Fault Diagnosis of Self-Localization in Autonomous Vehicles Using a Model-Based Approach

The WEpods Case

R.V. Kossen

01-05-2019

TU Delft

Fault Diagnosis of Self-Localization in Autonomous Vehicles Using a Model-Based Approach

The WEpods Case

by

R.V. Kossen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday May 7, 2019 at 13:15.

Student number:	4219686	
Thesis committee:	Dr. R. M. G. Ferrari,	Delft University of Technology
	Ir. F. Gaisser,	Robot Robots Company
	Prof. Dr. D. M. Gavrila,	Delft University of Technology
	Dr. Ir. W. Mugge,	Delft University of Technology

This thesis is confidential and cannot be made public until May 1, 2020.

Abstract

Autonomous driving is a development that has gained a lot of attention lately, because it can lead to major improvements in the mobility sector. First of all, safety will be improved because human errors are eliminated from the driving process. Also, the effects on the environment will be beneficial in terms of using optimal driving conditions and the outlook on car sharing possibilities. Thirdly, mobility will be improved for those that are normally not capable of participating in traffic: (highly) disabled and elderly people for example. Much research is conducted and even the commercial availability of (highly) automated cars has increased. One example of a research project that aims to develop vehicles that are capable of reaching the highest level of autonomy in driving, is the WEpods project. The goal of this research is in line with this aim, having the thesis objective defined as follows: *let the WEpods continue driving in autonomous mode more often than is currently the case.*

The WEpod shuttles are not yet completely able to drive autonomously due to their inability to handle unexpected behavior (terminology: faults). Currently, such faults need to be detected and solved by a steward, who will manually initiate a safe stop if necessary. The localization module, which is responsible for localizing the vehicle on a map, sometimes generates unreliable location estimates. This poses two challenges. First, the fact that there is a mismatch between reality and the sensor outcomes of the localization module that needs to be detected. Second, the question of how to prevent the system from showing behavior that is different from what is desired (terminology: failure) in case such a fault is present (terminology: fault tolerant contol). Fault tolerant control can be performed in either a passive or an active manner. The passive approach ensures that either the faults are prevented or the system is able to mitigate them by anticipation in the design. The approach evolves from passive to active fault tolerant control when an on-line adaptation of the system control is made. For applications in autonomous driving, it is apparent that it is important to handle not only anticipated faults, but also to be able to deal with unexpected faults in an on-line manner. This on-line fault tolerant control approach involves two fault diagnosis steps that lead to solving the first challenge: detection and isolation.

A so-called model-based fault diagnosis approach turned out to be most suitable, as it has been used for similar applications in the past. However, a model-based fault diagnosis approach has not yet been implemented for detecting and isolating faults in a localization module of autonomous driving, indicating the scientific relevance of this research. In the model-based approach, kinematic and dynamic equations of the research vehicle (WEpod) are used to build a computational model. This model is then subjected to an observer, that is able to compare the model outcomes with the actual measurements in an off-line way. A residual is drawn up by taking the difference between the model outcomes and the measurements. A threshold is computed based on noise on the measurements to compare the residual with. When the residual exceeds the threshold, an alarm is raised. This way, the system itself has been enabled to detect faults when they occur internally.

The research can be extended by an isolation step, that is able to determine which of the four different sensors within the localization module is faulty. The measurements of all these sources are inspected and the according residuals are computed on-line. Then, multiple observers analyze this and raise a message when and which of the sensors should be excluded from the fusion for computing the localization.

In order to test the fault diagnosis algorithm, real data logs from the WEpods are used. Most of the logs come with a description of the behavior during driving along the test track. Several logs that had a clear problem profile for known sensors (so no isolation was needed) were chosen, so the plugin was tested on its ability of detecting faults in that specific sensor. It is shown that the developed algorithm is indeed able to detect faults.

Based on these achievements, a suggestion can be given for updating the control and therewith tackling the second challenge. Actual inclusion of this update would make the vehicle active fault tolerant for its localization module, but this is left outside of the scope of this master's thesis.

Inclusion of the suggested fault diagnosis approach in an on-line manner into the system is a big step towards fully autonomous driving of the WEpods, and therefore the goal of this research is met.

Contents

List of Tables

List of Figures

1 Introduction

The research presented in this report is conducted in the context of the master track BioMechanical Design (specialization BioRobotics) in the major Mechanical Engineering at the Technical University of Delft. It has been exerted in collaboration with Robot Robots Company (RRC). This chapter aims to introduce the reader into the field of autonomous driving and the scientific relevance of introducing fault tolerant control in this domain (which is the thesis objective). At the end, an overview of the structure of this report is presented.

1.1. EU Interregional Automated Transport project (I-AT)

RRC is participating in an infrastructural initiative from the European Union called the I-AT: Interregional Automated Transport [33]. In this project, various (government funded) companies work together with the aim to improve the transportation facilities in the Dutch-German border area: provinces Gelderland, Brabant and Limburg in the Netherlands and bundesland Nordrhein-Westfalen in Germany. One of the improvements is the development of electrical shuttle buses that are highly automated, called: the WEpods. The exact partners involved are shown in Appendix A. The role of RRC is to provide the software that enables the vehicle to drive autonomously. Currently two WEpods (identified as *WElly* and *WUrbie* under the name of *WEasy*) are operational at Weeze airport and are capable of transporting up to six (6) people under a maximum driving speed of 25 km/h. Recently, the WEpods autonomous driving project got extended with the development of a third vehicle (identified as *Mission*). This new vehicle is ought to drive up to 50 km/h and transport up to sixteen (16) people. The launch of the *Mission* is planned for spring 2019 and will be tested on the roadways between Aachen (Germany) and Vaals (the Netherlands).

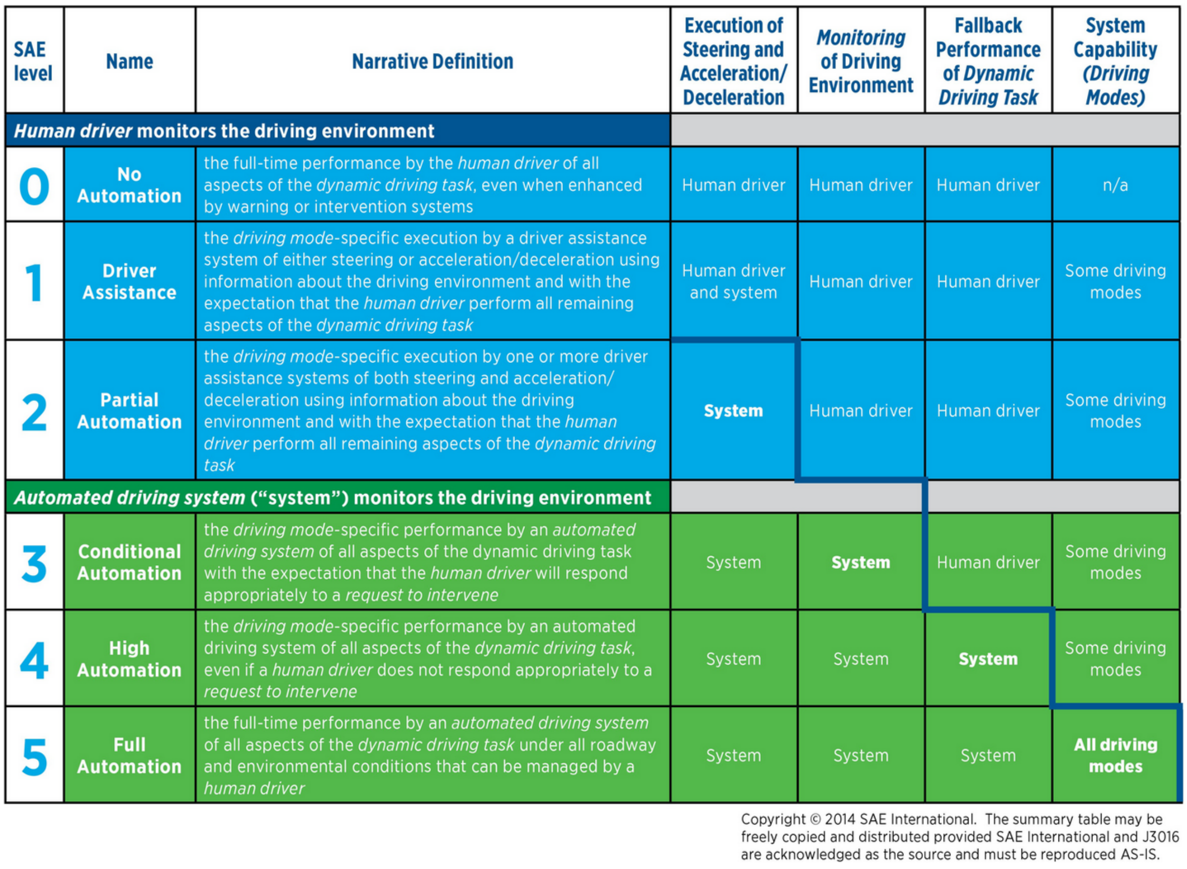
1.2. Classification levels in autonomous driving

The ultimate goal is to develop the WEpods towards level 5 of automation for on-road motor vehicles, as drawn up by the Society of Automotive Engineering (SAE) [25], shown in Figure 1.1. A short introduction into these levels (retrieved from [21]) is presented below:

- Level 0 indicates no automation, driver warning systems could be classified as level 0, because they strictly provide the driver with information, but do not interfere with the control of the vehicle in any kind of way.
- Level 1 contains driver assistance systems that perform control in one direction, literally either the longitudinal (along the vehicle) or lateral (sideways) direction. Examples are: (Adaptive) Cruise Control (A)CC and lane keeping assistance.
- Level 2 includes control enhancements in both longitudinal and lateral direction simultaneously. Until this level of automation, the human driver is still the one that monitors the driving environment and only some driving modes profit from these systems.
- From level 3 and higher, the system is capable of monitoring the driving environment and selecting an action accordingly, thus the subclass of higher automated driving is entered. However, for level 3, the human driver needs to remain within the loop as the responsibility for the fallback performance always

lies with him/her after a take-over request has been carried out by the system. Also the operational domain is delimited. For example a traffic jam assisting system, which is capable of carrying out the driving task on the highway below speeds of 60km/h [24]. Also self-parking systems can be included within this category.

- Level 4 does not require continuous attention of the human driver and is capable of driving fully autonomously within a certain domain (for example: transporting luggage at the airport). It could request the driver to take over control in certain situations, but when the system notices that the human is unable to do so timely, it is able to come to a safe stop itself.
- Level 5 finally covers all ranges of design domains, so from specially designed roads with only a certain type of road users, till urban traffic in which even dogs can intermingle in traffic. The car is now able to drive completely autonomously and does not need a human driver to interfere with the system for any driving mode. This is the ultimate goal for car manufacturers that aim for autonomous driving in the future.



SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	No Automation	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	Conditional Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	High Automation	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	Full Automation	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Figure 1.1: Levels of automation defined by Society of Automotive Engineers (SAE), reproduced AS-IS from SAE International, J3016.

The state-of-the art, however, is that a human operator is involved in the driving task as well that can take over in case something is about to go wrong. The Dutch law also still prescribes that a human driver should be present in any vehicle that is driving on the (public) road. The safety measures that need to be met in order to get a license from the RDW (Dienst Wegverkeer) for driving on the public road got restricted lately and also require research vehicles to meet high standards (as presented in ISO26262 [32]).

1.3. Fault Tolerant Control

An autonomous vehicle of level 5 should be able to fully perceive its environment and other road users, determine what actions need to be taken to navigate safely through urban road conditions and reach its destination without any interference of a human being. Therefore, it is important to design the vehicle as such that it won't fail. Of course, it is unthinkable for such a complex system existing of many many components to include all possible scenarios in which something goes wrong and prevent the system from emerging in a failure. At one point in time, of course, some components will inevitably wear out. Of course it is possible to try and achieve a longer lifetime by designing as robust as possible, by duplicating certain critical components in case one fails (called: redundancy). A large limiting factor here is of course budget. Also, it is not very environmentally friendly to have many components aboard (hence increased weight) that are not used in the nominal operation of the vehicle. The measures that can be taken during the design phase for preventing components from failing have therefore an upper limit. Apart from wear-out, there are many other factors that can lead to faults. Faults are defined as the occurrence of unexpected behavior that might, if they have an active character, lead to a complete system failure, in which the objected behavior cannot be met any longer [21]. Examples of 'faults' are: certain weather conditions like sunlight, fog or rain that influence the outcome of the sensors in the vehicle. But also malfunctioning of the hardware or other external influences can have an unexpected effect on the vehicle: less grip due to an icy road or a strong wind for example. Right now, the operator is asked to take over control in case some of these faults and/or expressions of unexpected behavior are detected. A fully autonomous system should be able to handle all these types of faults itself in order to prevent it from resulting in a complete failure.

The area of research which is called 'fault tolerant control' is tailored for that. It is assumed, as explained above, that a system contains (unexpected) faults. On the contrary of trying to design a system in such way that is resistant against all these type of faults, solutions are sought for a system in which faults occur. Various methods and required steps for this so-called 'fault tolerant control' methodology are to be found in the literature review [21]. Several aspects will be taken out of there to deploy further within this master's thesis. In short: in the first place, the outing of unexpected behavior needs to be detected. The next step is to find the origin of the detected fault (isolation) and finally the effects of the fault on the system will be predicated (estimation). The last two steps combined are called 'fault diagnosis'. This way, measures can be taken in order to mitigate the faults or adapt in another way such that the overall system behavior does not differ too much from the objected behavior.

1.4. Research goal and report structure

The application of the fault tolerant control methodoly onto the WEpods-project is the goal of this graduation thesis. From the literature review as found in [21], it became apparent that this approach, which knows a history for mainly industrial application, has not been used that often yet for the application of autonomous driving. This stresses the scientific relevance of the work presented here as it brings together two relevant fields. The aim that is to be met by bringing the field of autonomous driving together with the field of fault tolerant control, is to: *let the WEpods continue driving in autonomous mode more often than is currently the case.* More specifically, to enable the vehicle to continue driving autonomously *in case of certain faults.* These 'certain' faults are identified in Chapter 2, where it is researched which areas are most pending for receiving correction and therewith the actual problem is formulated. This is done by first providing a detailed description of the WEpods, both for the older pods from *WEasy* and for the newer bus, the *Mission.* In Chapter 3, the proper fault detection strategy is chosen and the methods are described. Chapter 4 gives an overview of the experiments that were designed and provides the results thereof. Finally, Chapter 5 concludes the research and stresses a couple of recommendations for future implementation.

