed. The innovation sequence is then used to determine whether the fault is present in the radar measurements.

### 6-2-1 Kalman Filter Design

The error dynamics of the controlled system in Eq. (2-11) has the PD controller included in the state matrix, which must be separated before the KF can be designed. System Eq. (2-11), after removal of the input update equation and separation of the control law, becomes:

$$\begin{bmatrix} \dot{e}_{1,i}(t) \\ \dot{e}_{2,i}(t) \\ \dot{e}_{3,i}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{\tau} \end{bmatrix} \begin{bmatrix} e_{1,i}(t) \\ e_{2,i}(t) \\ e_{3,i}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{\tau} \end{bmatrix} \begin{bmatrix} k_p & k_d & 0 \end{bmatrix} \begin{bmatrix} e_{1,i}(t) \\ e_{2,i}(t) \\ e_{3,i}(t) \end{bmatrix} \tag{6-1}$$

As can be seen in Eq. (6-1), the same state matrix is presented in Eq. (2-5) and therefore the discretized matrix $A$ will be the same as for the MoCap KF gain design.
The process and noise covariances of the noise signals that reach the fault detection KF must be derived. The simulated radar measurement data is the difference between the MoCap KF filtered states of the two vehicles

$$(\hat{x}_l - \hat{x}_f)(k) = (x_l - x_f)(k) + \nu \tag{6-2}$$

where $x_l$ is the state vector of the leader car and $x_f$ is that of the follower, $\nu$ is the measurement noises of both vehicle. However, the mentioned measurement noise is already taken care of

by the MoCap KF, which is used for filtering out the MoCap data. As defined in [38] the covariance matrix $P$, in steady-state is

$$E[(x(k)-\hat{x}(k))(x(k)-\hat{x}(k))^T] = P \tag{6-3}$$

Knowing that the states of the MoCap KF are position $p$, velocity $v$ and acceleration $a$, the covariance matrix $P$ corresponds to the following expectations, for both the leader and follower vehicles

$$P = E\begin{bmatrix} (\hat{p}(k)-p(k))^2 & (\hat{v}(k)-v(k))(\hat{p}(k)-p(k)) & (\hat{a}(k)-a(k))(\hat{p}(k)-p(k)) \\ (\hat{p}(k)-p(k))(\hat{v}(k)-v(k)) & (\hat{v}(k)-v(k))^2 & (\hat{a}(k)-a(k))(\hat{v}(k)-v(k)) \\ (\hat{p}(k)-p(k))(\hat{a}(k)-a(k)) & (\hat{v}(k)-v(k))(\hat{a}(k)-a(k)) & (\hat{a}(k)-a(k))^2 \end{bmatrix} \tag{6-4}$$

In Section 2-4 the position error is defined in Eq. (2-8) and based on the model presented in Eq. (2-9) it follows that velocity error equation is

$$e_2(k) = v_l(k) - v_f(k) - ha_f(k) \tag{6-5}$$

The measured error based on the MoCap KF results is

$$\hat{e}^*(k) = e(k) + W(k) \tag{6-6}$$

where $e(k)$ includes the position and velocity error terms. The estimation error equation is

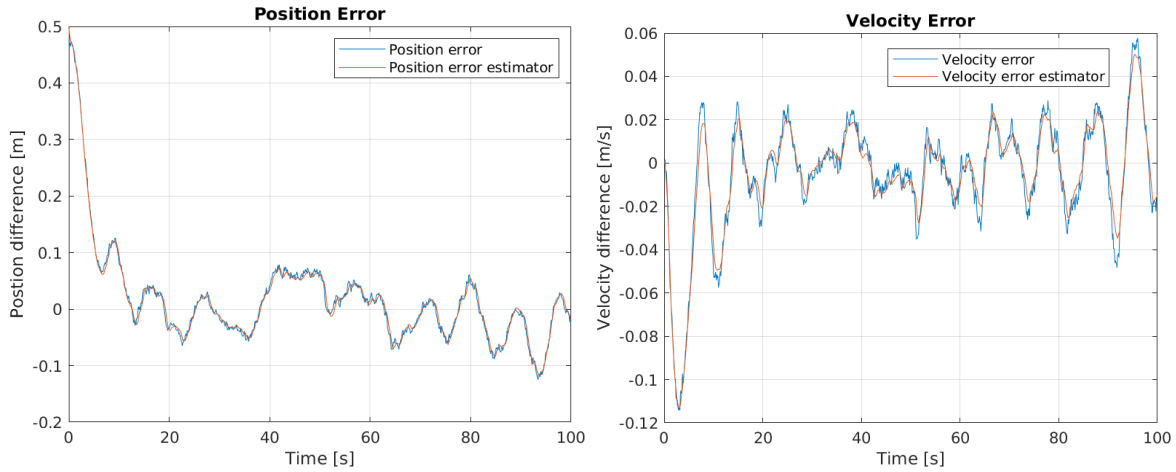
$$\underbrace{\begin{bmatrix} \hat{e}_1^*(k)-e_1(k) \\ \hat{e}_2^*(k)-e_2(k) \end{bmatrix}}_{W(k)} = \underbrace{\begin{bmatrix} \hat{p}_l(k)-p_l(k) \\ \hat{v}_l(k)-v_lk) \end{bmatrix}}_{\xi_l(k)} - \underbrace{\begin{bmatrix} \hat{p}_f(k)-p_f(k) \\ \hat{v}_f(k)-v_f(k) \end{bmatrix}}_{\xi_f(k)} + \underbrace{\begin{bmatrix} h\hat{v}_f(k)-hv_f(k) \\ h\hat{a}_f(k)-ha_f(k) \end{bmatrix}}_{\gamma_f(k)} \tag{6-7}$$

The covariance of the measurement error is the following

$$\begin{aligned} E[WW^T] &= E[(\xi_l-\xi_f-\gamma_f)(\xi_l-\xi_f-\gamma_f)^T] \\ &= E[\xi_l\xi_l^T] + E[\xi_f\xi_f^T] + E[\xi_f\gamma_f^T] + E[\gamma_f\xi_f^T] + E[\gamma_f\gamma_f^T] \\ &= P + P + \begin{bmatrix} hP(1,2) & hP(1,3) \\ hP(2,2) & hP(2,3) \end{bmatrix} + \begin{bmatrix} hP(2,1) & hP(2,2) \\ hP(1,3) & hP(2,3) \end{bmatrix} + \begin{bmatrix} h^2P(2,2) & h^2P(3,2) \\ h^2P(3,2) & h^2P(3,3) \end{bmatrix} \end{aligned} \tag{6-8}$$

In Eq. (6-8) it is assumed that the leader and follower noises are uncorrelated. Because the same MoCap KF was designed for both vehicles, the covariance matrix $P$ is the same for both as considered in Eq. (6-8).

The process noise is part of the vehicle dynamics, hence in the fault detection KF, assuming that the process noises of the leader and follower are uncorrelated, the process covariance is simply sum of the individual process covariances of each vehicle. The results of the fault detection KF can be seen in Figure 6-1.




**(a)** Position error measurements and estimation.

**(b)** Velocity error measurements and estimation.

**Figure 6-1:** Radar simulation.

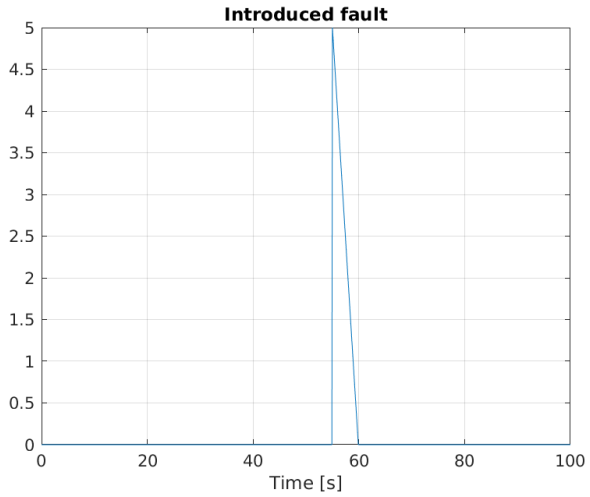
## 6-3 Fault Detection Threshold

Radar measurement data can be subject to faults from numerous sources, namely: radar interference, false vehicle detection, malfunctioning radar. Since in this thesis a virtual radar is simulated, faults in these measurements will be explored.
The following faults will be examined:

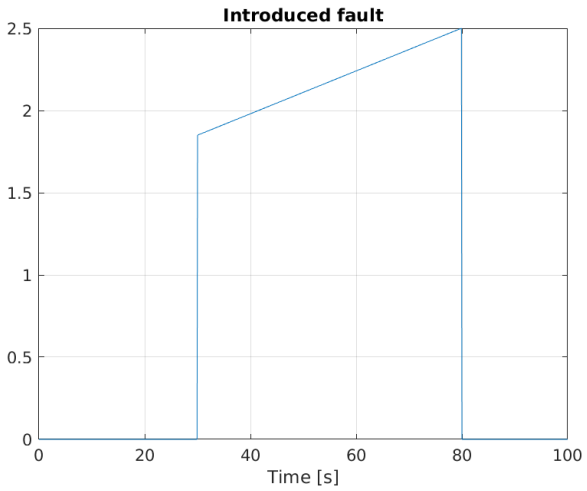
- Radar unexpectedly shuts down and the distance measurement between the leader and follower cars becomes 0.
- Radar starts malfunctioning and instead of continuously measuring the distance between the cars, the output data of a radar is stuck at a constant value.
- Radar locks onto an incoming car from the opposite lane and starts measuring the distance between the mentioned car instead of the leader car.
- Radar locks onto a car driving in the same direction on a parallel lane and starts measuring the distance between the mentioned car instead of the leader.

The last three mentioned fault implementations can be seen in Figure 6-2. The radar shut down fault is not depicted because it is 0. The faults are implemented using a switch, meaning that before the switch, the radar is measuring the relative distance and velocity from the leader vehicle, after the switch, the data seen in Figure 6-2 is read by the virtual radar. The chosen method for malicious behavior detection is model-based fault detection using the KF, which is designed in Section 6-2. Since in this thesis the measurement noise covariances are known and the KF makes the calculation of the measurement residual covariance easy, the square of Mahalanobis Distance (MD) was chosen as a fault detection method. MD, introduced by P. C. Mahalanobis in 1936, is a multivariate generalized measure used to determine the distance of a data point to the mean of the group [40] or, in this case, the distance from the observation to the prediction. It is defined as

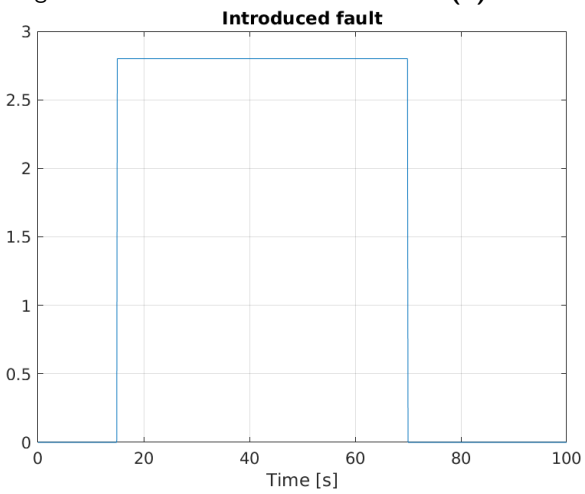
$$M = \sqrt{(y_k - \hat{y}_k)^T E^{-1} (y_k - \hat{y}_k)} \tag{6-9}$$




**(a)** Incoming car.




**(b)** Car on a parallel lane.




**(c)** Radar stuck on a constant value.

**Figure 6-2:** Fault implementations.

where $y_k$ is sensor measured data and $\hat{y}_k$ is its estimate. In the context of the KF, $E$ is defined as the covariance matrix of the innovation sequence

$$E = CPC^T + R$$

where $R$ is measurement noise covariance matrix, $P$ is state (prediction) error covariance matrix and $C$ is the observation matrix [7].
In order to perform fault detection, a threshold must be determined based on the square of MD, which should be Chi-square distributed with the dimensions of the observation vector being the degree of freedom. It is used to describe the distribution of a sum of squared random variables [20].
One way of determining a threshold is to come up with a relatively small upper bound $p$, if $\text{MD}^2 \leq p$, the estimate and the measurement matches, which means that no fault has been introduced to the radar. However, the measurements are considered to be faulty if the square of MD exceeds the defined bound. An alternative to using a $p$ value is using a predetermined

$\alpha$-quantile $\chi_\alpha$ of the Chi-square distribution. It is determined based on the degree of freedom and the significance level (probability) of the Chi-square distribution. The significance levels are usually chosen to be equal to 0.01, 0.05 or 0.10.

| | $\alpha$ | | | | | |
|---|---|---|---|---|---|---|
| **DF** | **0.50** | **0.10** | **0.05** | **0.02** | **0.01** | **0.001** |
| 1 | 0.455 | 2.706 | 3.841 | 5.412 | 6.635 | 10.827 |
| 2 | 1.386 | 4.605 | 5.991 | 7.824 | 9.210 | 13.815 |
| 3 | 2.366 | 6.251 | 7.815 | 9.837 | 11.345 | 16.268 |
| 4 | 3.357 | 7.779 | 9.488 | 11.668 | 13.277 | 18.465 |

**Table 6-1:** Values of the Chi-squared distribution [32].

In this work the observation vector consists of 2 values, position error and velocity error, therefore the number of degrees of freedom is equal to 2. For a 2-degree-of-freedom Chi-square distribution a significance level of 1% was chosen in this work. Then, according to the look-up table in Table 6-1, the $\alpha$-quantile is equal to 9.210. If the calculated square of the MD is higher than 9.210 or, if the probability of getting the result is less than 1%, the data is considered faulty, Figure 6-3.

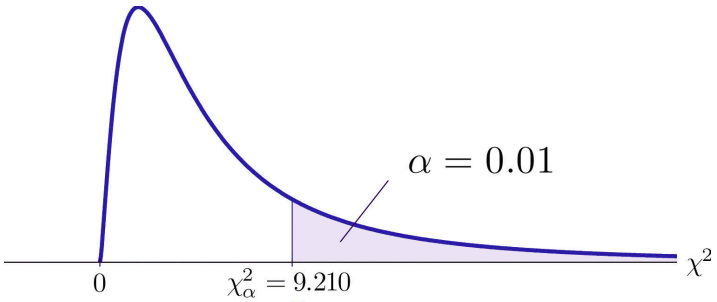



**Figure 6-3:** Rejection region [39].

# Chapter 7

# Simulation results

## 7-1 Introduction

The real Cooperative Adaptive Cruise Control (CACC) platoon has each vehicle equipped with a radar, which provides the required measurement data to compute inputs to the follower cars. In the Networked Embedded Robotics DCSC lab (NERDlab), the experimental setup does not include a radar. Instead, the Motion Capture (MoCap) system is used, which provides position data of each Radio Control (RC) vehicle, velocity and acceleration are derived from it. The measurements of the follower car are subtracted from the leader to simulate a simplified radar. This simulation aims to mimic the provided experimental setup and explore the effects of faulty radar measurement data.

## 7-2 Simulation Scenarios

The simulation is ran with the following constant parameters which can be seen in Table 7-1.

| Parameter | Value |
|---|---|
| Time headway $h$ | 0.7 |
| Time constant $\tau$ | 0.6 |
| Desired distance $r$ | 0.5 |
| Vehicle length $L$ | 0.53 |
| Gain $k_p$ | 0.2 |
| Gain $k_d$ | 0.7 |
| Significance level $\alpha$ | 0.9210 |

**Table 7-1:** Simulation values.

Time headway $h$ was chosen based on literature and time constant $\tau$ was identified in Subsection 4-5. The desired distance $r$ was chosen arbitrary, the smaller the distance the higher

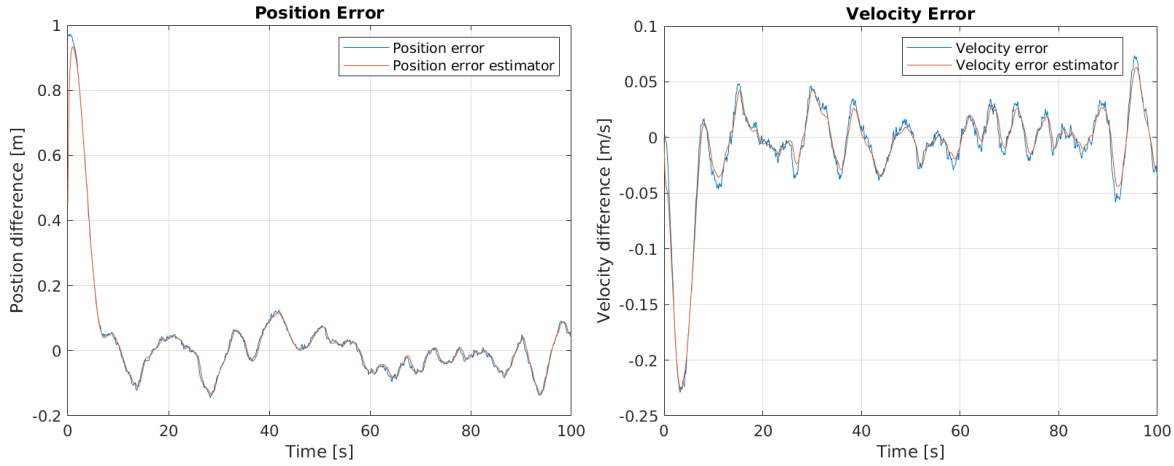
the chances of a crash once the fault is introduced. Vehicle length $L$ was given in the specifications of the Erle-Rover. The $k_p$ and $k_d$ gains were tuned based on the performance of the controller. The chosen judging index for identifying fault is equal to 9.210 and the choice is based on the Chi-squared distribution values in Table 6-1. The observation vector consists of 2 values, position error and velocity error, therefore the degree of freedom is equal to 2. The chosen significance level is 0.01 that yields the Chi-square distribution value of 9.210.
Simulation is ran 5 times:

1. Without any fault present.
2. Fault is introduced as unexpected shutdown of the simulated radar.
3. Fault is introduced as a glitch in the radar which makes it output a constant value.
4. Fault is introduced as radar locking onto incoming car from an opposite lane instead of a vehicle ahead on the same lane.
5. Fault is introduced as radar locking onto a car driving on a parallel lane instead of a vehicle ahead on the same lane.

## 7-3 Results

### 7-3-1 No Faults Present

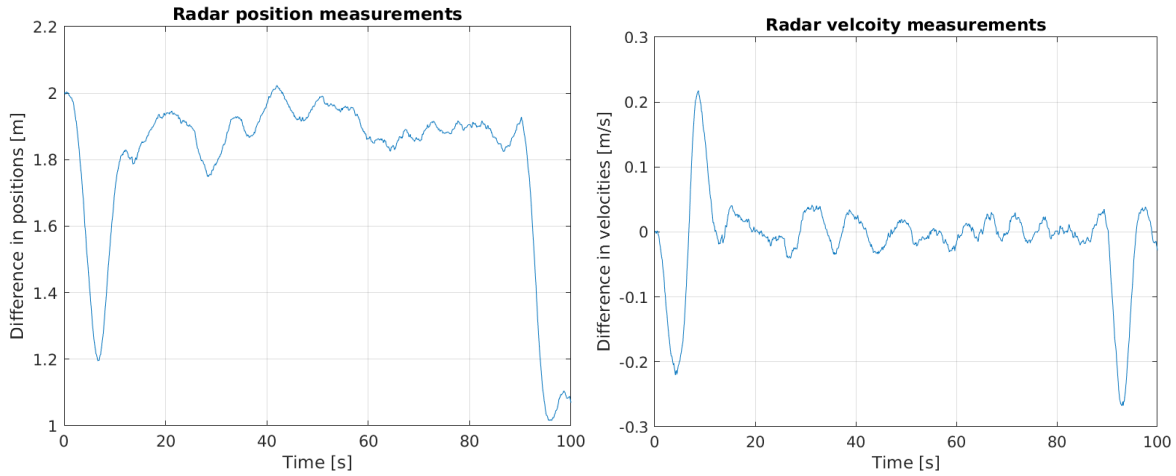
First, the simulation is ran without injecting the faults to radar measurements. As can be seen in Figure 7-1, when no faults are present, position and velocity errors are fluctuating around zero. The radar measurements are simply the difference in positions and velocities of the two




**(a)** Position error. **(b)** Velocity error.

**Figure 7-1:** Position and velocity errors when no faults are present.

Erle-Rovers. Because the controller performs well, the difference in velocities between the two vehicles is around zero, Figure 7-2b. Only in the beginning and the end of simulation there is a slight difference when the vehicles are accelerating and breaking. Vehicles follow each

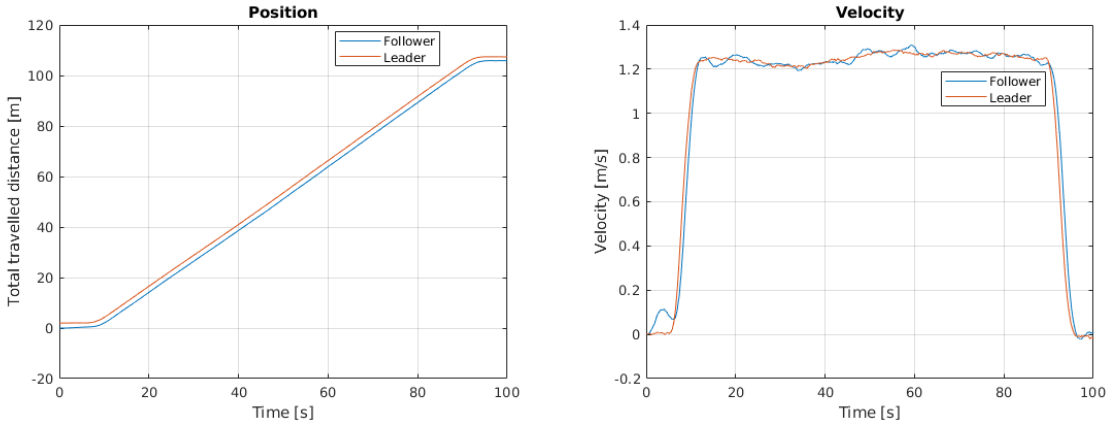



**(a)** Difference in positions between the leader and follower.

**(b)** Difference in velocities between the leader and follower.

**Figure 7-2:** Radar measurements when no faults are present.

other keeping 0.5 meter distance, as can be seen in Figure 7-3a. Velocities of both vehicles are approximately 1.2 $m/s$. In Figure 7-3b it can be seen that in the beginning of the simulation the follower vehicle accelerates before the leader vehicle does and then matches the speed. It happens because of the initial conditions, follower is further away than the desired distance $r$, and it immediately tries to correct this position error.