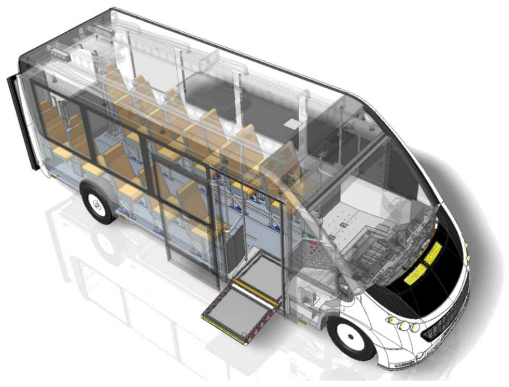
2
System description and problem formulation

Since November 2015, as part of the I-AT project, two WEpod shuttle buses have been taken into operation and a third (bigger and faster) one is on its way. First, a system description including hardware, software, behavior objectives and the incorporation of fault tolerance is included for this new *WEmi* bus and then a similar system description is presented for the already operational *WEasy* shuttles. The latter description contains more details regarding the model of the system, because these pods will be the main focus of the research. The reason for that is that these buses have been already in operation for a while, so test results are available. The new bus is at the time of writing (March 2019) still under construction, therefore less freedom in performing research is guaranteed.

2.1. WEmi - WEpods shuttle *Mission*

Before it is possible to evaluate where the possibilities lie for improvement in case any fault occurs, it is important to obtain understanding of how the nominal model of the system functions. First, the hardware of the system with its relevant sensors will be assessed briefly, then some schemes about the control architecture will be presented, along with an identification of all of its modules. A review of the driving modes is given after that. Finally, the following fault tolerance measures: FTA (Fault Tree Analysis), HARA (Hazard Assessment and Risk Analysis), FMEA (Failure Mode Effects Analysis) and a watchdog are discussed. In Figure 2.1, the designs of the exterior (left) and interior (right) of the *Mission* are shown.

(a) Exterior



(b) Interior

Figure 2.1: WEpod Mission design. Retrieved from: [20].

2.1.1. Hardware

The hardware of the *Mission* bus is provided by an external company, called UMS. It is out of scope to describe all those hardware components. For the purpose of making the WEpod fault tolerant, the focus will remain on the software-side, on which RRC is working. In Figure 2.2, the diagram of the hardware which is required to let the software run, that will be incorporated in the *Mission* (November 2018 version) is shown.

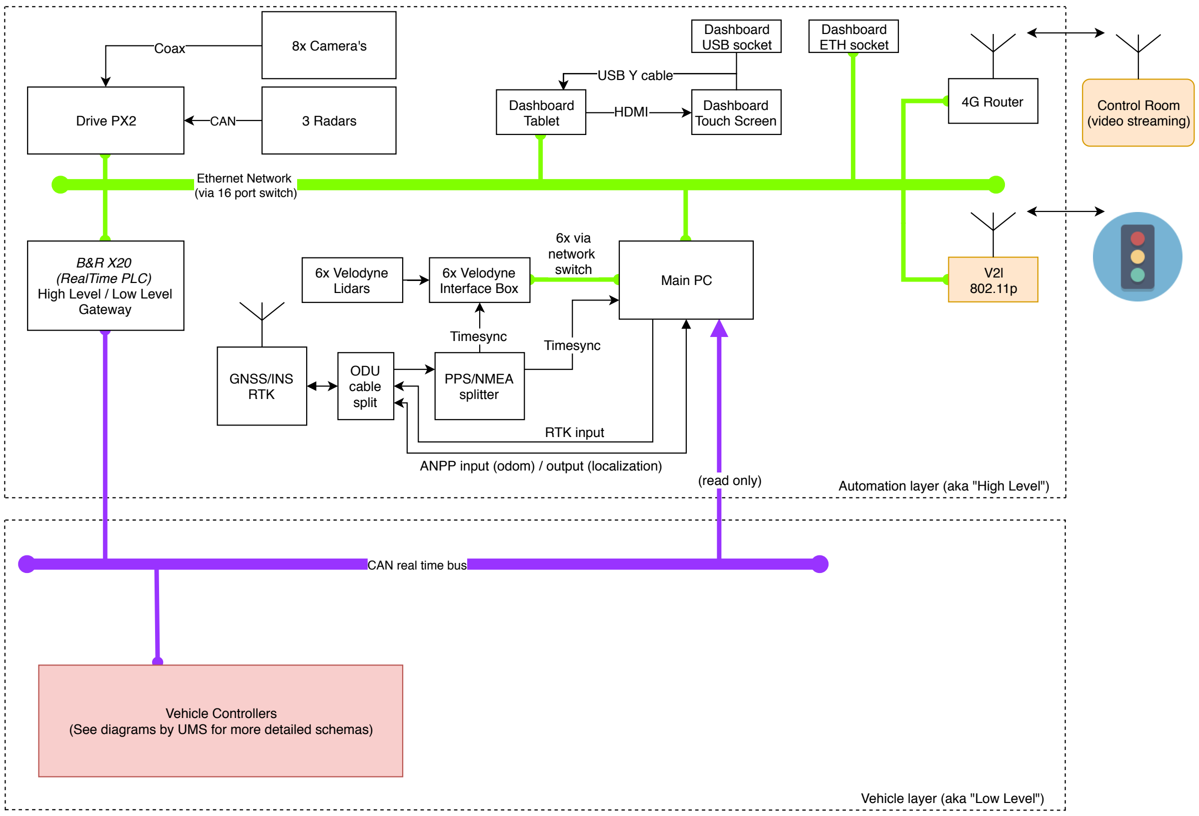

8x Camera's
Coax
Drive PX2
CAN
3 Radars
Dashboard USB socket
Dashboard ETH socket
USB Y cable
Dashboard Tablet
HDMI
Dashboard Touch Screen
4G Router
Control Room (video streaming)
Ethernet Network (via 16 port switch)
B&R X20 (RealTime PLC) High Level / Low Level Gateway
6x Velodyne Lidars
6x Velodyne Interface Box
6x via network switch
Main PC
V2I 802.11p
GNSS/INS RTK
ODU cable split
PPS/NMEA splitter
Timesync
Timesync
RTK input
ANPP input (odom) / output (localization)
(read only)
Automation layer (aka "High Level")
CAN real time bus
Vehicle Controllers (See diagrams by UMS for more detailed schemas)
Vehicle layer (aka "Low Level")

Figure 2.2: Hardware diagram of the WEpod Mission.

The software that is developed by RRC, is running on the Drive PX that has two separate Linux systems running, connected by the Main PC and finally there is one Dashboard Tablet present in the vehicle, which also runs Linux. The hardware that is most relevant in this research, are the sensors. First, the bus contains the same sensors that a modern car has: ABS (anti-lock braking system), ESP (electronic stability program) and some that are dedicated to measuring temperature, amount of light in the surrounding, etc. To prepare the bus for the incorporation of autonomous driving functions, RRC added the following sensors:

- Steering wheel change rate sensor;
- Steering wheel angle sensor;
- Motor rpm encoder;
- Breaking pressure sensor (cylinder);
- WiFi (for communicating V2V: vehicle-to-vehicle and V2I: vehicle-to-infrastructure);
- Eight cameras (as shown in Figure 2.2);
- Three radars (as shown in Figure 2.2);
- Six lidars (as shown in Figure 2.2);
- GNSS/INS (Global Navigation Satellite System/ Inertial Navigation System) RTK (Real Time Kinematic)-unit (as shown in Figure 2.2), including: odometer, IMU (Inertial Measurement Unit) and GPS (Global Positioning System) [19].

Focus is placed on the latter four types of sensors, because these are used in several modules for autonomous driving which are discussed in the upcoming sections. The combination of the first three are part of the the object detection module, whereas the latter two are used by the localization module.

2.1.2. Architecture

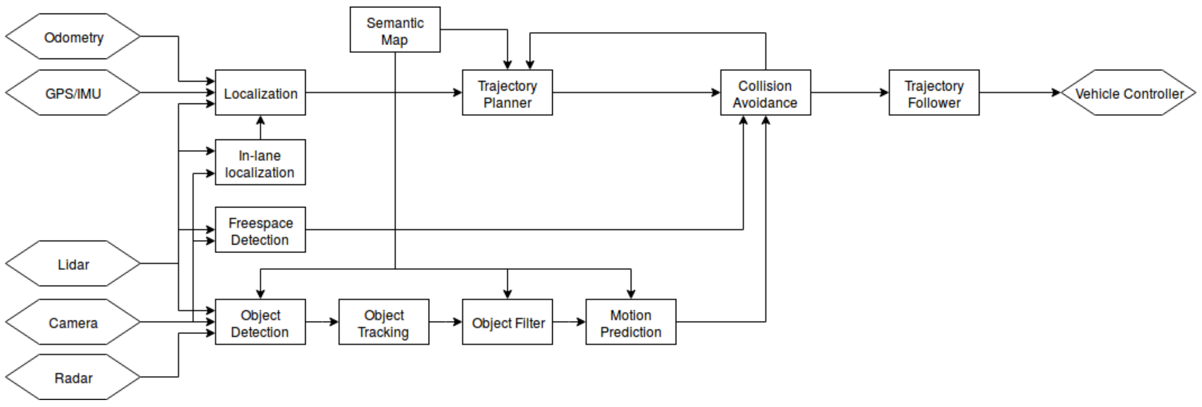


Figure 2.3: Control architecture of the WEpod Mission. Retrieved form: [20].

In Figure 2.3, the control architecture of the *Mission* is depicted. On the left side, the sensors are marked in the hexagon-shaped blocks. There is one odometry sensor and one combined GPS/IMU unit present in the system. There are six lidars (laserscanners), three radars and even eight cameras incorporated in the system. On the right side, another hexagon-shaped block is visible. This is the vehicle controller, which converts information that is flowing in from the trajectory follower, into executable control laws. Here, the desired speed and steering wheel angles are calculated and forwarded to the actuators. In the middle, there are many rectangular blocks to be found, which all (except for the semantic map), have their own computations and forward new information. The different modules within the control architecture will be briefly explained below.

Object detection module

In the bottom left, the object detection module is indicated. It performs its detection through software algorithms that receive input from various sensors. The eight cameras, three radars and six lidars provide information about the surroundings. They are able to 'see' objects and/or obstacles around. Based on a machine-learning algorithm, that compares the input to a large database, containing all sorts of 'road-objects', it identifies what objects are around and indicates whether these objects are ought to be static or dynamic.

Motion prediction module

In case of dynamic objects, the motion prediction module becomes of utmost importance, as it is aimed to predict the behavior in terms of motion of these objects. Think: other road users like cars, pedestrians, bicyclists, even pet animals, basically anything that moves around the car, so an assessment could be made on whether the dynamic 'object' will come into the range of motion of the WEpod.

Localization module

The localization module has the objective to identify the location (also named: ego-position) of the WEpod Mission in the world. It combines information from the GPS-unit (odometer, IMU, gyroscope and GPS) with the information coming in from the lidars.

Trajectory planning module

Once the information about the ego-position of the vehicle is known, it can be compared to the desired route (coming from an higher level as input). It uses the semantic map (provided by an external company, IBEO) to plan a trajectory in order to reach the goal. The trajectory planner can be triggered again once the collision avoidance module identifies dangers on the road ahead, as explained in the next subsection.

Collision avoidance module

The collision avoidance module brings together various outcomes of other modules. It compares the planned trajectory (resulting from the trajectory planning module) to the predicted motion of the surrounding objects. If a dynamic object seems to be interfering with the projected trajectory, an update of this trajectory should be made, in order to prevent any collisions. Once this is done, the newly planned trajectory could be sent towards the actual trajectory following module that converts the information into motor torques, etc. such that the vehicle controller can actually perform the appropriate control.

Concise version architecture

A more concise version of the scheme presented in Figure 2.3 is set up as follows in Figure 2.4. The in-lane localization and freespace detection are left out for now, because the main focus of the master's thesis will be on the other components that have the necessity of implementation. Also, the trajectory planner and trajectory follower could be regarded as one entity, such that the collision avoidance becomes the feedback loop. The green entries on the left side indicate the sensors, the light yellow blocks are the modules. Light blue indicates external entries, whereas the darker yellow blocks are the higher-level software modules. On the right, in light purple, the final control command is indicated that will be send to the vehicle.

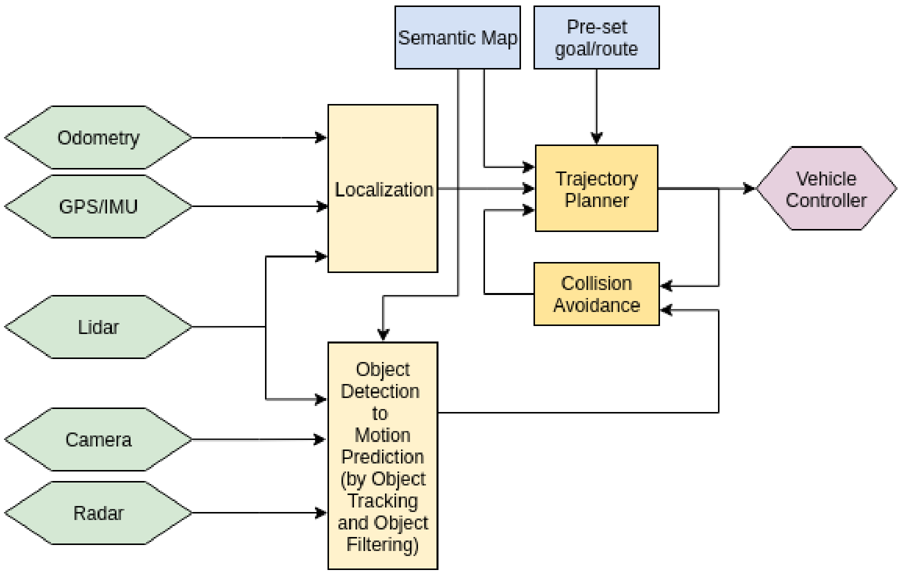


Figure 2.4: Simplified control architecture of the WEpod Mission.

2.1.3. Driving modes

The WEpod *Mission* will have three driving modes: manual, semi-autonomous and autonomous, described in each of the following items.

- **Manual**
  Only the driver is involved in the driving process and no extra, aside from the already present features like cruise control etc., automation is incorporated. This is indicated as SAE level 0 of automation as shown in the introduction.

- **Semi-autonomous**
  The driver remains attentive during the driving process and executes the speed control by having complete control over the brakes and gas pedal, therefore he/she is in charge of the longitudinal control of the vehicle. The autonomy is restricted to following the planned path, including the modules 'localization' and 'trajectory planner'. The perception of the environment is done by the driver. The perception of the environment is done by the object perception stack and actions are taken by the navigation stack accordingly. However, for complex manoeuvres such as lane change and overtaking, the driver needs to take over driving control. After the manoeuvre has been performed, the vehicle is given control again. This is indicated as SAE level 3 of automation as shown in the introduction. The lateral control architecture in case of autonomous driving mode, is shown in Figure 2.5.

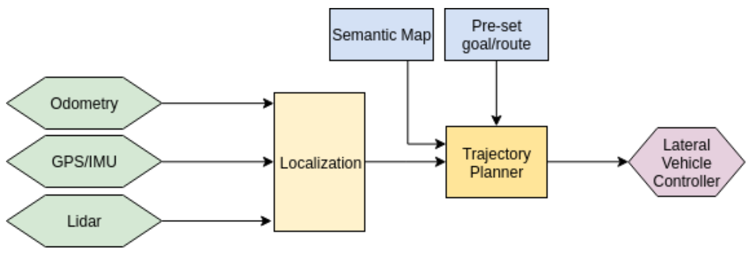


Figure 2.5: Control architecture for the semi-autonomous driving mode of the WEpod Mission.