**(a)** Leader and follower cars position data.

**(b)** Leader and follower cars velocity data.

**Figure 7-3:** Positions and velocities of leader and follower cars.

When no fault is present, square of Mahalanobis Distance (MD) values should be around zero. It can be seen in Figure 7-4, that the MD results are below the threshold, which means that no faults are detected in the provided measurements.

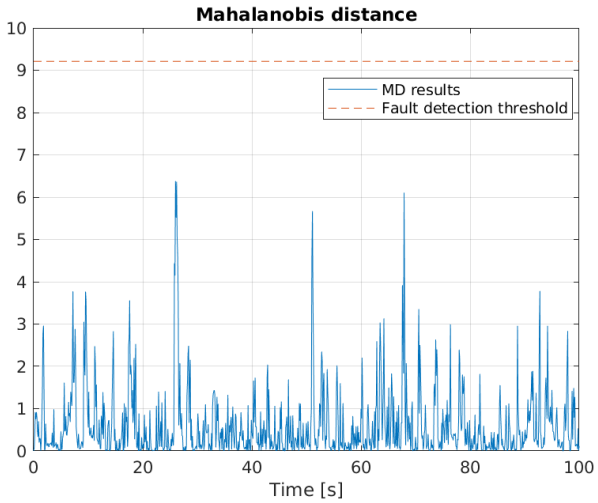
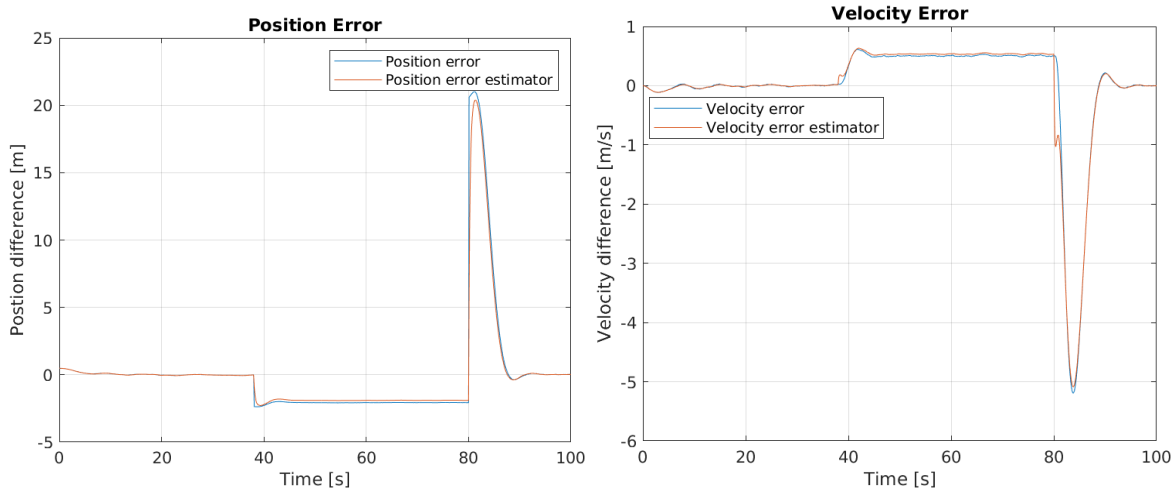



**Figure 7-4:** Mahalanobis distance results when no faults are present.

### 7-3-2 Radar Shutdown

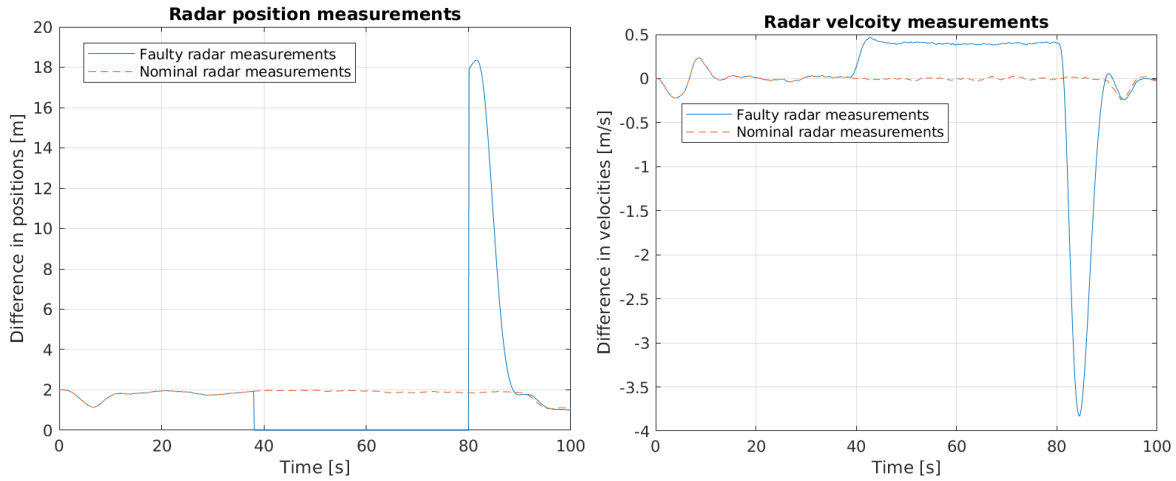
In this section, the first type of fault is introduced, which is the unexpected radar shutdown. Position and velocity errors are no longer approximately zero, Figure 7-5. Since the system is malfunctioning, the difference in Kalman Filter (KF) estimated state and the simulated measurement data is greater compared to Figure 7-1, when no faults are present.




**(a)** Position error. **(b)** Velocity error.

**Figure 7-5:** Positions and velocity errors when the radar shuts down.

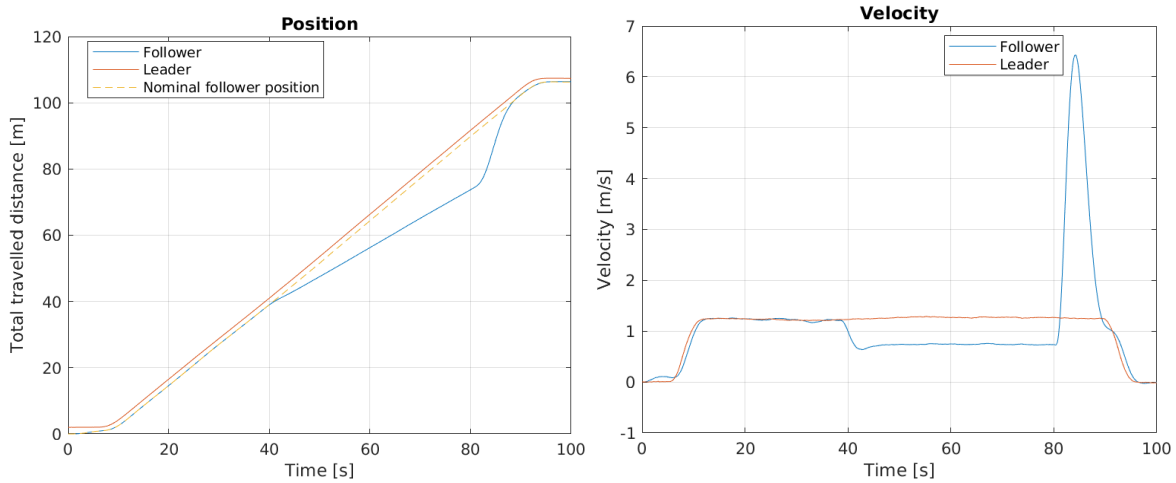
In the middle of the simulation, the radar stops measuring position difference, which also impacts the difference in velocities, Figure 7-6. This fault greatly impacts the platoon because the follower car starts lagging once the fault is introduced, Figure 7-7a. When the radar starts measuring again, the follower must catch up with the leader car and therefore the velocity of the second Erle-Rover has to overshoot. Even though vehicles do not crash in the provided plots, in the NERDlab vehicles are programmed to go in circles and therefore the slowdown




**(a)** Position radar measurements. **(b)** Velocity radar measurements.

**Figure 7-6:** Radar measurements when the radar shuts down.

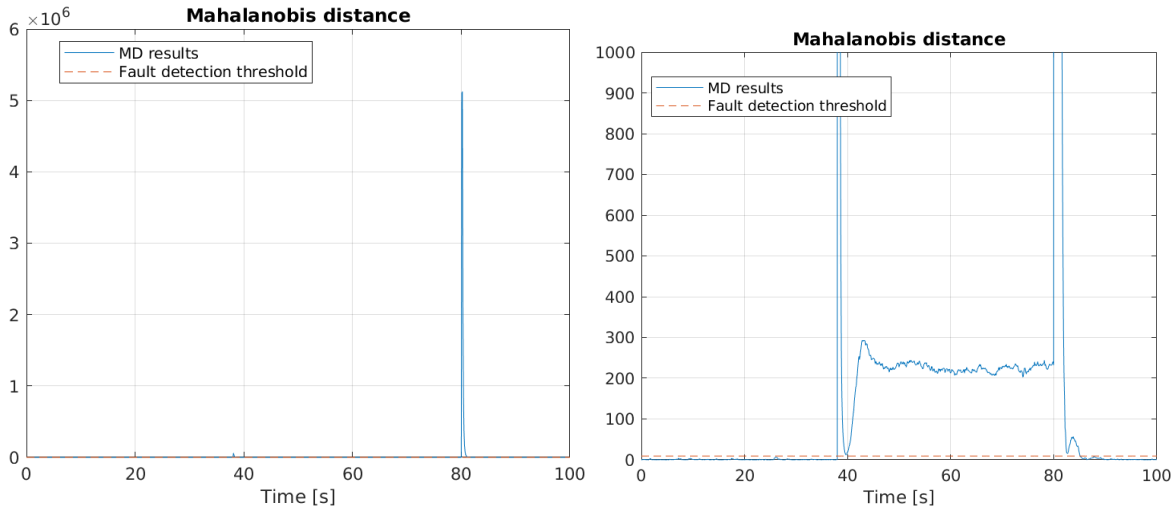
of the follower car could potentially result into the leader car crashing into the follower. Such crash would not be visible in the provided plots because the total travelled distance is plotted instead of the trajectories of both vehicles. However, this is an artifact of the setup in the NERDlab, but this does not affect the real CACC scenario.




**(a)** Leader and follower cars position data. **(b)** Leader and follower cars velocity data.

**Figure 7-7:** Positions and velocities of leader and follower cars.

There are 2 peaks in MD results, Figure 7-8. The first one appears at the time the radar shuts down and the second once it starts measuring again. Even though, in between these peaks, in Figure 7-8a, the value seems close to zero, in Figure 7-8b it can be seen that it is between 200 and 300.

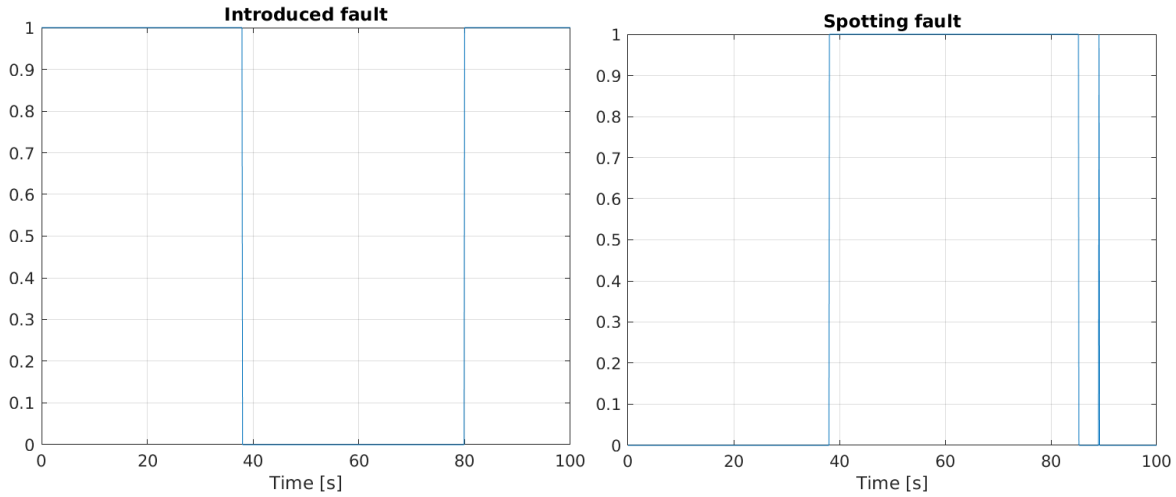



**(a)** Mahalanobis distance results when the radar shuts down.

**(b)** Scaled Mahalanobis distance results when the radar shuts down

**Figure 7-8:** Mahalanobis distance results when the radar shuts down.

The fault is introduced at 38-th second and stops at 80-th second. Because of the significant difference between the estimated error and the measured one, it is easy to identify the fault. However, according to results in Figure 7-9, the fault lasts 5 seconds longer and a false positive is identified at 89-th second. This happens because of the time the vehicle requires to adjust its position and go back to expected error dynamics (zero).



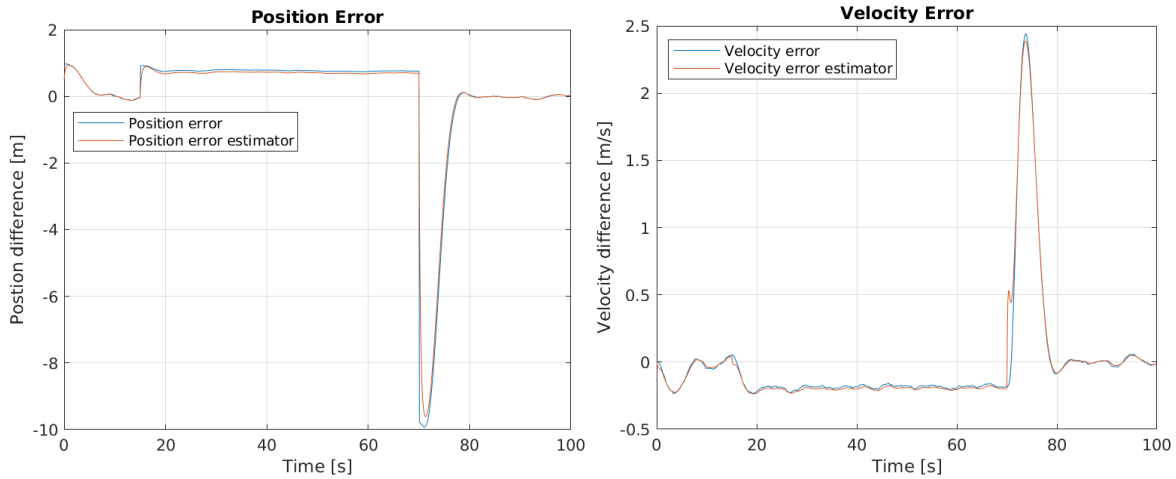

**(a)** Introduced fault.

**(b)** Spotting fault.

**Figure 7-9:** Introduced fault and its detection.

### 7-3-3 Radar Outputs a Constant Value

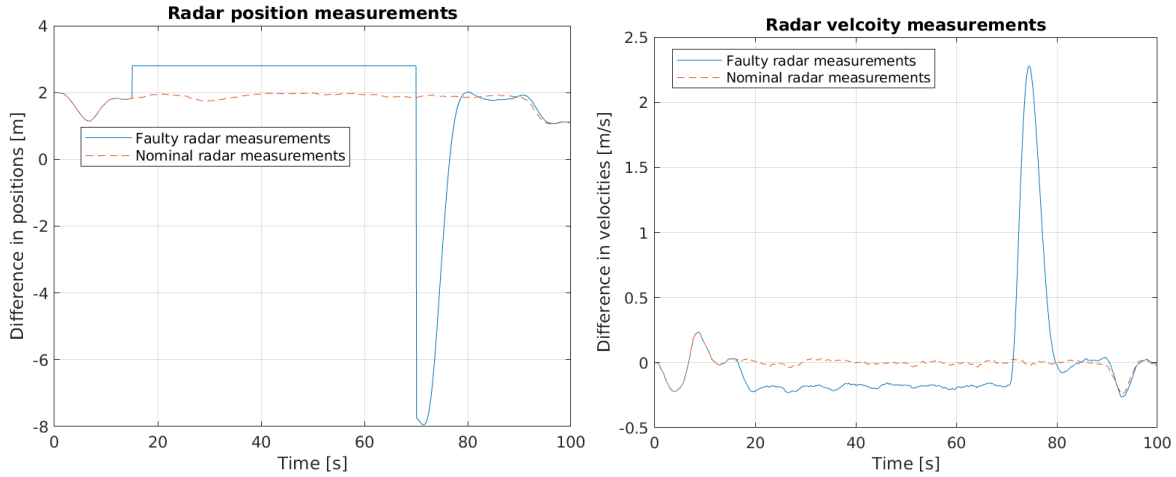
In this section, the fault of the radar, outputting a constant value, is explored. In position error data in Figure 7-10a, the fault introduction can be easily spotted not only because of the occurring peaks, but also because of the mismatch between the estimate and the measurement when the fault begins and ends. The same cannot be said about the velocity error, only when the radar is starting to measure properly again, a clear peak as well as the mismatch between the estimate and measurement can be seen, Figure 7-10. Because the introduced constant




**(a)** Position error. **(b)** Velocity error.

**Figure 7-10:** Positions and velocity error when the radar outputs a constant value.

fault value is very close to the actual distance between the two vehicles, the difference in velocities remains around zero as long as the fault is present, Figure 7-11b.

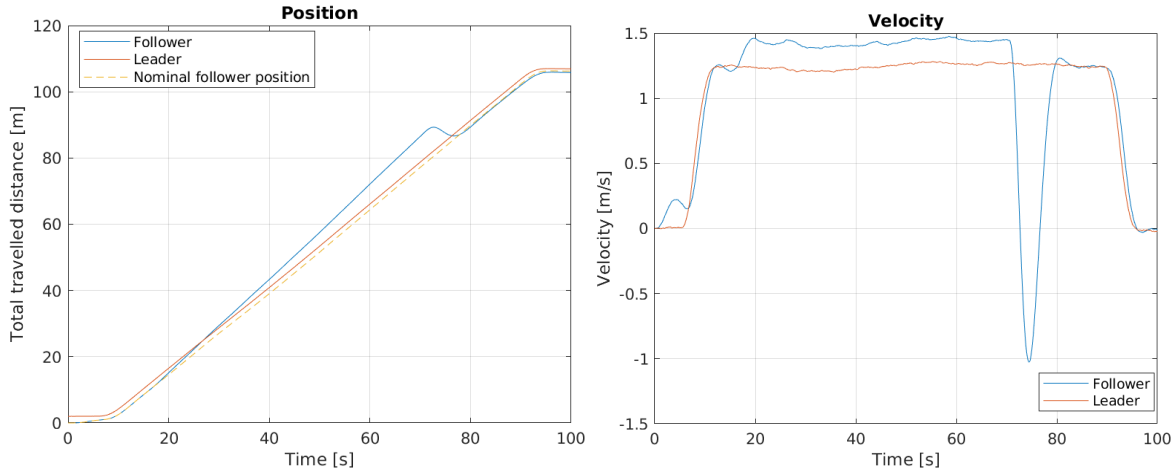



**(a)** Position radar measurements. **(b)** Velocity radar measurements.

**Figure 7-11:** Radar measurements when the radar outputs a constant value.

Because the positional error between the two cars is greater than zero, it causes the follower to speed up, which results into the two vehicles crashing, Figure 7-12a. It is clear from

Figure 7-12b that the follower exceeds the speed of the leader. The square of MD can easily

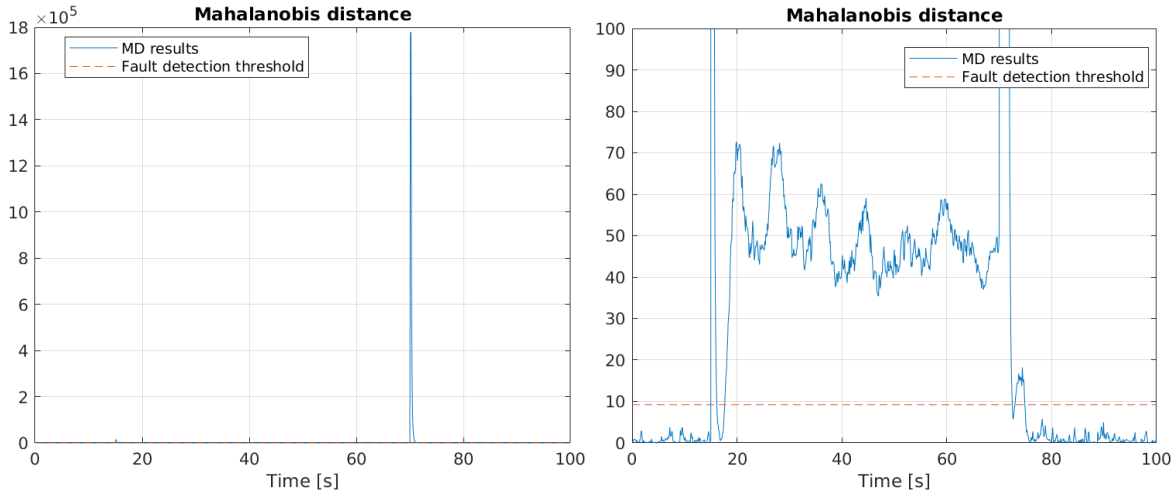



**(a)** Leader and follower cars position data.

**(b)** Leader and follower cars velocity data.

**Figure 7-12:** Positions and velocities of leader and follower cars.

pick up on the introduced fault. The peak, which appears after the fault is switched off, is visible in both images depicted in Figure 7-13. In Figure 7-13b, the square of MD is scaled so that smaller peaks could be visible as well.