- **Autonomous**
  In this driving mode, the vehicle is taking control of both its longitudinal and lateral control. It is able to fully perceives its surroundings and determine its behavior accordingly. All modules are incorporated: 'localization', 'trajectory planning', 'object detection', 'motion prediction' and 'collision avoidance'. The human driver is still able to take over the control of the vehicle by pressing the brakes and/or gas pedal and thereby switch to manual mode again. Since the bus is restricted to a maximum speed of 50km/h and only urban road conditions are regarded, this mode is indicated as SAE level 4 of automation as shown in the introduction.

2.1.4. Fault tolerance measures

In this subsection, some elaboration is made on the fault tolerance measures that have been included in the design already (note that these all have a passive nature). The following subsubsections are aimed, in a row, at identifying where faults can occur in each of the intermediate (control) steps by means of a Fault Tree Analysis (FTA), Hazard Assessment and Risk Analysis (HARA) and Failure Mode and Effect Analysis (FMEA). Then, the system that is designed to detect major issues in the software and disable the system when a failure is about to happen, called the *watchdog*, is presented.

Fault Tree Analysis (FTA)

When the overall system is regarded, a general fault tree analysis could be made which is feasible in all driving modes (manual, semi-autonomous, autonomous) and shown in Figure 2.6. The cloud-block represents control input that is sent to the vehicle. This could result from a manual manipulation of the gas pedal, brakes and/or steering wheel, but also from a control command that is drawn up by the autonomous modules. Therefore, this scheme can be used for all three driving modes. Fault tree analyses (FTAs) are performed for some of the WEpod modules and have been included in Appendix B.

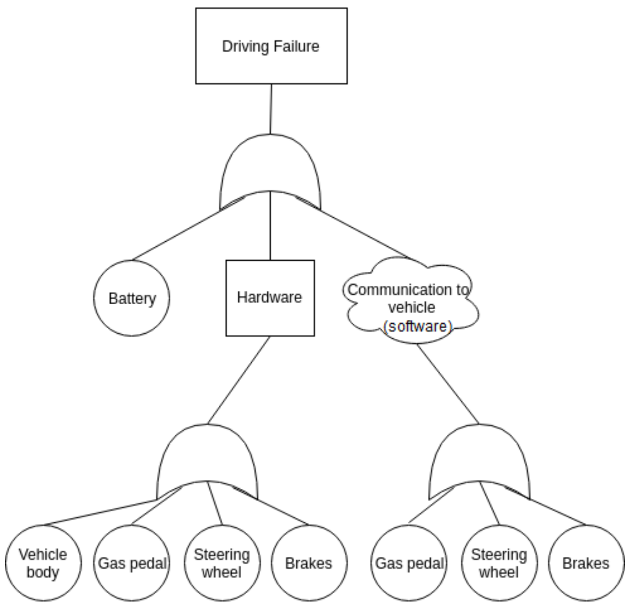
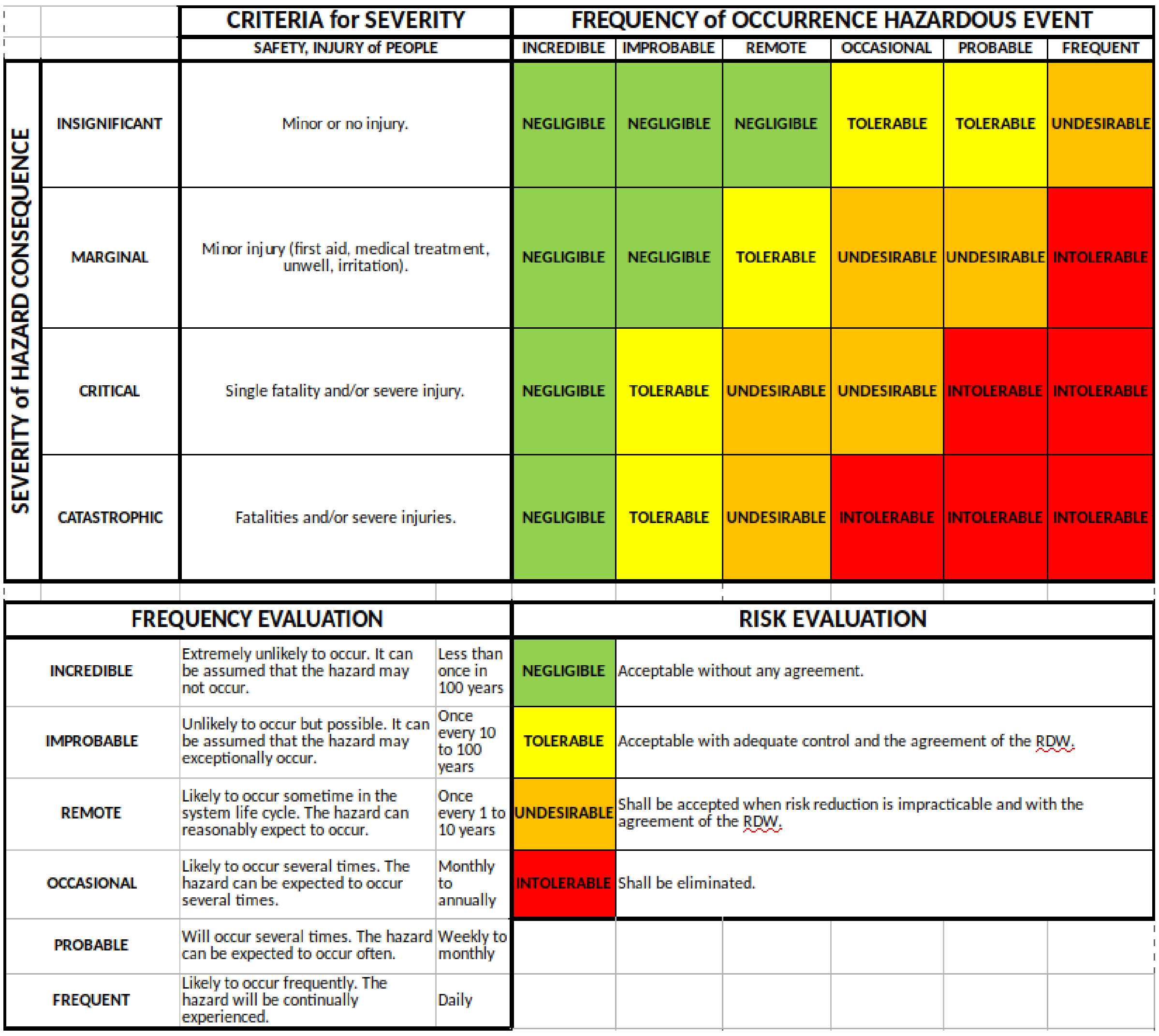


Figure 2.6: General FTA diagram WEpod.

Hazard Assessment and Risk Analysis (HARA)

The ISO26262 requires a broad hazard assesment and risk analysis, in which possible faults and failures are stated. It is a qualitative description of how faults effect the system. In a way it is therefore similar to the FMEA as described in [21], although it has no graphical representation. RRC used the following table (as presented in Figure 2.7) for possible severity and frequency of occurrence of hazardous events.



		CRITERIA for SEVERITY	FREQUENCY of OCCURRENCE HAZARDOUS EVENT					
		SAFETY, INJURY of PEOPLE	INCREDIBLE	IMPROBABLE	REMOTE	OCCASIONAL	PROBABLE	FREQUENT
SEVERITY of HAZARD CONSEQUENCE	INSIGNIFICANT	Minor or no injury.	NEGLIGIBLE	NEGLIGIBLE	NEGLIGIBLE	TOLERABLE	TOLERABLE	UNDESIRABLE
	MARGINAL	Minor injury (first aid, medical treatment, unwell, irritation).	NEGLIGIBLE	NEGLIGIBLE	TOLERABLE	UNDESIRABLE	UNDESIRABLE	INTOLERABLE
	CRITICAL	Single fatality and/or severe injury.	NEGLIGIBLE	TOLERABLE	UNDESIRABLE	UNDESIRABLE	INTOLERABLE	INTOLERABLE
	CATASTROPHIC	Fatalities and/or severe injuries.	NEGLIGIBLE	TOLERABLE	UNDESIRABLE	INTOLERABLE	INTOLERABLE	INTOLERABLE

FREQUENCY EVALUATION		
INCREDIBLE	Extremely unlikely to occur. It can be assumed that the hazard may not occur.	Less than once in 100 years
IMPROBABLE	Unlikely to occur but possible. It can be assumed that the hazard may exceptionally occur.	Once every 10 to 100 years
REMOTE	Likely to occur sometime in the system life cycle. The hazard can reasonably expect to occur.	Once every 1 to 10 years
OCCASIONAL	Likely to occur several times. The hazard can be expected to occur several times.	Monthly to annually
PROBABLE	Will occur several times. The hazard can be expected to occur often.	Weekly to monthly
FREQUENT	Likely to occur frequently. The hazard will be continually experienced.	Daily

RISK EVALUATION	
NEGLIGIBLE	Acceptable without any agreement.
TOLERABLE	Acceptable with adequate control and the agreement of the RDW.
UNDESIRABLE	Shall be accepted when risk reduction is impracticable and with the agreement of the RDW.
INTOLERABLE	Shall be eliminated.

Figure 2.7: HARA table WEpod.

Failure Mode Effects Analysis (FMEA)

A preliminary FMEA has been performed and represented in excel prior to the start of this project. Several subdivisions for evaluation have been made based on the functions that the vehicle is ought to exert, namely:

- Steering
- Braking
- Acceleration
- Base vehicle functions (hardware like: wiper, lights, horn, power supply, etc.)

A visual representation of the FMEA is made and included in Appendix C for the first category: steering. Similar representations and further elaboration of mathematics are left out now, for the sake of brevity, but can be set up along the same line of thought.

Watchdog

For controlling the performance of the software itself, a so-called 'Watchdog', as depicted in Figure 2.8, is designed for the WEpods Mission. This is aimed at checking the information flows coming from the Linux system and the lower-level modules (like object detection and localization, but also the sensors themselves) and makes sure all processes are running normally. It also checks whether the information coming from the modules matches the chosen driving mode (retrieved from the upper arrow). It forwards this information by means of sending a summary to the high level/low level gateway. This is the only loop through which the information resulting from the software, can reach the hardware of the system. This one gateway is very reliable on either passing or not passing information. The 'Process Manager'-block is responsible for starting and stopping the system only, it does not incorporate any higher-level thinking. The complete software (indicated with blue) is running in ROS and can fail as a whole, therefore the 'Watchdog watcher' is included that checks whether the software is running in the first place or if it might have experienced a crash.

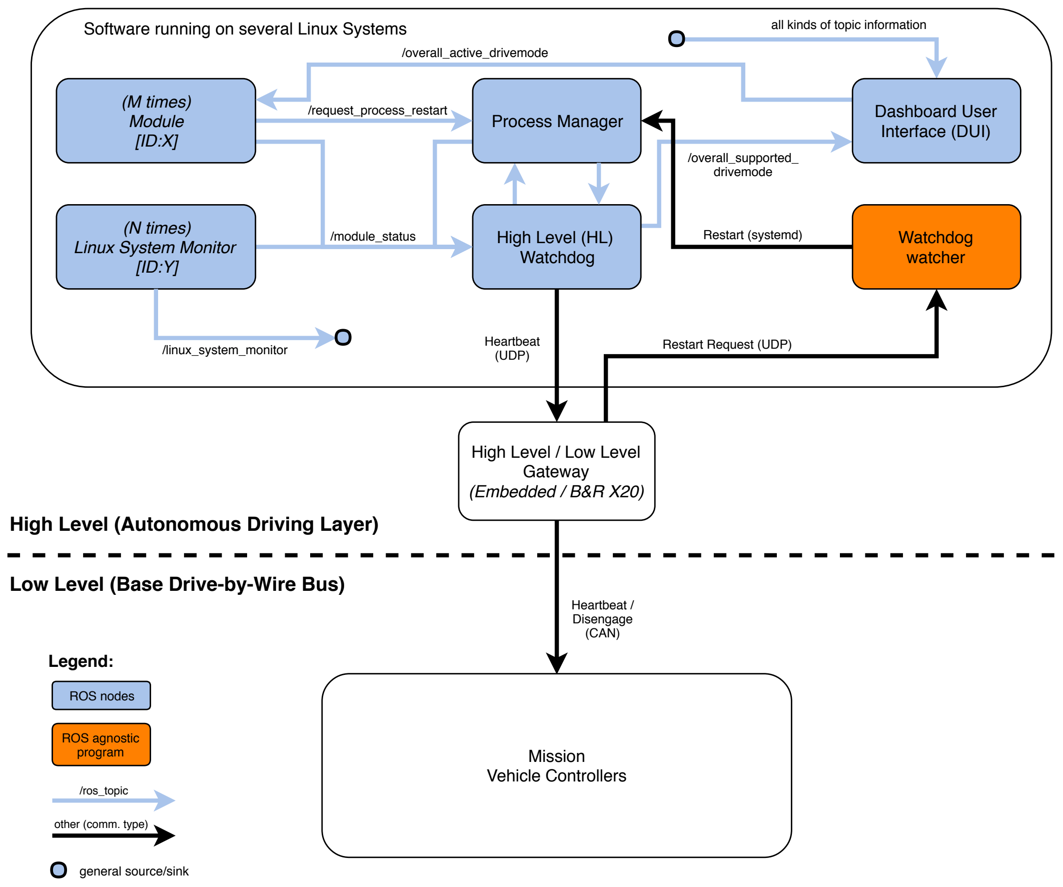
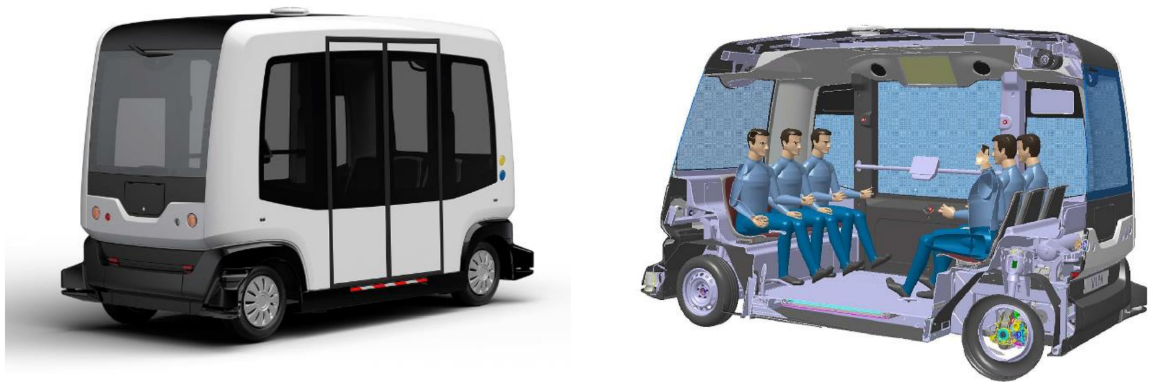

Software running on several Linux Systems
all kinds of topic information
/overall_active_drivemode
(M times) Module [ID:X]
/request_process_restart
Process Manager
Dashboard User Interface (DUI)
/overall_supported_ drivemode
(N times) Linux System Monitor [ID:Y]
/module_status
High Level (HL) Watchdog
Restart (systemd)
Watchdog watcher
/linux_system_monitor
Heartbeat (UDP)
Restart Request (UDP)
High Level / Low Level Gateway (Embedded / B&R X20)
High Level (Autonomous Driving Layer)
Low Level (Base Drive-by-Wire Bus)
Heartbeat / Disengage (CAN)
Legend:
ROS nodes
ROS agnostic program
/ros_topic
other (comm. type)
general source/sink
Mission Vehicle Controllers

Figure 2.8: Watchdog and process management diagram WEpod.