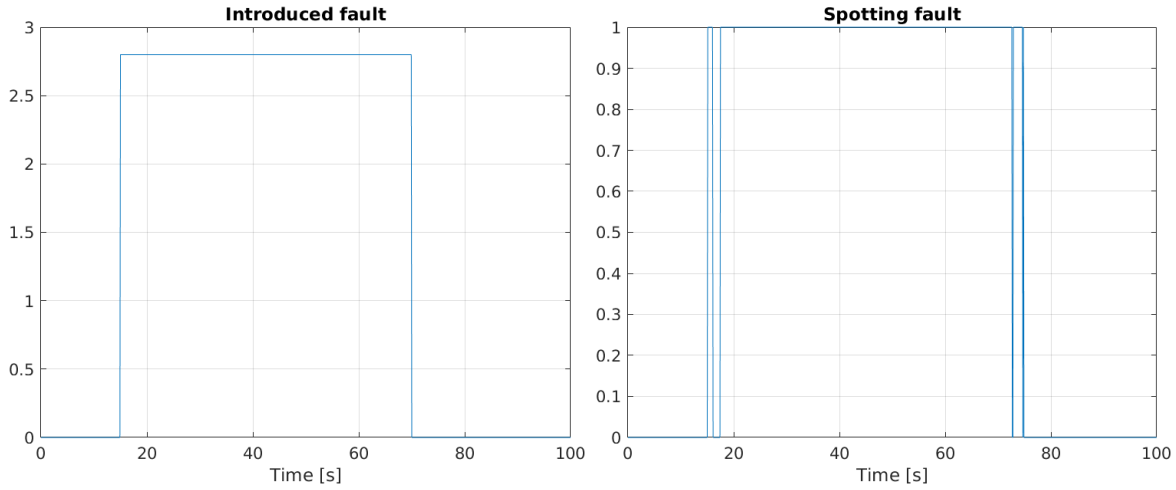



**(a)** Mahalanobis distance.

**(b)** Scaled Mahalanobis distance.

**Figure 7-13:** Mahalanobis distance when the radar outputs a constant value.

The fault is introduced at 15-th second and ends at 70-th. The fault detection in Figure 7-14b is being detected for a longer time than it is introduced for. It happens because it takes time for the follower vehicle to correct its position (position error must be close to zero).

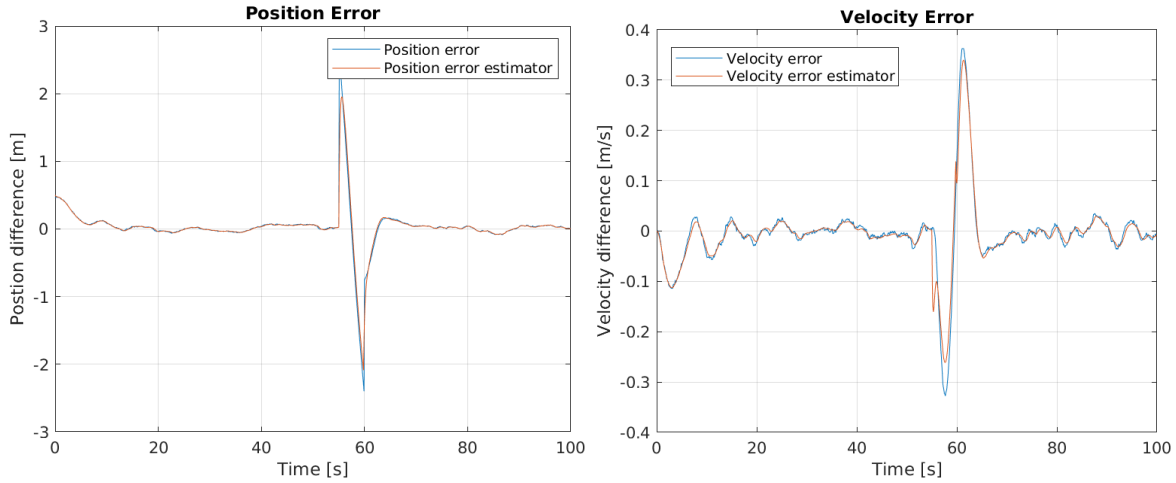



**(a)** Introduced fault. **(b)** Spotting fault.

**Figure 7-14:** Introduced fault and its detection.

### 7-3-4 Radar Locks onto Incoming Car

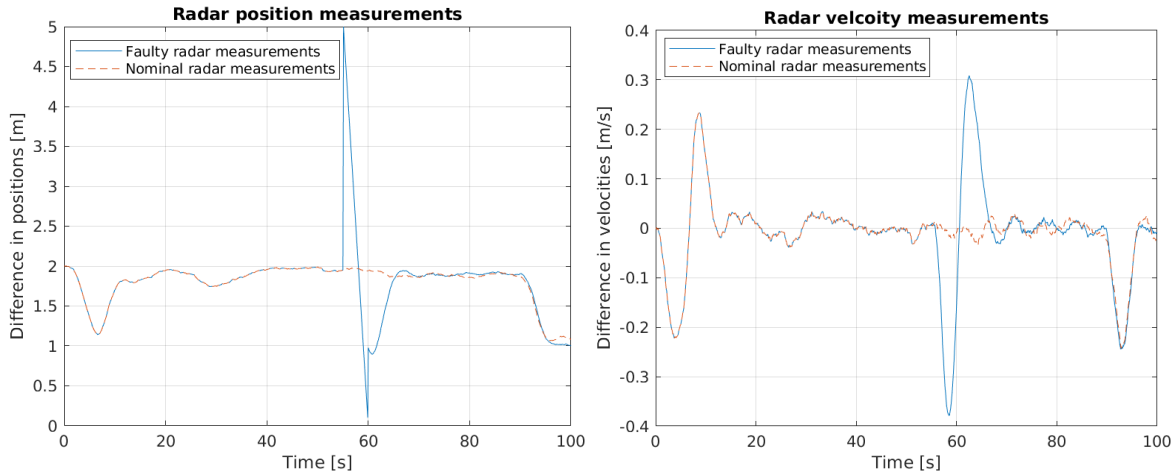
In this section, radar locking onto incoming car on the opposite lane will be simulated. At the time the foreign vehicle is introduced, a clear peak in Figure 7-15 can be seen, where a very clear difference in the estimate and measurement can be noticed.




**(a)** Position error. **(b)** Velocity error.

**Figure 7-15:** Positions and velocity errors when the radar locks onto incoming vehicle.

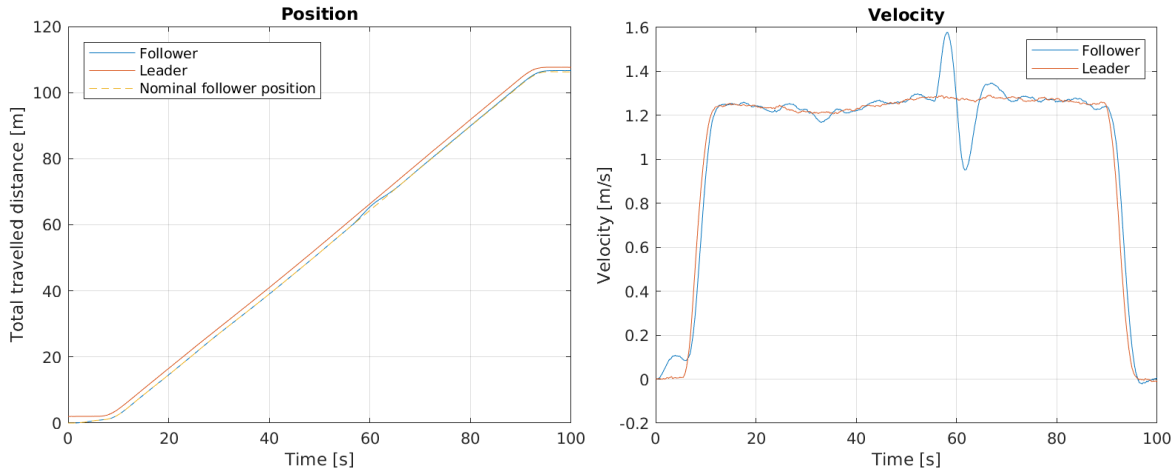
In the radar measurements, a ramp is visible in Figure 7-16a and the inverse of it in Figure 7-16b. Because to the follower car the leader appears to be much further away, it starts accelerating, and once the incoming car reaches the same position as the radar carrying car, it tries to decelerate to avoid the crash.




**(a)** Position radar measurements. **(b)** Velocity radar measurements.

**Figure 7-16:** Radar measurements when the radar locks onto incoming vehicle.

In figure Figure 7-17a, it can be seen that the Erle-Rovers had almost crashed. If the incoming car had been located at a further distance, the follower would have slammed into the leader. The velocity of the follower first overshoots and then undershoots because of false detection. The fault can be very clearly seen from the quare of MD results in Figure 7-18. The first




**(a)** Leader and follower cars position data. **(b)** Leader and follower cars velocity data.

**Figure 7-17:** Positions and velocities of leader and follower cars.

peak indicates the start of fault detection, the second - the end. In this scenario the results of MD in between the peaks are clearly non zero, which makes this fault easily detectable.

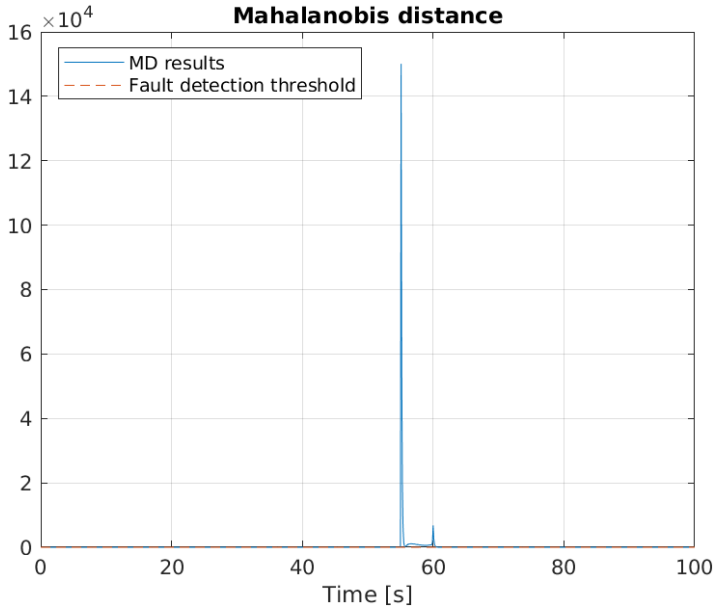
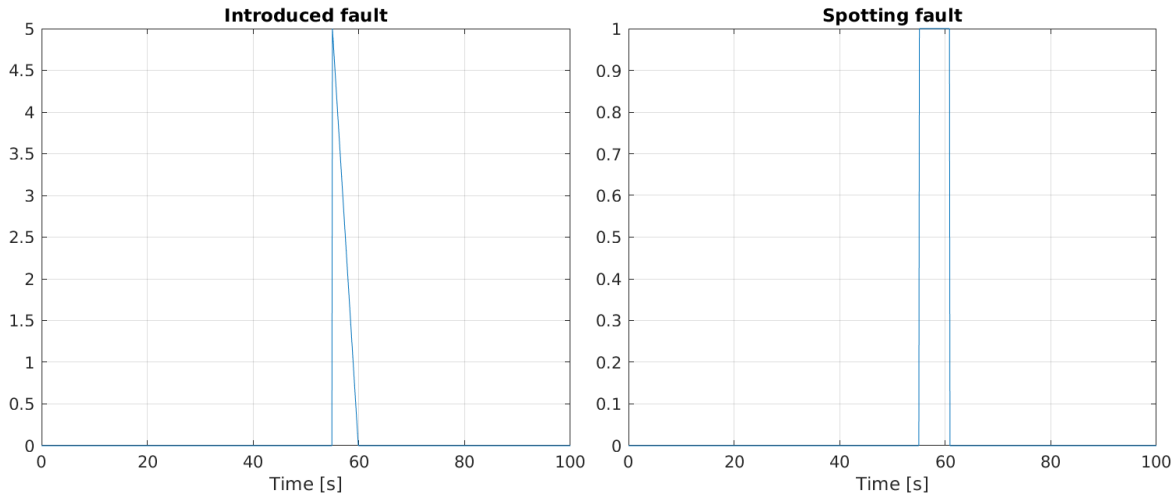



**Figure 7-18:** Mahalanobis distance results when the radar locks onto incoming vehicle.

The incoming car is introduced to the system as a ramp which starts 5 meters away from the follower car and quickly reaches zero. The fault is introduced at 35-th second and ends at 60-th second. The detection time of the beginning of the fault is correct, the end is delayed by a second because of the time it takes for the follower to go back to the expected error dynamics.

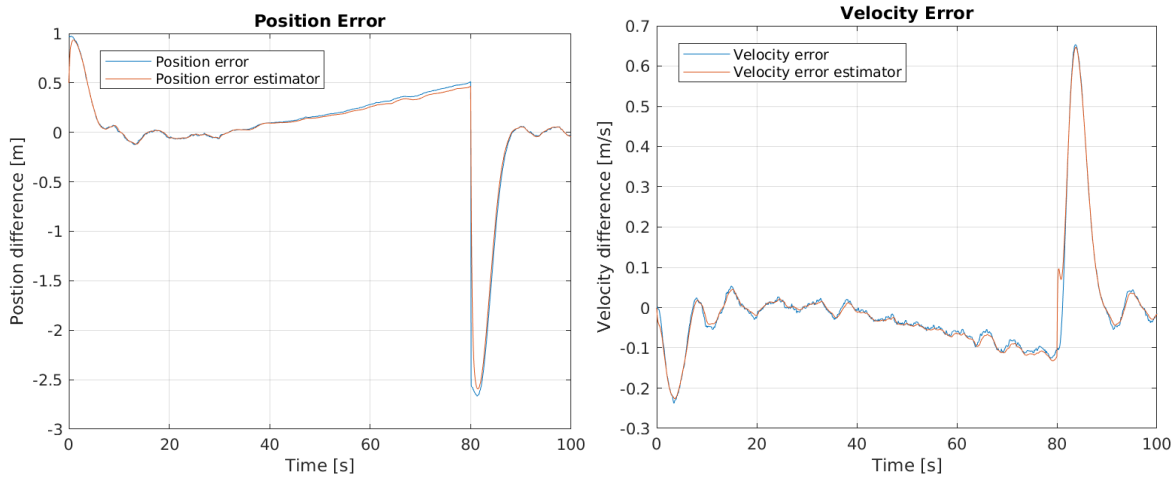



**(a)** Introduced fault. **(b)** Spotting fault.

**Figure 7-19:** Introduced fault and its detection.

### 7-3-5 Radar Locks onto a Car in a Parallel Lane

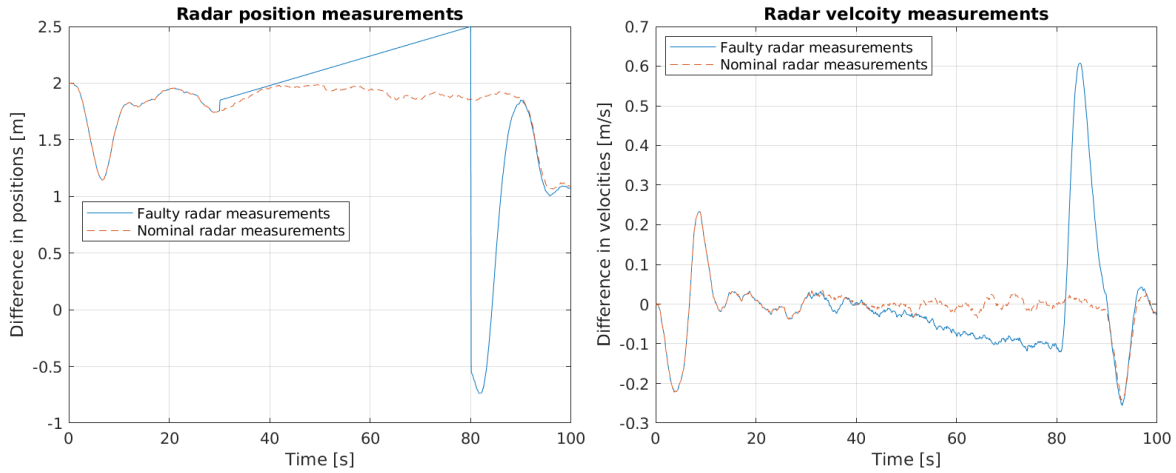
The last explored fault scenario is radar locking onto a vehicle going the same direction but in the parallel lane. Because the vehicle the radar locks onto slowly offsets the position and velocity errors, at the beginning of fault injection, the estimate and the measurement are almost the same. As time goes by, they start to divert from one another, as can be seen in Figure 7-20a.




**(a)** Position error. **(b)** Velocity error.

**Figure 7-20:** Positions and velocity errors when the radar locks onto a parallel vehicle.

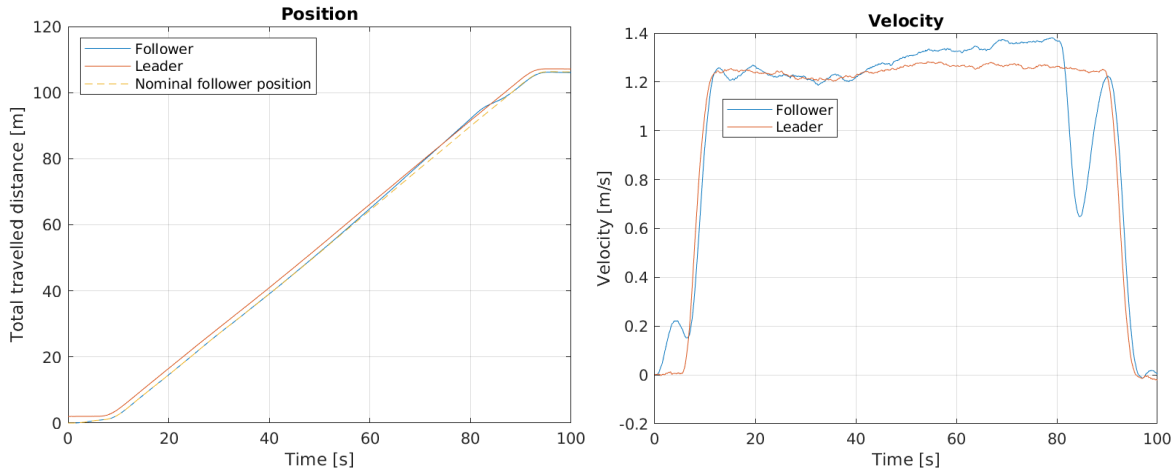
It is clear from Figure 7-21 that the vehicle in the parallel lane is going at almost the same speed as the leader, which makes such a fault hard to detect. Because the interfering vehicle




**(a)** Position radar measurements. **(b)** Velocity radar measurements.

**Figure 7-21:** Radar measurements when the radar locks onto a parallel vehicle.

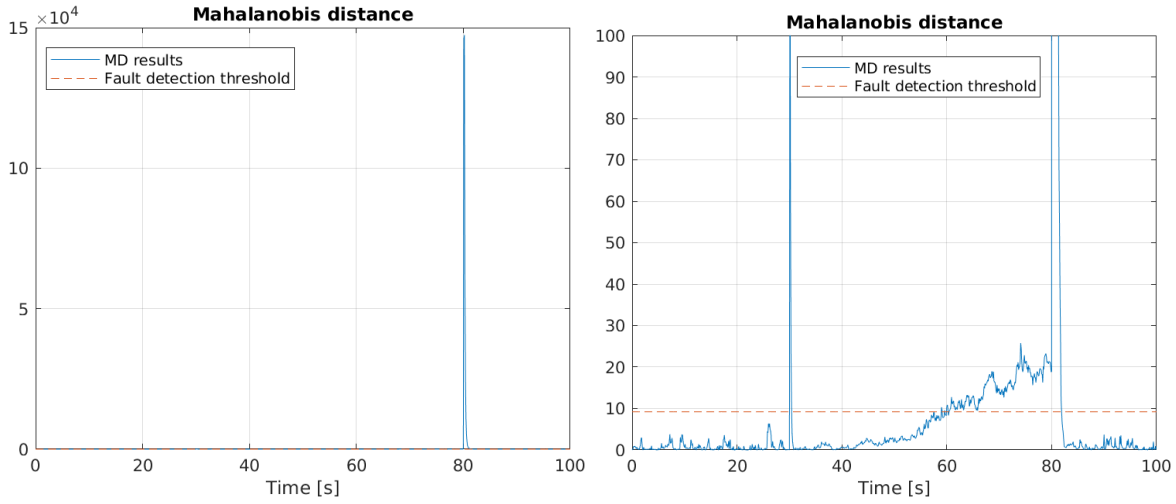
is slightly faster, the follower Erle-Rover crashes into the leader Figure 7-22a. Such fault scenario causes the follower to exceed the desired velocity, Figure 7-22b.