2.2. WEasy - WEpods shuttles *WElly* and *WURbie*

The design of the two shuttle buses that have been developed and taken into operation in 2015, is depicted in Figure 2.9.



(a) Exterior (b) Interior

Figure 2.9: WEasy shuttle design. Retrieved from: [23].

These are called *WElly* and *WURbie* and have an identical set-up (the only difference lies in the batteries: *WElly* has an additional 9.6 kWh and therefore an additional driving range of 30km - summing up to 80km in total). They have the maximum capability of transporting up to 6 persons under a maximum speed of 25 km/h. Because of the fact that the WEpod *Mission* is planned to be operational from spring 2019 onwards, the choice for this research is made to focus on the shuttles that are already in use. This way, test data can be used for the design of the diagnosis modules.

2.2.1. Hardware

The hardware of the WEasy shuttles has been provided by an external company, called EasyMile. It is out of scope to describe all those hardware components individually, the full description can be found in [23]. For the purpose of making the WEpods fault tolerant, the focus will remain on the software-side, on which RRC is working. The hardware included in the WEpod shuttles that is relevant for this software development is described in this section. A list, which is retrieved from [23], is made of the sensors and actuators included in the shuttles, along with their influence they exert on the physical way (i.e. the engineering versus the physical units will be described).

- Independent front and rear steering with electric actuation, turning radius 7m
- 4 Sick Lidars for obstacle detection; one on each corner
- For semi-automated control, a wired plug in manual control unit is provided in the vehicle, enabling: Control for speed, brakes (acceleration) and steering. Control switches for start, stop and 'call operator'.
- 9 Melexis High dynamic range cameras
- 9 airnozzles to clean the cameras
- 1 interior camera (for the operator)
- 9 Continental radars. Radars and cameras are solidly mounted in pairs on all sides of the vehicle at bumper height behind non obstructive plastic shielding that is transparent for the camera and radar
- 6 IBEO lidars
- Ultrasonics 9 sensors including controller
- Localization 2 GPS receivers are mounted on the roof at Y=0 at front and rear
- 1 INS is mounted nearest to the center of the floor

The sensors have their impact onto the rest of the system in the way as shown in Figure 2.10.

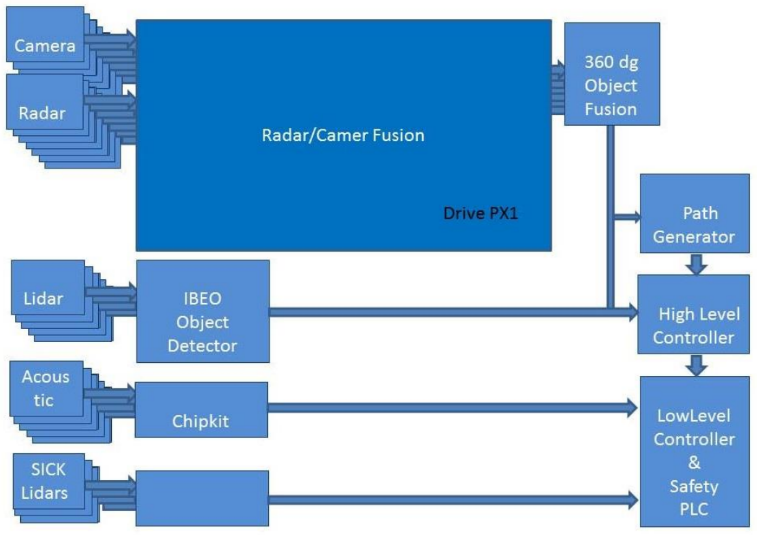


Figure 2.10: Sensor fusion. Retrieved from: [23].

The cameras and lidars are fused in the Drive PX1 and function as an input the the path planner (here: 'Path Generator') and the localization (here: coming together with the 'IBEO Object Detector'). The IBEO Object Detector is a piece of external software that outputs its localization based on the data it receives from the lidars, GPS and IMU. All is forwarded to the High Level Controller, where the behavior is determined and control inputs are given to the Low Level Controller.

In Figure 2.11, the diagram of the hardware which is required to let the software run, that is incorporated in the shuttles, is shown. Description of the interface as retrieved from [23]:
"From top to bottom the figure shows:

- Three NVIDIA DrivePX 1 computers that each contain two Tegra GPU chips, whereas each DrivePX connects 3 cameras each through a serial CS2 interface, and 3 radars all connected to a dedicated CAN bus. The DrivePX computers are connected to the BRIX computer of the Object Fusion Center for the fusion of the data of the 9 camera/radar pairs, as well as to the Central Vehicle computer for household tasks and time synchronization.

- The INS/GNSS system (a.k.a. Advanced Navigation System) that takes care of the vehicle positioning through GPS, RTK and odometry. It is connected with serial lines to the IBEO system to send vehicle pose information and to the High Level Controller to receive time synchronization signals and odometry data from the vehicle's CAN bus.

- The Central PC connects to all modules for the sake of time synchronization and watchdog signals to check the sanity of the system. In addition it uses information from all modules to perform a scenario analysis, e.g. for stopping at bus stop, traffic lights, interface with traffic lights, opening doors, etc. It also connects to a relay box to switch on alarm lights, wipers, etc. Finally it connects to the control room through a dedicated 4G connection. The vehicles front and rear cameras as well as the internal vehicle camera and intercom system are connected to the control room through this computer. Finally all system data logging is regulated by this computer.

- The entertainment system is a separate system that can be used by the passengers, e.g. to obtain touristic information along the route. The passengers can also use a WIFI service. With their app that connects to the server in the control room, passengers can make vehicle reservations and control the entertainment system.

- The IBEO system, which uses its 6 LIDARs and stored internal key-point map, while receiving pose information from the INS/GNNS system to output pose information to the BRIX computer of the fusion centre/path planner and the B&R computer for the high level controller via the ethernet bus.

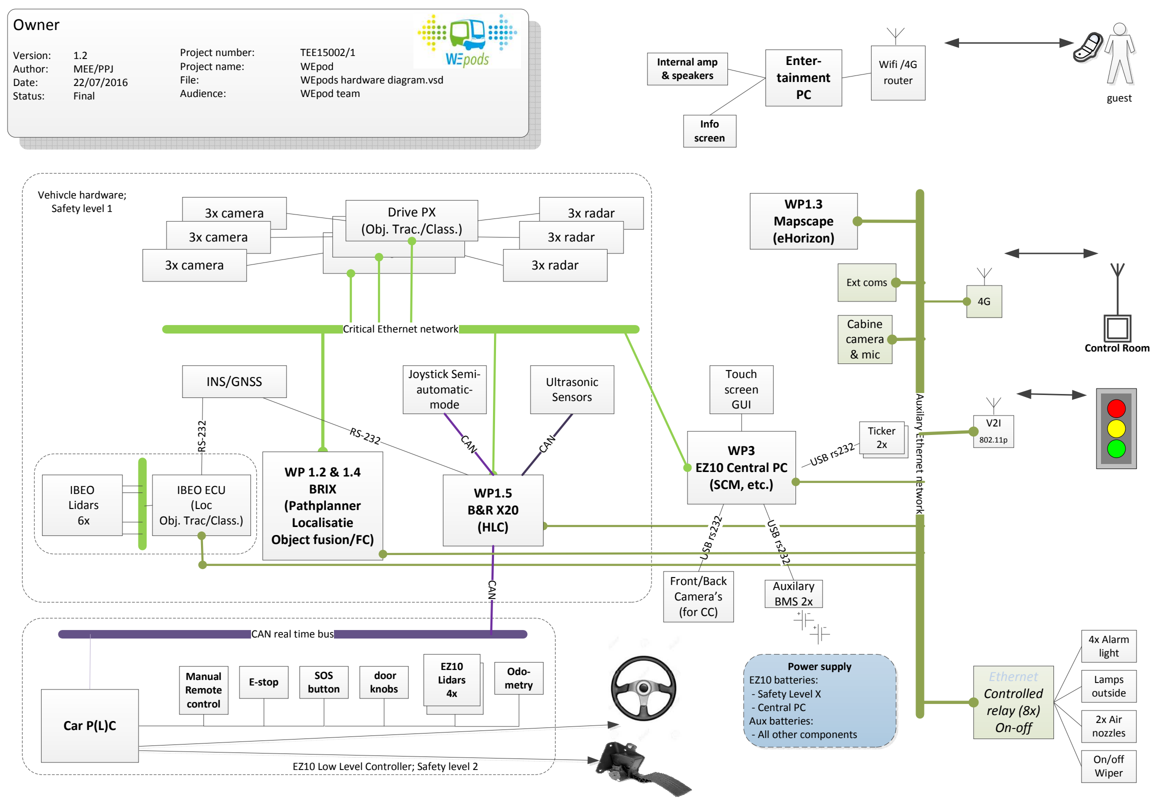


Figure 2.11: Hardware diagram of the WEasy shuttles. Retrieved from: [23].

- The BRIX computer which performs the fusion of data from the DrivePX systems. The data from 9 radar/camera pairs for objects around the vehicle is fused together and fused with object data from the IBEO system to a 360° environment around the vehicle in which all moving objects are known, including their velocity vector and class label. It also receives through the Ethernet bus vehicle position data from the IBEO system. The path planner consists of an off-line planning tool that uses eHorizon data reconstructed from a map of the environment to generate a course track from a certain route; e.g. the campus loop. This track can be driven and manually re-adjusted. When a satisfying track is found the same track can be driven (several times) with the IBEO system switched on in learning mode. It then learns the "virtual rail" that represents the true position on the road, even when the GPS/RTK fails. The path-planner generates a list of points that predict ahead the trajectory points that have to be followed by the High Level Controller. It sends this list over the Ethernet bus. The path planner can make use of objects and their velocity vector to reduce the velocity of the vehicle, and within constraints it can make an evasive motion, e.g. to overhaul parked cars.

- The B&R X20 system is a strict real-time system that houses the High Level Controller. It accesses the CAN bus to listen to messages from the Vehicle and sends velocity set-points to the Low Level Controller of the EZ10 vehicle. It receives overlapping trajectory point lists from the path planner in the BRIX, as well as velocity vectors of objects in the environment of the vehicle. It sends odometry data to the INS/GNSS over a serial line. It receives data from the 10 ultrasonic sensors around the vehicle through the Ethernet connection. It also receives information from a joystick that is used for the semi-automatic mode through a dedicated CAN bus. In this mode the velocity of the executed path can be increased or decreased by the Steward."

Additional geometric parameters

- Wheelbase: 2800 mm;

- Weight of WEpod: 1350 kg.

2.2.2. Architecture

The (high-level) control architecture of the WEasy shuttles is presented in Figure 2.12. It is visible that the representation is done differently than for the newly designed bus. The main difference is that the sensors have been left out in this original design.

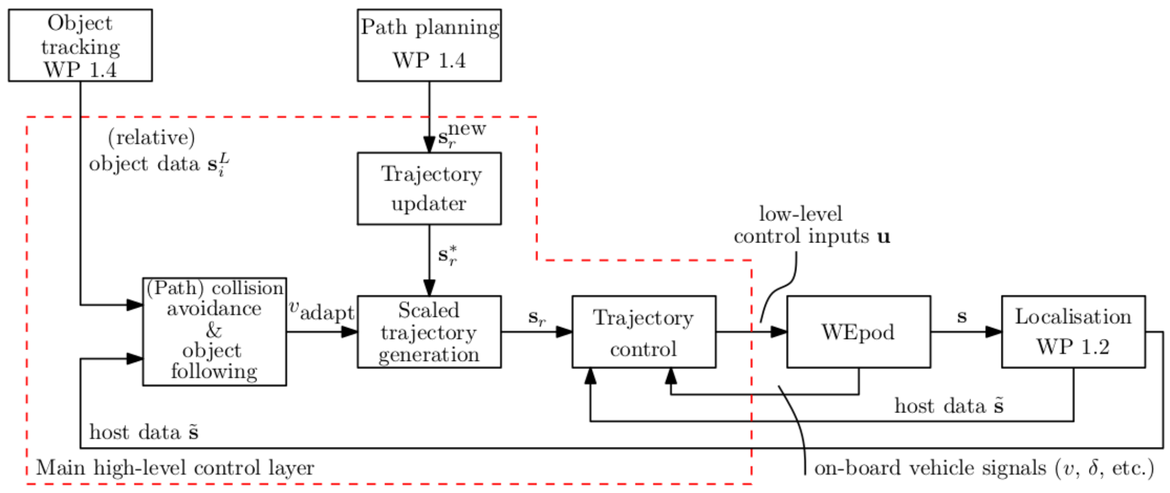

Object tracking WP 1.4
Path planning WP 1.4
s_r^{new}
(relative) object data s_i^L
Trajectory updater
s_r^*
low-level control inputs $\mathbf{u}$
(Path) collision avoidance & object following
v_{adapt}
Scaled trajectory generation
s_r
Trajectory control
WEpod
$\mathbf{s}$
Localisation WP 1.2
host data $\tilde{s}$
host data $\tilde{s}$
Main high-level control layer
on-board vehicle signals (v, δ, etc.)

Figure 2.12: Control architecture of the WEasy shuttles. Retrieved from: [34].

A description of the modules that are regarded as the high-level controller (indicated in Figure 2.12 by area that is delimited by the red dotted line) has been found and retrieved directly from [34] and is presented below.

Trajectory control
Following a desired trajectory $\mathbf{s}_r$ in the absence of other traffic.

Scaled trajectory generation
Slowing down reading out the nominal trajectory stored as a table in the on-board WEpod computer. By slower reading out the desired trajectory, effectively a lower desired velocity is achieved. This methodology is called time-scaling.

Object-following
Adapting the velocity of the vehicle to other relevant road users.

Path predictive collision avoidance
Predicting other road users' trajectories and, if the intersect with the WEpods trajectory, activate a braking action.

Collision avoidance
Braking to prevent (or at least mitigate the effects of) a collision with other road users, including vehicles, bicyclists, and pedestrians.