**(a)** Leader and follower cars position data. **(b)** Leader and follower cars velocity data.

**Figure 7-22:** Positions and velocities of leader and follower cars.

Detecting such a fault using the square of MD fails in this scenario. The only visible peak in Figure 7-23a is when the radar starts detecting the leader car again. In scaled Figure 7-23b it can be seen that up to the 60-th second, the results of MD match the results before the fault was introduced. Because the speed of a vehicle in the parallel lane matches that of the leader car, detecting unexpected radar switch is not possible with the provided information. The

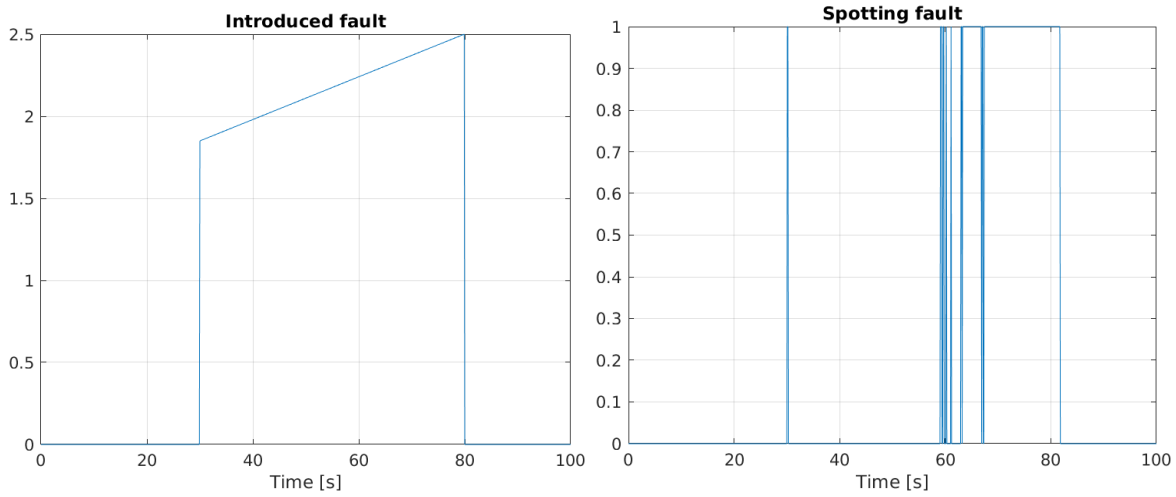



**(a)** Mahalanobis distance. **(b)** Scaled Mahalanobis distance.

**Figure 7-23:** Mahalanobis distance results when the radar locks onto a parallel vehicle.

fault is introduced at 30-th second and ends at 80-th second. However, the fault is spotted as a short peak at 30-th second and is detected for 0.2 second. Then, it continues to state that the measurements are not faulty for the next 25 seconds. Such an instant and short peak is not enough to discard the rest of the measurements and could be interpreted as a false positive just like it was the case towards the end of the constant fault and radar shutdown simulations. At approximately 60-th second, the fault detection starts outputting positive results every $\sim 0.15$ second and at approximately 68-th second, it starts to clearly state that

the measurement data is corrupted, as can be seen in Figure 7-24. During this simulation, the fault was detected (with multiple positives) in time to avoid the crash. With such detection, it would be possible to prevent the crash, since it happens a few second after the multiple detection peaks appear. However, detecting radar measuring a car in a parallel lane instead of the leader, proves to be the hardest to detect in the presented scenarios.




**(a)** Introduced fault. **(b)** Spotting fault.

**Figure 7-24:** Introduced fault and its detection.

## 7-4 Result Analysis

Based on the simulation results, it is clear that the most challenging fault to detect is radar locking onto a car in a parallel lane. Radar shutdown or radar being stuck at a constant value is easy to detect because of the clear switch from the measurements to a predefined constant value. Radar locking onto an incoming car is also easy to detect since vehicles are getting closer to each other much faster than what the KF estimates. However, when the radar locks onto a vehicle in a parallel lane, the fault is detected 30 seconds after the fault is introduced. That happens because the vehicle in the parallel lane moves at a similar speed.
Fault detection would greatly improve if multiple sensors could be used instead of one to measure the position and velocity data of both vehicles. One sensor can cause false positives and is not reliable enough to conclude that the data is faulty. An available sensor, which is already used in cars, is a vehicle speed sensor. It is used to verify the speed of the car wheels. It could potentially be used to compare the relative velocity measured by the radar with the measurements of the vehicle speed sensor.
Another issue with the results is false positives. Apart from verifying the results with another sensor, a possible solution would be determining a threshold for how long the fault must be detected or how many times within a certain period of time. Based on the acquired results, such a threshold could be equal to 1.5 seconds, meaning that any fault detected for less than 1.5 seconds is disregarded unless within that time the fault is detected multiple times. To accurately determine such a threshold, more experiments must be done with various fault injection scenarios.

# Chapter 8

# Conclusions and Future Work

## 8-1 Conclusions

In this thesis, the implementation of Cooperative Adaptive Cruise Control (CACC) on Radio Control (RC) vehicles was explored. The CACC aims to keep each vehicle at a predefined distance from a vehicle ahead by automatically controlling the speed of each vehicle in the platoon. In the ideal scenario each vehicle carries a radar, which measures the relative distance and velocity from the vehicle ahead. These measurements are then used to compute the control law, which ensures the desired inter-vehicle distance is being kept. However, the experimental setup did not include the radar and therefore it was simulated using the data provided by the Motion Capture (MoCap). The virtual radar was simplified to the difference in position and velocity between the leader and follower vehicles. The dynamics of the Erle-Rover were modeled and identified based on the collected position data while the Networked Embedded Robotics DCSC lab (NERDlab) could be accessed. The identification was done by writing an optimization algorithm that minimized the error between the measured position data and the estimate.This model was then simulated in Simulink, where CACC, which relies on inter-vehicle distance and relative velocity measurements, was implemented. This data was provided by the simulated virtual radar, which was corrupted with 4 different types of faults. The chosen method for fault detection was model-based fault detection using the Kalman Filter (KF) for estimating the expected measurement data, and the square of Mahalanobis Distance (MD) to detect changes in the statistical properties of its prediction-errors. This method was chosen because it incorporates the knowledge of the mathematical model of the vehicle, inputs to the system and the measured outputs, and tackles the problem of the noisy measurement data. The goal of this thesis was to spot these faults on time and analyze whether they could be identified before they impact the platoon.
The first fault injected into the simulated platoon was the complete radar shutdown. Instead of providing distance measurements from the follower to the leader car, it was outputting zero. This fault was easily detected during simulation because of a significant difference in measured and estimated error dynamics. Radar shutdown caused the follower car to slow down because the provided measurement data stated that the vehicles were much closer to

each other than they actually were. This fault was identified at the time it was injected.
The second explored fault was the constant fault, which was outputting 2.8 meters as the distance between the two rovers. This fault was also easily detected on time for the same reason as the previous one. The fault of the radar locking onto an incoming car can be precisely detected as well because of a sudden change in the difference of velocities between the follower car and what was assumed to be the leader.
The radar locking onto a vehicle in a parallel lane proved to be the hardest fault to detect. If both vehicles are going at similar speeds, it is a challenge to detect such a fault and in the simulation the detection failed.
Out of 4 presented fault scenarios, 3 were identified on time. The fault of radar locking onto a vehicle in the parallel lane, was the only one, which was not identified before it affected the behavior of the platoon. As was discussed in the results analysis, this could be solved by introducing another sensor to a vehicle, with which the radar measurements would be verified. Therefore, according to the results, it is possible to come up with a precise fault detection methods for certain scenarios, but, in this thesis, it does not cover all the possible faults.

## 8-2 Future Work

The presented work can be expanded upon in multiple different ways, among which:

- The presented model-based fault detection algorithm should be implemented and evaluated on the Erle-Rovers.
- More MoCap data should be gathered to improve the identified vehicle model.
- Instead of simulating a radar, a real one could be implemented in practice and the results could be weighted against the MoCap system results or another sensor of choice to improve fault detection.
- Only longitudinal dynamics were explored in this thesis. Therefore, a new model that includes lateral dynamics could be derived and implemented on the Erle-Rovers. This model would also enable experiments on more complex trajectories.
- Multiple fault detection methods should be explored. The square of MD would not always be an ideal choice for malicious behavior detection because it requires knowledge of noise and process covariances of a sensor, which are not always known.

# Appendix A

# The appendices

## A-1 Custom ROS Node

### A-1-1 Header File

```
#ifndef ROVER_CONTROL_H_
#define ROVER_CONTROL_H_

#include <ros/ros.h>
#include <mavros_msgs/OverrideRCIn.h>
#include <mavros_msgs/RCIn.h>
#include <geometry_msgs/Twist.h>
#include <ros/callback_queue.h>
class roverControl
{
private:
ros::NodeHandle rovernh_;
ros::Subscriber pose_;
ros::Publisher pub_;
int throttle = 1600;
int steering = 1700;
int step = 5;
int throttle_max = 1900;
int throttle_min = 1300;
int steering_max = 1800;
int steering_min = 1300;
const int receivedT=0, receivedS=0;
bool start = false;
void incrementThrottle();
void decrementThorttle();
void incrementSteering();
void decrementSteering();
void initializePublisher();
```

```
29 void initializeSubsribers();
30 void check(int a, int b);
31 public:
32     roverControl(ros::NodeHandle* rover_handle);
33     void msgVel(const geometry_msgs::Twist msgVel);
34     void publishVelocity();
35 };
36
37 #endif
```

### A-1-2 Main Program

```
1  #include "rover_control.h"
2  roverControl::roverControl(ros::NodeHandle* rover_handle):rovernh_(*
       rover_handle)
3  {
4      initializePublisher();
5      initializeSubsribers();
6  }
7
8  void roverControl::initializeSubsribers(){
9      pose_ = rovernh_.subscribe("turtle1/cmd_vel", 1000, &roverControl::
           msgVel, this);
10 }
11
12 void roverControl::initializePublisher(){
13     pub_ = rovernh_.advertise<mavros_msgs::OverrideRCIn>("/leader_mavros/
           rc/override", 1000);
14 }
15
16 void roverControl::msgVel(const geometry_msgs::Twist msgVel){
17     check(msgVel.linear.x, msgVel.angular.z);
18 }
19
20 void roverControl::check(int a, int b){
21     if(a>0){
22         ROS_INFO_STREAM("Throttle command received");
23         incrementThrottle();}
24     else if(a<0){
25         ROS_INFO_STREAM("Throttle command received");
26         decrementThorttle();}
27     if(b>0){
28         ROS_INFO_STREAM("Steering command received");
29         decrementSteering();}
30     else if(b<0){
31         ROS_INFO_STREAM("Steering command received");
32         incrementSteering();}
33 }
34 void roverControl::publishVelocity(){
35     mavros_msgs::OverrideRCIn out_RC;
36     ROS_ERROR_STREAM("Steering="<<steering);
37     ROS_ERROR_STREAM("Throttle="<<throttle);
38     out_RC.channels[0] = steering;
```

```
39      out_RC.channels[1] = throttle;
40      out_RC.channels[2] = throttle;
41      out_RC.channels[3] = throttle;
42      out_RC.channels[4] = throttle;
43      out_RC.channels[5] = throttle;
44      out_RC.channels[6] = throttle;
45      out_RC.channels[7] = throttle;
46      pub_.publish(out_RC);
47  }
48
49  void roverControl::incrementSteering(){
50      if(steering<steering_max){steering+=step;}
51  }
52  void roverControl::incrementThrottle(){
53      if(throttle<throttle_max){throttle+=step;}
54  }
55
56  void roverControl::decrementThorttle(){
57      if(throttle>throttle_min){throttle-=step;}
58  }
59  void roverControl::decrementSteering(){
60      if(steering>steering_min){steering-=step;}
61  }
62
63  int main(int argc, char** argv){
64      ros::init (argc, argv, "roverControl");
65      ros::NodeHandle nh;
66      ROS_INFO("main: instantiating an object of type roverControl");
67      roverControl examplecontrol(&nh);
68      ROS_INFO("main: going into spin; let the callbacks do all the work");
69      ros::Rate rate(3);
70      while (ros::ok()){
71          ros::getGlobalCallbackQueue()->callAvailable(ros::WallDuration
                (0.1));
72          examplecontrol.publishVelocity();
73          rate.sleep();
74      }
75      return 0;
76  }
```