Trajectory updater
A newly provided trajectory $\mathbf{s}_r^*$ will be smoothly connected to the current one. This is relevant, because it is expected that, due to memory constraints, the trajectory for the upcoming 200 m or so will be loaded, and subsequently will be updated during driving.

2.2.3. Driving modes

The WEpods have the same three driving modes as described in *section 2.1.3*: manual, semi-autonomous and autonomous, described in each of the following items. The switching between the driving modes for the WEpods is visualized in Figure 2.13.

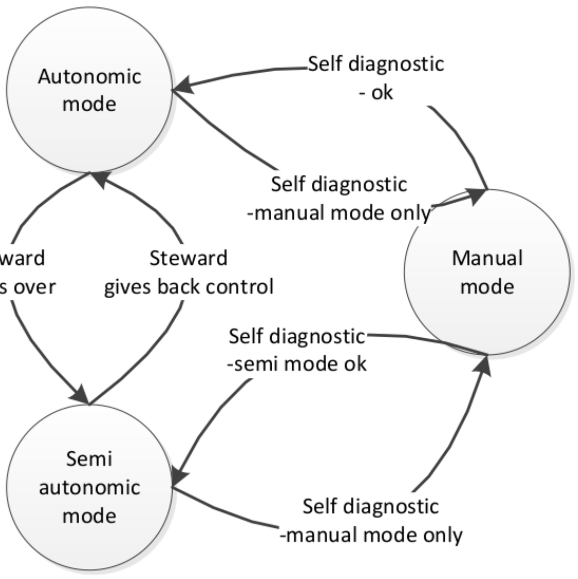

Autonomic mode
Self diagnostic - ok
Self diagnostic -manual mode only
Steward takes over
Steward gives back control
Manual mode
Self diagnostic -semi mode ok
Semi autonomic mode
Self diagnostic -manual mode only

Figure 2.13: Switching between the three possible driving modes. Retrieved from: [27].

2.2.4. Modelling and control

This section is aimed to describe the modelling and control of the WEpods. The vehicles are actuated by exerting a steering angle to the front and to the back axle (which are capable of turning independently). One approach for obtaining a model of such a vehicle is to take the bicycle model with non-holonomic constraints for movement in the lateral direction. Although they can be actuated independently, the control applied to the WEpods is the same for the front and back and acts in a counter-steering manner. Therefore the bicycle model approach can be simplified to a unicycle model.

Unicycle representation

The unicycle model that is used for representing the vehicle, is shown in Figure 2.14.

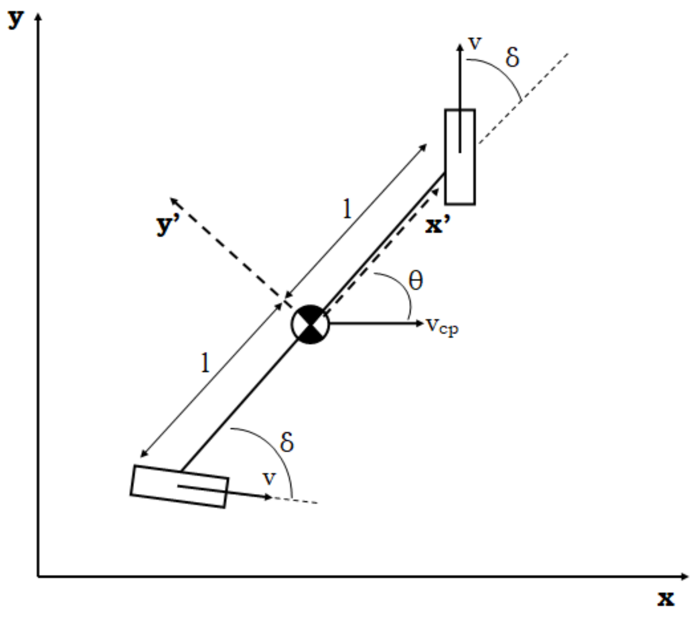


Figure 2.14: Unicycle model representation of the WEasy. Adapted from: [17].

First, the kinematic model is retrieved, in which a description is given purely based on motion - no forces are regarded. The front and rear axle of the pods are connected in such way that the exact opposite value of the steering angle is given to the the back axle of the vehicle as to the front axle. The velocity of the wheels in their own forward direction is indicated by v, the angle under which the wheels operate in comparison with their local longitudinal direction by δ. Therefore, the overall vehicle speed of the center point equals in the longitudinal direction (i.e. along the x'-axis of the local coordinate system):

$$v_{cp} = v \cdot \cos\delta. \tag{2.1}$$

Expressing this speed in terms of the global coordinate system (x,y) this comes down to:

$$\begin{aligned} v_{cp,x} = \dot{x} = v_{cp}\cos(\theta), \\ v_{cp,y} = \dot{y} = v_{cp}\sin(\theta). \end{aligned} \tag{2.2}$$

The angular velocity of the vehicle is expressed as the time derivative of the vehicle orientation in global coordinates (i.e. the x,y system), θ: $\omega = \dot{\theta}$. The angular velocity is determined by dividing the 'perpendicular' velocity by the radius, which in this case equals half of the wheelbase (1400mm): l. Thus:

$$\omega = \dot{\theta} = \frac{v\sin(\delta)}{l} \tag{2.3}$$

Now, by substituting equation 2.1 into equations 2.2 and taking equation 2.3, the kinematic model can be set up as:

$$P_{kin} = \begin{cases} \dot{x} & = & v \cdot \cos\delta\cos(\theta) \\ \dot{y} & = & v \cdot \cos\delta\sin(\theta) \\ \dot{\theta} & = & v\sin\delta/l \end{cases} \tag{2.4}$$

The nominal dynamics of the system are defined by means of a state vector and a state space representation. The state space representation is given in equation 2.5 (an expression for the A-, B- and C-matrices is provided later).

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + B\mathbf{u} \\ \mathbf{y} &= C\mathbf{x} \end{aligned} \tag{2.5}$$

The state vector is defined as follows:

$$\mathbf{x} = \begin{bmatrix} x_{pos} & y_{pos} & \theta & v & \delta \end{bmatrix}^T \tag{2.6}$$

In which the entries mean: x_{pos} represents the position in longitudinal direction of the vehicle in global coordinates, y_{pos} represents the position in lateral direction of the vehicle in global coordinates. θ indicates the orientation of the vehicle in the global coordinate frame and finally v and δ indicate respectively the forward speed of the vehicle and the steering wheel angle in the local coordinate system.

Input

The input of the system consists of the desired speed and the desired velocity v_{des} and δ_{des} respectively and results as an output from the trajectory controller, which has an input s_r (r indicates the reference) of:

$$s_r = \begin{bmatrix} x_r & y_r & \theta_r & \dot{x}_r & \dot{y}_r & \dot{\theta}_r & \ddot{x}_r & \ddot{y}_r & \ddot{\theta}_r \end{bmatrix} \tag{2.7}$$

Therefore, the input vector **u** is defined as:

$$\mathbf{u} = \begin{bmatrix} v_{des} & \delta_{des} \end{bmatrix}^T \tag{2.8}$$

Finally, the output vector contains the actual velocity v and steering angle δ is equal to the state vector:

$$\mathbf{y} = \begin{bmatrix} x_{pos} & y_{pos} & \theta & v & \delta \end{bmatrix}^T \tag{2.9}$$

The complete system becomes:

$$\dot{\mathbf{x}} \quad = A \cdot \begin{bmatrix} x_{pos} & y_{pos} & \theta & v & \delta \end{bmatrix}^T \quad + \quad B \cdot \begin{bmatrix} v_{des} & \delta_{des} \end{bmatrix}^T \tag{2.10}$$

$$\mathbf{y} \quad = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{pos} & y_{pos} & \theta & v & \delta \end{bmatrix}^T = \mathbf{x} \tag{2.11}$$

Now, it is interesting to inspect how the desired control law relates to the actual output of the vehicle. The low level control is exerted by actuators that are constructed by the company EasyMile. This controller functions as a black box and takes away the nonlinearities at a low level. The actuator dynamics, however, are known to behave as follows:

$$\dot{v}(t) = -\frac{1}{\tau_v} v(t) + \frac{1}{\tau_v} v_{des}(t - \phi_v), \tag{2.12}$$

$$\dot{\delta}(t) = -\frac{1}{\tau_s} \delta(t) + \frac{1}{\tau_s} \delta_{des}(t - \phi_s). \tag{2.13}$$

In which $\tau_{v,s}$ indicate the time constants and $\phi_{v,s}$ are the time delays that describe the dynamics for the speed of the tires. These values have been identified (and verified in comparison with a previous identification) in [18] and measure the following:

$$\begin{array}{ccccc} & & \tau_v & = & 0.496 \quad [s] \\ & & \tau_s & = & 0.17 \quad [s] \\ \phi_v & = & \phi_s & = & 0.105 \quad [s] \end{array} \tag{2.14}$$

Based on the equations that were drawn up before (equations 2.4, 2.12 and 2.2.4), the A and B matrices become as follows:

$$A = A(x) = \begin{bmatrix} 0 & 0 & 0 & \cos(\delta)\cos(\theta) & 0 \\ 0 & 0 & 0 & \cos(\delta)\sin(\theta) & 0 \\ 0 & 0 & 0 & \sin(\delta)/l & 0 \\ 0 & 0 & 0 & -1/\tau_v & 0 \\ 0 & 0 & 0 & 0 & -1/\tau_s \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1/\tau_v & 0 \\ 0 & 1/\tau_s \end{bmatrix}. \tag{2.15}$$

2.2.5. Fault tolerance measures

Right now, there are some physical redundancies in the sensors, as they have some overlap in namely their ‘view’ to the front. The perception system consists of six lidar (laserscanner) sensors, eight cameras (which are used as monoview sensors only) and three radars. This system as a whole is redundant. The lidars have 360 degree view, therefore have some (semi-)redundancy for higher resolutions. The lidars and cameras provide raw data, whereas the radar already performs some filtering and abstraction in order to output meaningful data. Because it has its own software module inside, it is able to output information about the reliability of its measurement too. This is forwarded along with the actual object estimation.

Modules that make use of their own software modules too are the localization and the software prediction. The localization module assesses the amount of standard deviation that is resulting from its computations. When this has an unreasonable value, it is planned in future (for the development of the *Mission*), that the in-lane localization module will take over (note that this is a costly process, CPU-wise).

2.3. Problem formulation

The aim of this research, as stated in the introduction, is to achieve a higher level of autonomy within the WEpods project by including certain fault tolerant control measures. Because it is impossible to tackle everything within the scope of a master’s thesis, it is important to identify which part of the system requires an update most. An initial overview of the possibilities of where focus on, are as follows:

- restrict to hardware failures of the car body (steering gear, motor, actuation, ...)
- restrict to failures in the electrical circuit (battery malfunctioning)
- restrict to localization module (mainly sensoring, existing of: GPS, IMU, camera, radar, lidar)
- restrict to path planning module (control: receiving data from localization module, object detection and high level controller)
- restrict to object detection module (existing of: camera, radar, lidar)
- restrict to computational limits (software and control)

By having conversations with experts from the company, an overview of most pending fault occurrences is gained. This overview is included in the next section.

2.3.1. Overview of fault occurrence

The two modules that were introduced in the previous section that currently suffer from most problems, are:

- Processing time of the object detection and/or their motion prediction. When the environment is too densely filled with objects, the pathplanner controller is unable to process this amount of information quickly enough and gets stuck. A decision could therefore be made to drive slower or to stop performing any overtaking manoeuvre.
- The localization module is sometimes off, leading to a frequent take-over request for the pilot. A visualization is provided in *subsection 2.3.3*. An improvement which could be made here is to assess a degraded version of the performance, i.e. drive at a lower maximum speed, and execute this in the autonomous mode.

As the former merely focuses on processing speed, efficient coding and hardware capabilities, the focus of this master's thesis will lie on the localization module, because it is possible to perform diagnosis of the faults well and adapt the performance accordingly depending on the situation in order to retrieve a situation in which the desired behavior still can be executed (i.e. not come to a complete system failure).

2.3.2. Detailed description of localization module

Before moving on, it is important to gain some understanding in how this localization module actually works. A functional description has been provided by [10].

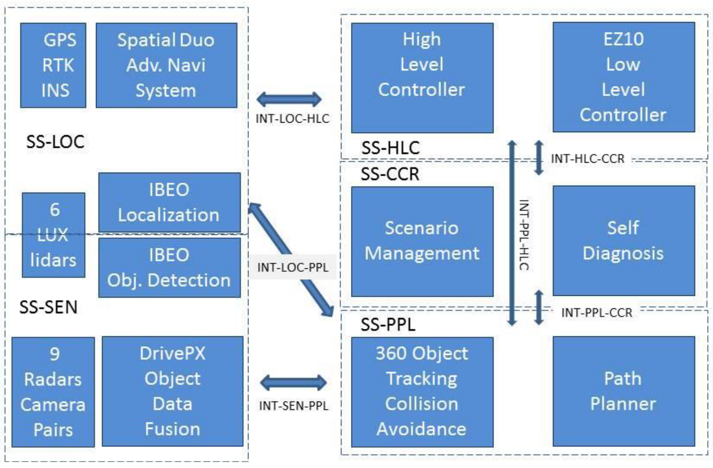
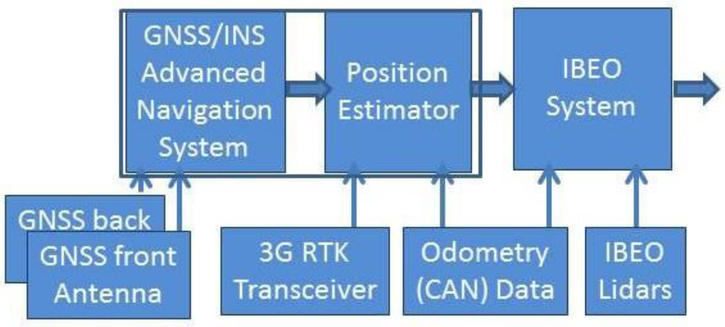
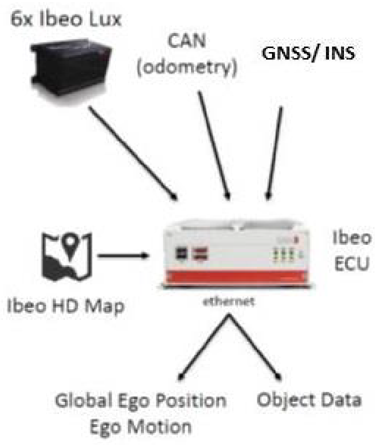


Figure 2.15: Interface design description from localization to High Level Controller. Retrieved from: [23].

As shown in Figure 2.15, the localization block (top left) consists of multiple algorithms that combine GPS (Global Positioning System) updated with a GNSS (Global Navigation Satellite System) correction from a so-called RTK (Real-Time Kinetic) network, IMU (Intertial Measurement Unit as part of the INS: Inertial Navigation System [19], laserscanner: lidar (IBEO) and odometer information with an off-line map of the environment in order to determine the position, orientation and velocity of the vehicle in the actual world. These algorithms consist of on the one hand the advanced navigation unit combining the GPS, IMU and odometer information, of type: 'Spatial Dual'. More information can be found, also on documentation (see Spatial Dual Reference Manual for that) on [1]. On the other hand it consists of the laserscanner from IBEO which combines the information this sensor retrieves as a point cloud with the outcomes of the advanced navigation unit, the latter is visualized in Figure 2.16.



(a) IBEO input block description



(b) IBEO input visualization

Figure 2.16: Visualization of the IBEO input sources, left: block-scheme, right: visuals. Retrieved from: [23].

Three drawbacks of using the IBEO system as it is are:

- using this external piece of hard- and software is expensive;
- the software is a blackbox, making it impossible to inspect what happens inside and debug accordingly;
- it appears to perform less robust than expected and required (in certain situations).

A laserscanner has the characteristic that is very well capable of determining a well-defined pointcloud in case there are not too many inputs from the surroundings. This way objects can be succesfully distinguished from another and recognised from what is known beforehand. Also, in an environment where surroundings are very similar: for example a narrow street with high buildings that look similar makes it hard to distinguish the individual characteristics and determine the position. Another drawback from a dense environment, is that GPS performs less, as the signal coming from the satellite could be obstructed. An overview and qualitative effect of environmental parameters, among which also wheather influence, on GPS is given by [22].

The assumption is made that the system therefore contains faults, that are mostly inevitable. The aim is to find a way to handle these faults in such way that the system does not emerge into a complete failure.

2.3.3. Visualization of faults in the localization module

In this section, visualizations are provided in Figures 2.17-2.19 that show the (unexpected) behavior of the advanced navigation localization and IBEO localization respectively. In order to understand what is shown in the visualizations, it is important to get some understanding of what is visible. The screenshots were obtained from a so-called visualization tool RVIZ that run on a docker that contained ROS. In the figures, the path is shown in yellow. The path that is planned by the trajectory controller (as explained in section 2.2.4) based on the map containing the trajectory and the information about the location and orientation of the vehicle, is indicated by the thick green path ahead of the vehicle. The vehicle is represented by a rectangle containing an arrow that indicates the driving direction. The orange vehicle and past trajectory represent the measurements from the advanced navigation unit, whereas the green vehicle and past trajectory represent the localization produced by the IBEO software piece. It is clearly visible in all three, but mainly in Figure 2.17, that the advanced navigation (orange) has an offset and constantly produces wrong information about the exact location of the vehicle. Figure 2.18 introduces us to some sort of unrealistic 'sawtooth' behavior. Over time, the IBEO localization tends to perform this kind of faults. At the start of the day, this behavior is less.

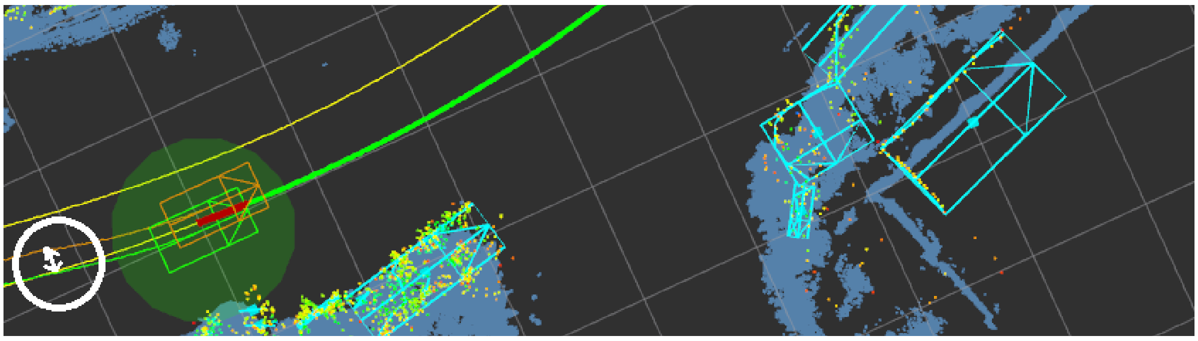


Figure 2.17: Offset advanced navigation. Log files from March 18, 2019.

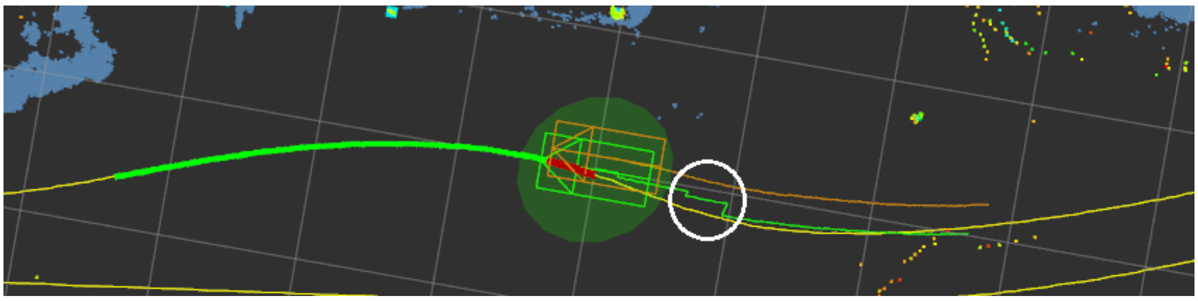


Figure 2.18: IBEO sawtooth. Log files from March 18, 2019.

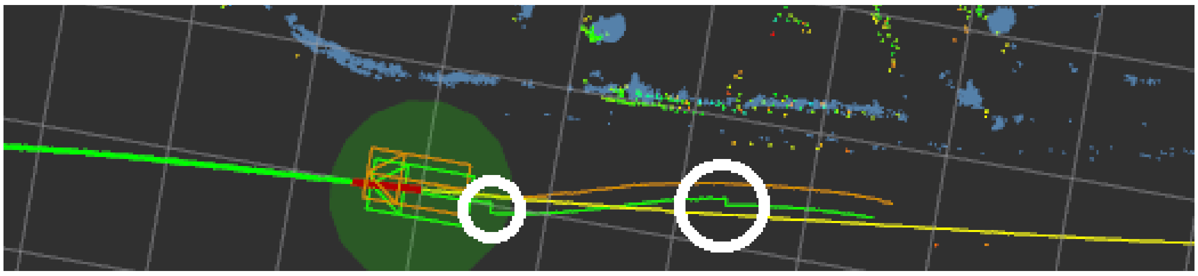


Figure 2.19: IBEO overcompensation. Log files from March 18, 2019.

As a first step towards fault tolerant control for the WEpods, the decision is made to focus on the fault detection of the IBEO localization sensor. This is a problem that influences the pathplanner most in a negative way. The path is planned based on the localization output. This leads to overshooting, zig-zag like behavior in case the vehicle was already at the right track. RRC will be helped tremendously when this problem could be tackled successfully. In order to be able to do so, several methods regarding the fault detection strategy and preparation of the input are required. These will be elaborated on in Chapter 3.

3 Methods

This chapter is aimed to describe the methods that are used in order to introduce fault tolerance into the WEpods project, and in particular to tackle the problems in the localization module. First, an introduction about the chosen fault diagnosis: detection and isolation (FDI) methodology is given along with the general approach. Next, the data that is used in this research and all the required cleaning and preparation steps are described. The third section provides insight into the uncertainty in the system. Finally, the actual model is introduced first by setting up the kinematic model, then extending the model with an observer and finally preparing the model for fault detection and isolation.

3.1. Fault diagnosis: model-based

In this section, the main idea of what approach is used for fault diagnosis in the localization module of the WEpods is sketched. There are many ways to enable detection of faults in an automated system. [21] stressed that there are four main approaches for fault detection: signal-based, change detection, model-based and soft computing techniques. All methods incorporate the computation of a so-called residual. This residual indicates the difference between the nominal (healthy) and actual (faulty) conditions of a system. Once this residual becomes large enough (i.e. exceeds a pre-computed threshold) and stays large for a long enough duration, a detection takes place. In order to choose which method is most suitable for the application at hand, the characteristics of each of these approaches are briefly summarized below.

While soft computing techniques and machine learning in general are very promising, they complicate characterization of modelling uncertainty and derivation of robust detection thresholds. Therefore, this approach is disregarded within the rest of this research.

The signal-based approach seems an appropriate candidate at first glance: the output of the IBEO localization unit directly provides the knowledge that something is incorrect. It is visible in Figure 2.18, that the output indicates that the vehicle performs a sudden jump from one location to the other. This seems an easy way to detect faults. However, using this information for detection directly, is only valid for the conditions in which smooth driving behavior is assumed. When the vehicle drives for example some sort of zig-zag type of path, it is apparent that the localization will change instantaneously under some sharp angle, simply because this is what the vehicle does in reality. Therefore, the signal-based approach is not robust enough, as it does not apply to all possible driving behaviors.

The change detection approach uses statistical parametric models for detection of faults. An example of which is that sudden changes in the mean of the sensor output indicate that it is erroneous. The sensors used in the WEpods-project are assumed to have zero mean of the output, because otherwise the sensors will always have a constant offset. The drawback of this method is that when it is applied directly to signals from a system, the assumption needs to be made that the system remains in steady state, which is not always the case in driving [11].

The model-based approach is therefore the best candidate for the application at hand. For this approach, a model of the system that can determine its nominal behavior is required. This nominal (fault-free) model will be compared to a faulty representation of the system. The general adopted approach is shown in Figure 3.1. The black ('inner') stream is used when only simulated variables are regarded, the red ('outer') stream comes into play as well when real recordings (measurements) from the vehicles sensors are used.

The MODEL blocks contain the kinematic model, that will be described in detail in section 3.4.1, based on the equations introduced in section 2.2.4. The top model (REAL) has slightly different parameters than the bottom one (NOMINAL). This is done to be able to include the deviations that reality has in comparison to the nominal model. The parameters that can be tweaked in order to represent reality better with the 'real' model, are the time delays on the actuators: τ_v (steering angle) and τ_s (drivetrain). However, these parameters have been identified twice as described in [18], so not much deviation from reality is expected.

The input u, that is the same for both models, consists of the desired velocity v_{des} and the desired steering angle δ_{des}. The nominal state is represented by x: position in x-direction, position in y-direction, orientation θ, speed v and steering angle δ. This state is subject to (measurement) noise, represented by external input ξ. When this is added, output y is obtained. After that, there is a functionality to add faults ϕ manually. The objective is to compare the model of the real (faulty) system, with the nominal model. This comparison delivers residual r, which will be used as an input to the ALARM DETECTOR. This block is responsible for raising an alarm when it detects a fault. It does so when the residual exceeds the threshold, as explained above and in [21] and calculated in section 3.4.4.

In order to avoid excessive drift between the nominal model and reality (either output of the REAL model or of the measurements inside the RECORDING block directly), the former is not implemented in open loop, i.e. in free run. Instead, an observer is employed. This observer gain Λ consists of a diagonal matrix with λ: $0 \leq |\lambda| \leq 1$ on its diagonal entries. Values for λ close to 1 mean that a lot of weight is given to system measurements (trusting them more), while values close to 0 mean that more weight is given to the model (trusting the measurements less). The observer design is explained in section 3.4.2.

At the time of drawing up the scheme, it was unknown that the desired velocity and steering angle were logged in the recording as well. That is why the PATH PLANNER loop was initially included for calculation of these values. Later, it turned out that these inputs were stored as well, see also section 3.2. Therefore, the drawing can be simplified by removing all the red parts, except for the RECORDING block with connections to the switch and in addition to u directly.

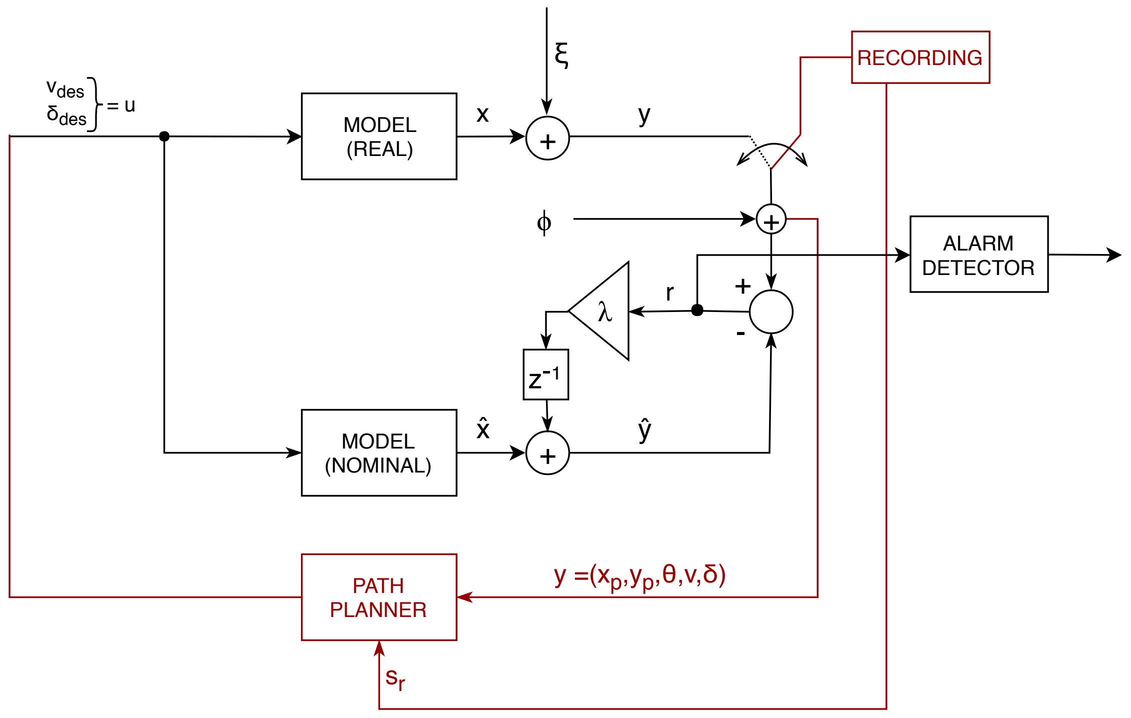


Figure 3.1: Approach for developing a fault detection system.

3.2. Data

The data used in this research is recorded during testdrives with the WEasy pods and stored in so-called rosbag logs. First, it is identified which of these rosbag logs are of interest and how to convert them to readable data. Then, some preparation (coordinate frames) and cleaning steps are introduced for both the input and localization data in order to be able to use it for further computation.

3.2.1. Rosbag logs

During test drives with the WEpod vehicles, data is recorded in rosbags. A rosbag is used to handle certain message data and is aimed to store, process, analyze and also visualize data that is put into them. A so-called node (which is a process that performs computation) can exchange messages via a certain topic [26]. For the localization module for example, there is a rosbag called 'advnav_gps_localization_<date_and_time>.bag'. This contains a topic that is called */advnav/gps_localization* which exchanges messages from type: *wepods_-msgs/Localization*. All has been inspected by running the pre-built WEpods docker on Ubuntu, which has ROS (Robot Operating System) installed. It became clear that quite some information is returned within such a message. However, not everything is either interesting or filled with data, so they can be disregarded. Therefore, first the relevant rosbags are identified below and then a selection on which of the messages are relevant will be provided. Then it is described how this data can be converted to MATLAB-readable input. It is important to use a format that can be handled by MATLAB, because the research will be exerted using this programme as it contains a powerful extension called Simulink, in which models can be built.