## A-2 Rostopic List of the Erle-Brain 3

```
1  /camera/camera_info
2  /camera/image/compressed
3  /diagnostics
4  /mavlink/from
5  /mavlink/to
6  /mavros/actuator_control
7  /mavros/battery
8  /mavros/cam_imu_sync/cam_imu_stamp
9  /mavros/extended_state
10 /mavros/global_position/compass_hdg
```

```
/mavros/global_position/global
/mavros/global_position/local
/mavros/global_position/raw/fix
/mavros/global_position/raw/gps_vel
/mavros/global_position/rel_alt
/mavros/image/camera_image
/mavros/image/camera_image/compressed
/mavros/image/camera_image/compressed/parameter_descriptions
/mavros/image/camera_image/compressed/parameter_updates
/mavros/imu/atm_pressure
/mavros/imu/data
/mavros/imu/data_raw
/mavros/imu/mag
/mavros/imu/temperature
/mavros/local_position/local
/mavros/manual_control/control
/mavros/mission/waypoints
/mavros/mocap/pose
/mavros/px4flow/ground_distance
/mavros/px4flow/raw/optical_flow_rad
/mavros/px4flow/temperature
/mavros/radio_status
/mavros/rc/in
/mavros/rc/out
/mavros/rc/override
/mavros/safety_area/set
/mavros/setpoint_accel/accel
/mavros/setpoint_attitude/att_throttle
/mavros/setpoint_attitude/attitude
/mavros/setpoint_attitude/cmd_vel
/mavros/setpoint_position/local
/mavros/setpoint_velocity/cmd_vel
/mavros/state
/mavros/time_reference
/mavros/vfr_hud
/mavros/vibration/raw/vibration
/mavros/vision_pose/pose
/mavros/vision_pose/pose_cov
/mavros/vision_speed/speed_vector
/mavros/wind_estimation
/rosout
/rosout_agg
/tf
/tf_static
```

# Bibliography


[1] Erle-rover. http://docs.erlerobotics.com/erle_robots/erle_rover. Accessed June 2020.

[2] Indoor flying. https://risc.readthedocs.io/1-indoor-flight.html. Accessed August 2020.

[3] Offboard control. https://dev.px4.io/v1.9.0/en/ros/offboard_control.html. Accessed August 2020.

[4] M. Addel-Geliel, S. Zakzouk, and M. El Sengaby. Application of model based fault detection for an industrial boiler. In *2012 20th Mediterranean Conference on Control Automation (MED)*, pages 98–103, 2012.

[5] C. Angeli. Diagnostic expert systems: From expert's knowledge to real-time systems. *Advanced Knowledge Based Systems: Model, Applications and Research*, 1, 2010.

[6] Quick Start Guide: Getting Started. https://v22.wiki.optitrack.com/index.php?title=Quick_Start_Guide:_Getting_Started. Accessed: March 8th, 2020.

[7] Guobin Chang. Robust kalman filtering based on mahalanobis distance as outlier judging criterion. *Journal of Geodesy*, 88(4):391–401, jan 2014.

[8] J. Chen, Y. Zhou, and H. Liang. Effects of acc and cacc vehicles on traffic flow based on an improved variable time headway spacing strategy. *IET Intelligent Transport Systems*, 13(9):1365–1373, 2019.

[9] E. M. Cimpoesu, B. D. Ciubotaru, and D. Stefanoiu. Fault detection and diagnosis using parameter estimation with recursive least squares. In *2013 19th International Conference on Control Systems and Computer Science*, pages 18–23, 2013.

[10] Swaroop Darbha, J.K. Hedrick, C. Chien, and Petros Ioannou. A comparision of spacing and headway control laws for automatically controlled vehicles1. *Vehicle System Dynamics*, 23:597–625, 11 1994.

[11] Swaroop Darbha and K.R. Rajagopal. Intelligent cruise control systems and traffic flow stability. *Transportation Research Part C: Emerging Technologies*, 7(6):329 – 352, 1999.

[12] K. C. Dey, L. Yan, X. Wang, Y. Wang, H. Shen, M. Chowdhury, L. Yu, C. Qiu, and V. Soundararaj. A review of communication, driver characteristics, and controls aspects of cooperative adaptive cruise control (cacc). *IEEE Transactions on Intelligent Transportation Systems*, 17(2):491–509, 2016.

[13] I. Fagarasan and S. S. Iliescu. Parity equations for fault detection and isolation. In *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, volume 1, pages 99–103, 2008.

[14] Shuo Feng, Yi Zhang, Shengbo Li, Zhong Cao, Henry Liu, and Li Li. String stability for vehicular platoon control: Definitions and analysis methods. *Annual Reviews in Control*, 03 2019.

[15] Rolf Isermann. Model-based fault detection and diagnosis - status and applications. *IFAC Proceedings Volumes*, 37(6):49 – 60, 2004. 16th IFAC Symposium on Automatic Control in Aerospace 2004, Saint-Petersburg, Russia, 14-18 June 2004.

[16] Vishrut Jain. Longitudinal control for heterogeneous vehicle platooning with uncertain dynamics: Vehicle platooning. Master's thesis, TU Delft, 2019.

[17] D. Jia, K. Lu, J. Wang, X. Zhang, and X. Shen. A survey on platoon-based vehicular cyber-physical systems. *IEEE Communications Surveys Tutorials*, 18(1):263–284, 2016.

[18] R. E. Kalman. A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, page 35, 1960.

[19] Twan Keijzer and Riccardo Ferrari. A sliding mode observer approach for attack detection and estimation in autonomous vehicle platoons using event triggered communication. *CDC2019*, 2019.

[20] Robert Kissell and Jim Poserina. Chapter 4 - Advanced Math and Statistics. In *Optimal Sports Math, Statistics, and Fantasy*, pages 103 – 135. Academic Press, 2017.

[21] Frank L. Lewis. *Applied Optimal Control and Estimation: Digital Design and Implementation*. Prentice-Hall, 1992.

[22] K. Li, W. Ni, E. Tovar, and M. Guizani. Lcd: Low latency command dissemination for a platoon of vehicles. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, 2018.

[23] H. Mahama and Y. Chen. Lane based platoon control of homogeneous platoons. In *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, pages 1–9, 2019.

[24] Dubravko Miljković. Fault detection methods: A literature survey. pages 750–755, 05 2011.

[25] Optitrack for Robotics. https://optitrack.com/motion-capture-robotics/. Accessed: March 2nd, 2020.

[26] G. J. L. Naus, R. P. A. Vugts, J. Ploeg, M. J. G. van de Molengraft, and M. Steinbuch. String-stable cacc design and experimental validation: A frequency-domain approach. *IEEE Transactions on Vehicular Technology*, 59(9):4268–4279, 2010.

[27] Jason M. O'Kane. *A Gentle Introduction to ROS*. Independently published, October 2013. Available at http://www.cse.sc.edu/~jokane/agitr/.

[28] R.J. Patton and J. Chen. A review of parity space approaches to fault diagnosis. *IFAC Proceedings Volumes*, 24(6):65 – 81, 1991. IFAC/IMACS Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS'91), Baden-Baden, Germany, 10-13 September 1991.

[29] J. Ploeg, B. T. M. Scheepers, E. van Nunen, N. van de Wouw, and H. Nijmeijer. Design and experimental evaluation of cooperative adaptive cruise control. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 260–265, 2011.

[30] O. Sawade and I. Radusch. Survey and classification of cooperative automated driver assistance systems. In *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*, pages 1–5, 2015.

[31] Steven Shladover, Christopher Nowakowski, Xiao-Yun Lu, and Robert Ferlis. Cooperative adaptive cruise control (cacc) definitions and operating concepts. 01 2015.

[32] Richa Singhal and Rakesh Rana. Chi-square test and its application in hypothesis testing. *Journal of the Practice of Cardiovascular Sciences*, 1, 01 2015.

[33] S. S. Stankovic, M. J. Stanojevic, and D. D. Siljak. Decentralized overlapping control of a platoon of vehicles. *IEEE Transactions on Control Systems Technology*, 8(5):816–832, 2000.

[34] D. Swaroop and J. K. Hedrick. String stability of interconnected systems. In *Proceedings of 1995 American Control Conference - ACC'95*, volume 3, pages 1806–1810 vol.3, 1995.

[35] A. Vahidi and A. Eskandarian. Research advances in intelligent collision avoidance and adaptive cruise control. *IEEE Transactions on Intelligent Transportation Systems*, 4(3):143–153, 2003.

[36] B. van Arem, C. J. G. van Driel, and R. Visser. The impact of cooperative adaptive cruise control on traffic-flow characteristics. *IEEE Transactions on Intelligent Transportation Systems*, 7(4):429–436, 2006.

[37] E. van Nunen, J. Ploeg, A. M. Medina, and H. Nijmeijer. Fault tolerancy in cooperative adaptive cruise control. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1184–1189, 2013.

[38] Michel Verhaegen and Vincent Verdult. *Filtering and System Identification: A Least Squares Approach*. Cambridge University Press, 2007.

[39] Zhiyi Zhang and Douglas Shafer. *Introductory Statistics*. 01 2014.

[40] Zhipeng Wang, Zili Wang, Laifa Tao, and Jian Ma. Fault diagnosis for bearing based on mahalanobis-taguchi system. In *Proceedings of the IEEE 2012 Prognostics and System Health Management Conference (PHM-2012 Beijing)*, pages 1–5, 2012.

# Glossary

## List of Acronyms

| | |
|---|---|
| **TU Delft** | Delft University of Technology |
| **RC** | Radio Control |
| **NERDlab** | Networked Embedded Robotics DCSC lab |
| **ROS** | Robot Operating System |
| **ADAS** | Advanced Driver-Assistance Systems |
| **ACC** | Adaptive Cruise Control |
| **CACC** | Cooperative Adaptive Cruise Control |
| **V2V** | Vehicle-to-Vehicle |
| **MoCap** | Motion Capture |
| **KF** | Kalman Filter |
| **LTI** | Linear Time-Invariant |
| **MD** | Mahalanobis Distance |
| **ESC** | Electronic Speed Controller |
| **ZOH** | Zero-Order Hold |
| **PPM** | Pulse Position Modulation |
| **RPi** | Raspberry Pi |
| **GS** | Ground Station |
| **MAVLink** | Micro Air Vehicle Link |
| **IP** | Internet Protocol |
| **SSH** | Secure Shell |
| **ITS** | Intelligent Transportation Systems |