Relevant rosbags

Several recorded rosbags are of interest to the research, namely:

- *vehicle_can_bridge_commands*: here, the actual control commands for the speed (v_{des}) and the steering angle (δ_{des}) are stored (either resulting from a manual joystick movement in case of the manual driving mode (recall section 2.1.3), or from the pathplanner in case of the autonomous driving mode;
- *vehicle_can_bridge_measurements*: this bag contains the measurements coming from the odometer sensors in the wheels of the speeds and steering angles that are actually achieved;
- *pathplanner_trajectory_controller*: here, the output of the pathplanner defined as the objected state s_r of the vehicle is stored (recall equation 2.7, containing position in x and y, orientation and speed) if it would be following its projected reference path;
- *advnav_gps_localization*: this returns the output of the GPS localization with relevant values for the configuration of the vehicle (x, y, θ) in UTM coordinate frame, the xx-,xy- and yy-covariances and the standard deviation of theta;
- *advnav_wgs84*: relevant values for the configuration of the vehicle (x, y, θ) in WGS84 coordinate frame;
- *ibeo_republisher_gps_localization*: this returns the output of the GPS localization of the external software piece from IBEO with relevant values for the configuration of the vehicle (x,y,theta) in UTM coordinate frame, the xx-, xy- and yy-covariances and the standard deviation of theta;
- *ibeo_republisher_localization*: this returns the output of the overall localization of the external software piece from IBEO with relevant values for the configuration of the vehicle (x, y, θ) in UTM coordinate frame, the local xx-, xy- and yy-covariances and the standard deviation of theta.

Relevant topics and messages

By inspecting the rosbags through the docker with ROS installed by writing the command: *rosbag info <path_-to_rosbag_file/rosbag.bag>*, it became clear which topics were used. To find out what messages that are published within these topics are relevant, *rosbag play* is run, while displaying the according topics that have been identified by *rosbag info*. In Appendix D, the topics (/...), message names (*italic*) and important messages (SpeedMps, etc.) are listed for each of the rosbags. The (Header).Stamp.Sec message includes the Unix time that indicates the time instance of saving each of the logs in amount of seconds after the Unix Epoch (initial time stamp 0), which is set to Thursday, 1 of January, 1970 [3]. The (Header).Stamp.Nsec can be added, so nanosecond precision is reached.

Convert rosbag logs to MATLAB data

In order to be able to use the data in MATLAB at a later stage, the rosbag data had to be converted in such a way that it became readable to MATLAB as well. An m-file (executable MATLAB file) called *read_in_data_from_-rosbags.m* was made in order to reach this goal and is included in Appendix E. This m-file uses a graphical user interface (GUI) that opens the file browser appointed to the folder where the data is saved off-line. The user can navigate manually to the desired test data (date and time of the rosbag logs). Once arrived at the correct folder, all *vehicle_can_bridge_commands* files with the extension *.bag* are shown, so that the desired commands can be chosen. Using the information about the path where the file has been found (date and time window) and the information from the name of the selected file (exact time and recording number), the other six important rosbags (as listed before) are loaded in automatically. Using the command *rosbag*, a rosbag will be selected for which a certain topic can be read. Using the bag selection functionality *select* in MATLAB, all the messages belonging to the relevant topics (the list above is used in the code) for the selected bag, are saved in a *structure*. The interesting messages are then taken out of these structures successively and finally stored in *.mat* files, so they can be loaded in at a later stage to use for the fault diagnosis.

3.2.2. Coordinate frames

Because the data recordings exist in various coordinate systems (as briefly mentioned in the rosbag descriptions), a preparation step is required to get the data into the same configuration. The coordinate systems used are:

- *Local coordinate system in the vehicle*
  The origin coincides with the mass point of the vehicle, x pointing in the longitudinal (forward) direction and y pointing in the lateral (to the left) direction.
  The covariances of the IBEO localization are recorded in this frame.

- *Global Cartesian (Eucledian) coordinate system in UTM coordinates.*
  The Universal Transverse Mercator (UTM) system contains a latitude and longitude value that is unique for its position in the world [2][5].
  Most of the data is recorded in this frame.

- *Global coordinate system containing geographic coordinates WGS84*
  The World Geodetic System (WGS) frame had is last revision in 1984 (hence WGS84) and has its origin at the center of mass of the earth and represents its location with respect to this location based on spherical equations [4].
  The GPS (*advnav_wgs84*) is measured and saved in this frame.

- *Global Cartesian (Eucledian) coordinate system*
  By normalizing the data that is provided or converted in the UTM frame for the x- and y-position of the vehicle, the origin is moved to the start of the driven track.
  This coordinate system is what will be used throughout the entire approach.

The WGS84 is first transformed to UTM using the following code, as part of the script *read_in_data_from_-rosbags.m* provided as Appendix E. The UTM zone of the test location is ‘32N’ [31]:

```
%% This script is intended to convert data to correct coordinate systems
utmstruct = defaultm('utm');
utmstruct.zone = '32N';
utmstruct.geoid = wgs84Ellipsoid;
utmstruct = defaultm(utmstruct)

lat = y_adv_odom;
lon = x_adv_odom;

[x_adv_wgs84_utm,y_adv_wgs84_utm]=mfwdtran(utmstruct,lat,lon)
```

Then, in order to compare the UTM frame to the frame used in the kinematic model, a translation has to be made by subtracting the UTM offset. In Figure 3.2 it is shown how the coordinate systems relate to each other.

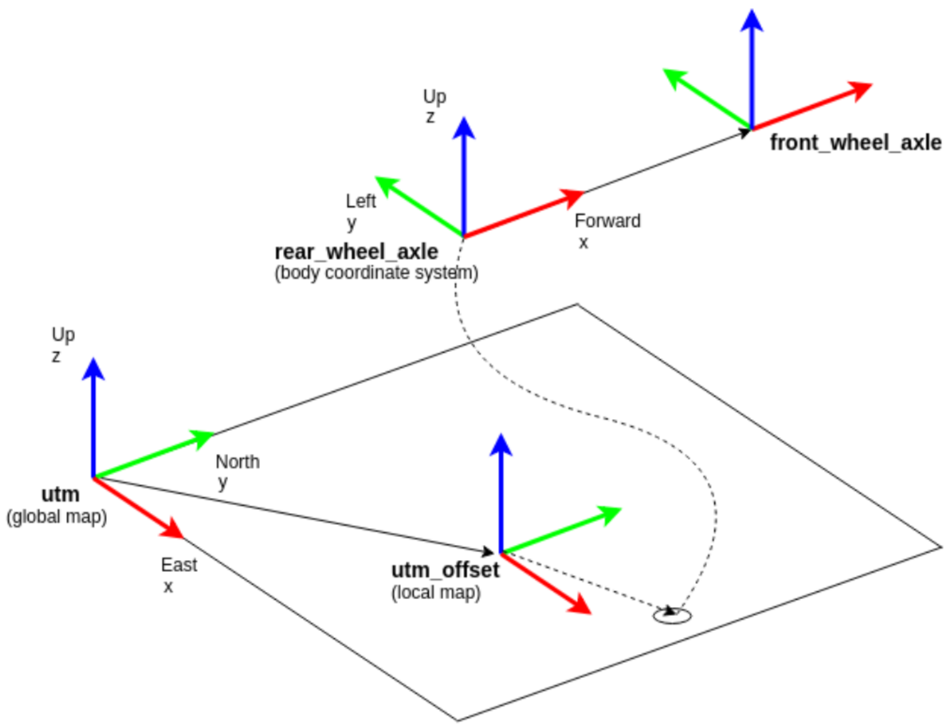


Figure 3.2: UTM coordinate system relating to global and local frames. Retrieved from: [35].

3.2.3. Input data

The data used was retrieved from the rosbags that were filled during the test drives. First, the date and time of the log is retrieved by using the MATLAB function *datetime*. The dateType (third entry) is defined as the 'posixtime', because this represents the Unix time. The timezone is set to Europe/Amsterdam, because Weeze is in the same timezone as Amsterdam.

```
date_and_time = datetime(commands(1,1),'convertfrom','posixtime','TimeZone','Europe/
    Amsterdam')
```

The input to the vehicle was logged in the *vehicle_can_bridge_commands* bags. In order to see whether the vehicle actually meet the desired speeds and steering angles, the commands are compared to the measurements from the vehicles' internal sensors. This data is logged in *vehicle_can_bridge_measurements*. Several cleaning steps were required to make the data usable. These are described in Appendix F.

3.2.4. Localization data

The localization data returned by the advanced navigation GPS, WGS84, IBEO GPS and IBEO localization sensors have been inspected on sampling frequency and other apparent oddities. Three similar preparation steps as for the input data emerged and are included in Appendix F as well.

3.2.5. Sensor (fusion) noise data

Finally, the noise data has been inspected at the time it was used for threshold computation. Only then, it appeared that there was something wrong with the noise data and some preparation steps were needed in order to correct for this, which are added to Appendix F as well.

3.3. Uncertainty characterization

It is of utmost importance to take influencing factors due to uncertainty into account when modelling a vehicle. When left unassigned, these factors could lead to unexpected changes between reality and the model, simply because the vehicle does react differently and the controller will be unable to reach the reference values. Below, a brief overview is provided of (external) influences that could possibly make up for the uncertainty in the system.

- **Time delays on actuators**
  The actuators that are responsible for controlling the vehicle, are subject to time delays. These have been identified for both the steering angle (τ_s) and the drivetrain (τ_v). In case they change, because of wear-out or accumulation of dirt or dust in the hardware, the performance will be different than expected, because the desired input does not lead to the desired output one-to-one any longer.

- **Tire stiffness/deformation**
  The upper operational limit of the vehicle lies at a speed of 15 km/h, which is slightly less than 4.2 m/s. As deformation of the tires and hence increased friction will play a major role for higher speeds, it is assumed that this will not pose any problem [8].

- **Wheel slip**
  When driving on an icy road under a speed that is too high, the wheels can start slipping when braking is initiated. Also aquaplaning is an example of what can cause the wheels to slip. This is obviously not measured by the odometer sensors (although there are ways of incorporation this, as found by: [7]), as the wheel remains in standstill. This is problematic for the sensor output. However, the WEpods are driving when the weather conditions are good enough, so this is not a problem for the data logs used either.

- **Tire inflation**
  One influencing factor is the extent to which the tires are in- or deflated. Because of increased friction when the tires are slightly deflated, the actual speed will be lower than in full inflated state. Also, the number of rotations of the wheel is not linearly related to the distance travelled in the same way as before any longer, because the diameter is slightly decreased. The measurements from the wheel sensors will therefore assume a larger distance travelled than is actually the case. In this application, the position based on those sensors is not saved. The vehicle speed, however, is saved. Since the speed is equal to the derivative of the position, the problem arises here as well. The tires are pumped up on a regular basis, so for this research, it is assumed that tire inflation does not influence the data accuracy.

- **Wind**
  Side-wind can influence the lateral dynamics of the vehicle [36]. It has to deliver a larger force in order to compensate for the extra force that pushes the vehicle aside. This is something that has to be taken into account in the control loop. Not only the lateral dynamics can suffer from wind influence, but the longitudinal dynamics as well, think: headwind and tailwind.

- **Inertia**
  The inertia of the vehicle is another influence on the longitudinal dynamics. When a heavy object has accelerated to a certain speed, it is harder to again slow it down. For higher speeds, this influence gets larger. The influence due to inertia can be determined based on the weight of the vehicle of 1350 kg and a maximum speed of 4.2 m/s.

- **Wear-out of brakes**
  In case the brakes start to wear out, the braking applied as a result of the control, will lead to different results than expected as well. When the speeds tend to overshoot, this can imply such wear-out of the brakes.

3.4. Model design

In order to build a model, the software plugin Simulink as an extension of MATLAB has been used. This section describes the Simulink model and what it consists of. First, the kinematic model is discussed in section 3.4.1, then the observer design is included in section 3.4.2, after which a validation of the overall model is given in section 3.4.3. After that, two sections are devoted to fault diagnosis: fault detection is described in 3.4.4 and fault isolation in section 3.4.5.

3.4.1. Kinematic model

In Figure 3.3, the first Simulink model that is set up for preparing the kinematic model is shown. It includes one function block that calculates the A-matrix that is dependent on the state. Recalling again the equations:

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + B\mathbf{u}, \\ \mathbf{y} &= C\mathbf{x} = \mathbf{x}. \end{aligned} \tag{3.1}$$

With x and u:

$$\mathbf{x} = \begin{bmatrix} x_{pos} & y_{pos} & \theta & v & \delta \end{bmatrix}^T, \mathbf{u} = \begin{bmatrix} v_{des} & \delta_{des} \end{bmatrix}^T. \tag{3.2}$$

And A and B:

$$A = A(x) = \begin{bmatrix} 0 & 0 & 0 & \cos(\delta)\cos(\theta) & 0 \\ 0 & 0 & 0 & \cos(\delta)\sin(\theta) & 0 \\ 0 & 0 & 0 & \sin(\delta)/l & 0 \\ 0 & 0 & 0 & -1/\tau_v & 0 \\ 0 & 0 & 0 & 0 & -1/\tau_s \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1/\tau_v & 0 \\ 0 & 1/\tau_s \end{bmatrix}. \tag{3.3}$$

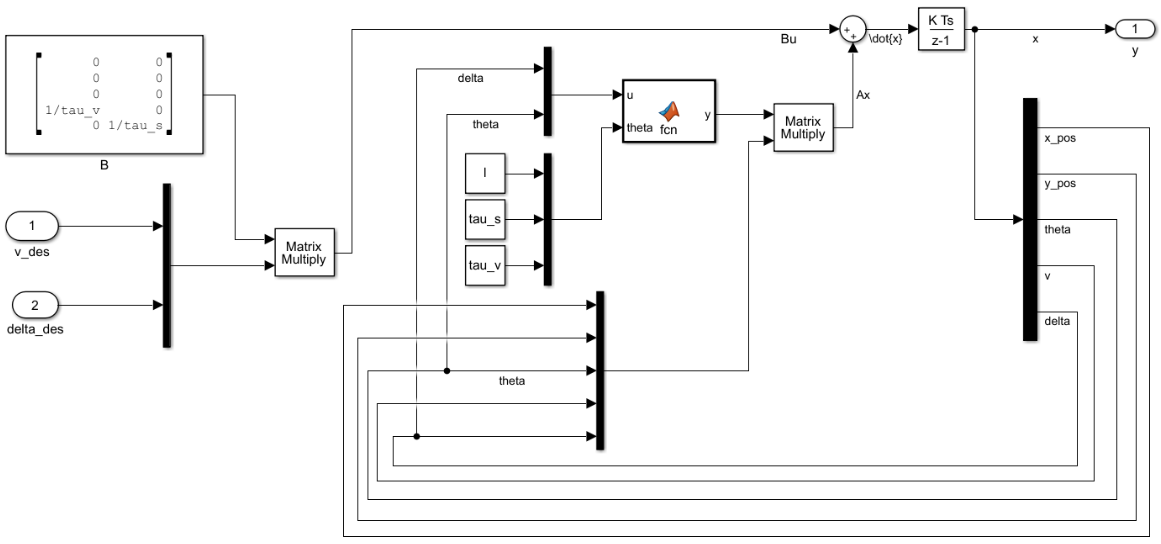


Figure 3.3: Kinematic model in Simulink.

A discrete-time integrator block, which is visible in the top stream as $\frac{K*Ts}{z-1}$, is included for retrieving the state $\mathbf{x}$ from the computed $\dot{\mathbf{x}}$. The step time (Ts) of the model has been defined as 0.01s, because the sampling frequency of the commands equals 100Hz. The end time (Tstop) of the simulation is defined as the number of samples (i.e. always 30,000) of the commands minus 1, multiplied by the step time. These preparations are made in the *prepare.m* file.

Initial position

The $\frac{K*Ts}{z-1}$ integrator block starts from an initial position. This initial position includes the configuration of the vehicle (i.e. its state) from the moment that the driving has been initiated. The initial position is determined in *prepare.m* as well. Instead of taking the UTM coordinates of the location of the vehicle (x,y), the data is normalized letting the vehicle drive always from a starting position of 0 meter driven in x-direction and 0 meters driven in y-direction. This is done to simplify inspection. Therefore, the initial position in x and y always equals 0. The orientation angle θ is taken as the first value of the measurements because the vehicle is left off under a certain angle in the global frame. In case no orientation is known (the WGS84 sensor for example only returns postions), the orientation angle is set to 0. The speed and steering angle (that are mostly non-zero as well at the start of a log because the vehicle has been driving already), are taken as the first entry of the can bridge measurements, or, when the sensor outputs a speed as well, using that speed output.

3.4.2. Observer design

When the model is run in open loop for the x- and y-position, it turns out that the model diverges when compared to the measurements. See Figures 3.4 and 3.5.

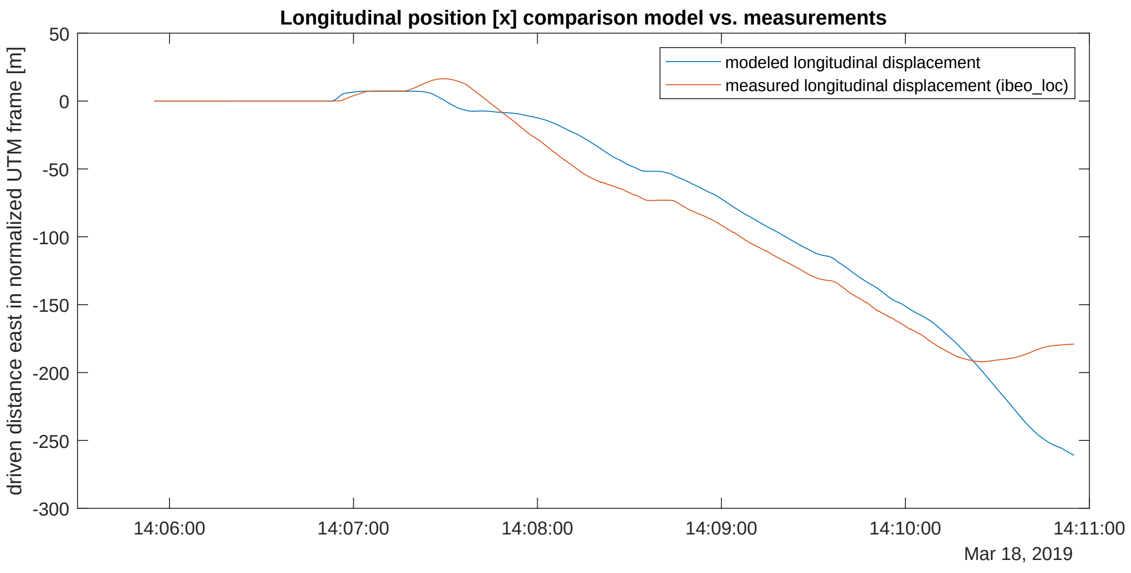


Figure 3.4: Comparison between measured x-position (normalized) and the open loop model output for the x-position.

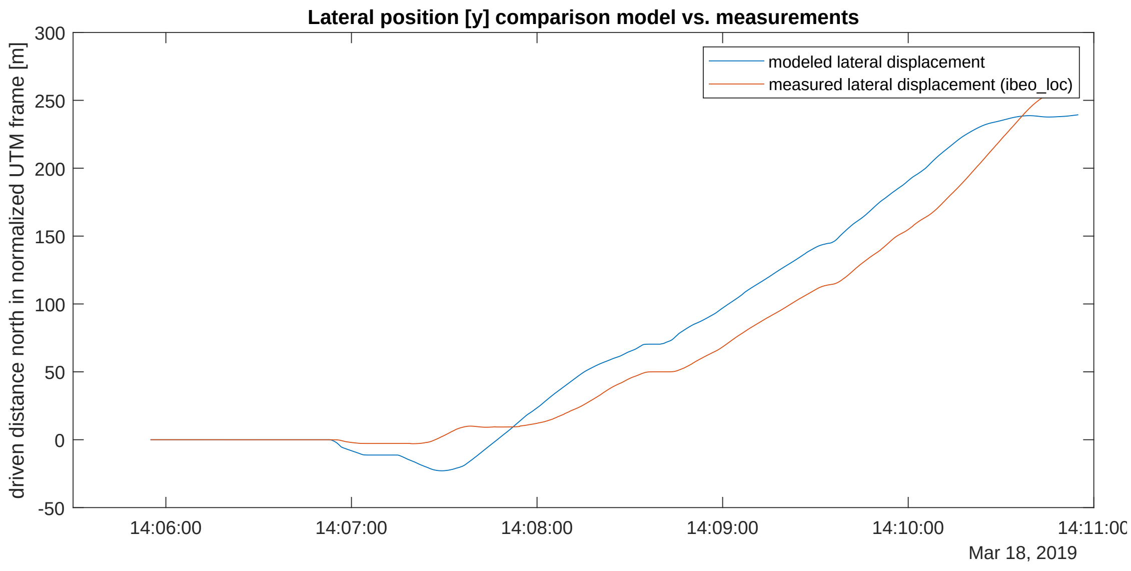


Figure 3.5: Comparison between measured y-position (normalized) and the open loop model output for the y-position.

This is why an observer is included, that prevents drifting away of the model, by calculating the difference between the model output and the actual model (under some gain value) and adding this offset to the current output of the model. This way, it will shift towards the ‘correct’ position. The equations used for designing such an observer are as follows:

$$\begin{aligned} \hat{x}_+ &= f(y,u) + \Lambda(\hat{y} - y), \\ &= Ay + Bu + \Lambda(\hat{y} - y). \end{aligned} \tag{3.4}$$

$$\begin{aligned} \hat{x}_+ &= f(\hat{x},u) + \Lambda(\hat{y} - y), \\ &= A\hat{x} + Bu + \Lambda(\hat{y} - y). \end{aligned} \tag{3.5}$$

$$\hat{y} = \hat{x}. \tag{3.6}$$

The observer in equation 3.4 will be called a soft observer, that takes the measurements (y) as an input to the kinematic model (by multiplying it with A). The observer in equation 3.5, however, only takes the measurements into account in the observer part ($\Lambda(\hat{y} - y)$). Therefore, the second equation has been used in the model. The Simulink model of this observer design is included in Figure 3.6. Note that the observer is the third input to the mux-block in the middle.

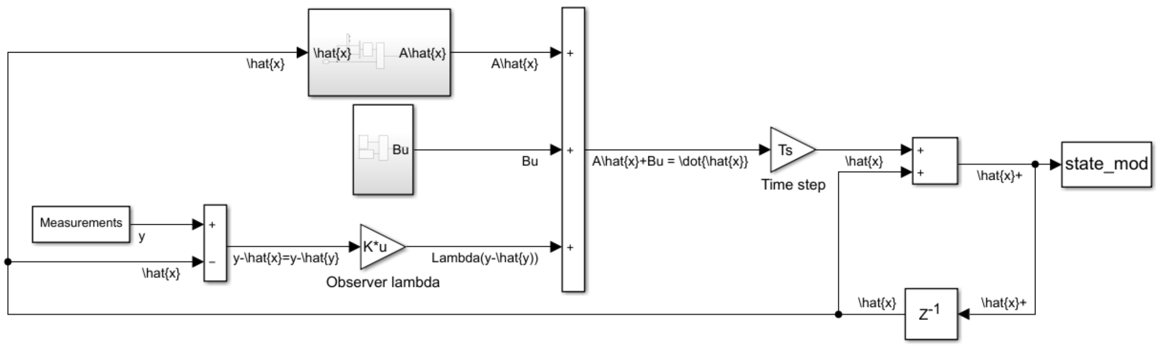


Figure 3.6: Model with observer in Simulink.

It is visible that the model has been refactored in order to achieve a better overview, in comparison to Figure 3.3. The subsystem blocks are shown in Appendix H.

Determination of the observer gain values λ_i in diagonal observer matrix Λ has the following characteristics:

- High observer gain (close to 1) puts a lot of trust on the sensors, while a low observer gain (close to 0) depends solely on the model;
- When there is a lot of sensor noise, it is advisable to trust the model more, whereas in case there is less noise, the measurements could be trusted. This can be determined using covariance indications of the measurements;
- A high gain means that by trusting the sensor a lot, the sensor fault will be ‘followed’ and therefore the observer will not be very sensitive to detecting it. A lower gain is better for the sensitivity and therewith detectability of the observer.

A gain value for Λ is chosen as 0.9. This is because a gain close to 1, takes the measurements into account more. By keeping it close to 1, the model will start ‘following’ the fault. A value of 0.9 forms a good compromise, between filtering out noise and being sensitive to fast changes like those emerging from faults. This way, smooth model outputs are reached that are good for comparison at a later stage. Therefore the matrix looks as shown below in equation 3.7.

$$\Lambda = \begin{bmatrix} 0.9 & 0 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 0 & 0 \\ 0 & 0 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 0 & 0.9 \end{bmatrix} \tag{3.7}$$

New comparisons in the x- and y-positions are made now the model has been updated with the observer loop. The localization output of the model and the measurements are visualized in Figures 3.7 and 3.8.

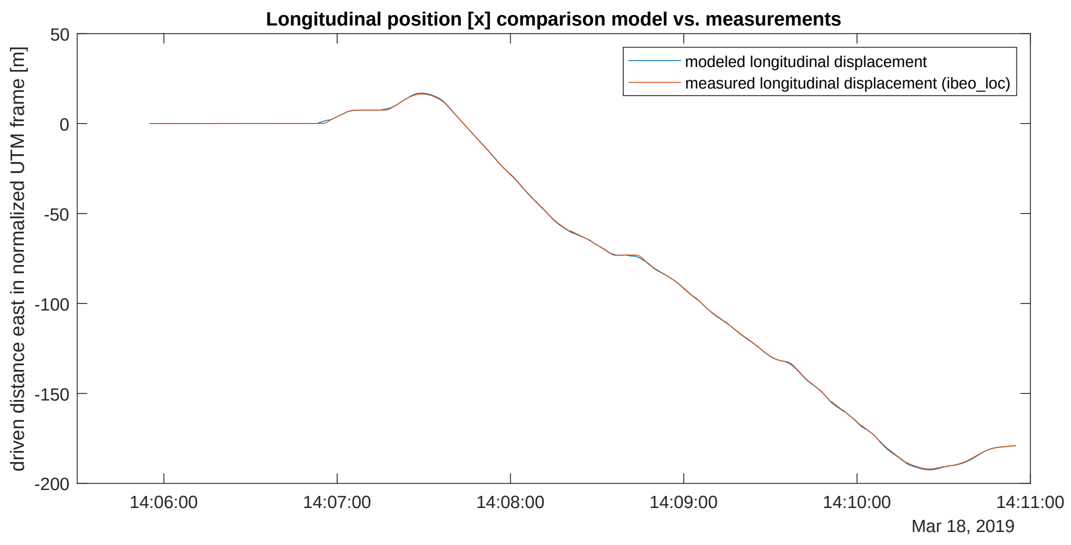


Figure 3.7: Comparison between measured x-position (normalized) and the closed loop model output for the x-position.

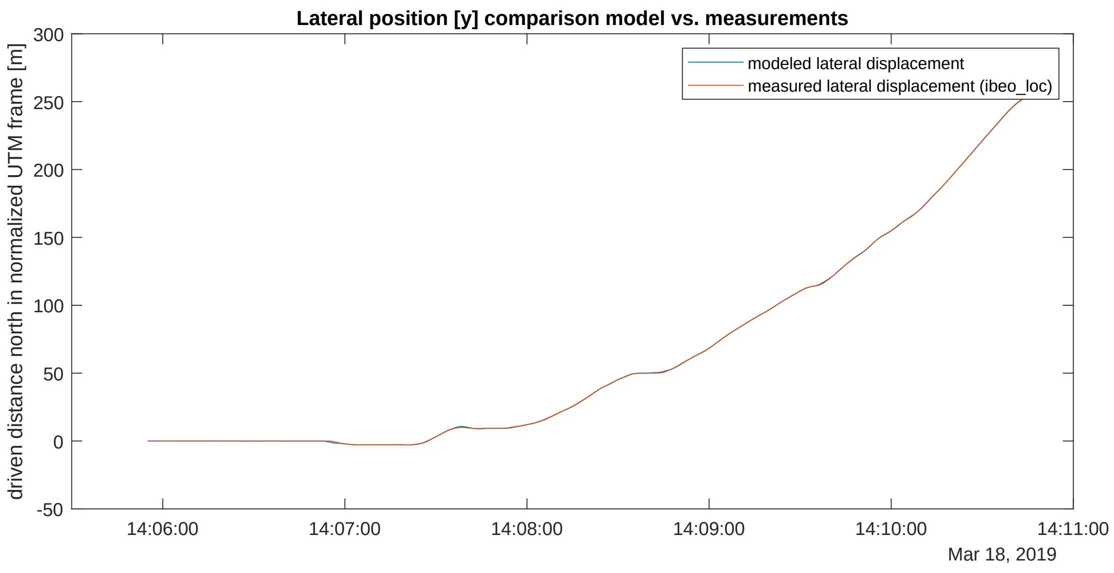


Figure 3.8: Comparison between measured y-position (normalized) and the closed loop model output for the y-position.

3.4.3. Validation

Inspecting the Figures provides the knowledge that the model at least seems to match the measurements quite well. In this section, a validation check will be done to see if the model indeed behaves as expected. When zooming in on the plots of the model versus the measurements, as shown in 3.9 and 3.10, first of all, a small delay of the model with respect to the measurements is visible for the instances where the vehicle changes direction. This is an indication of some unexpected behavior in longitudinal direction in reality, which is investigated further in section 3.4.4.

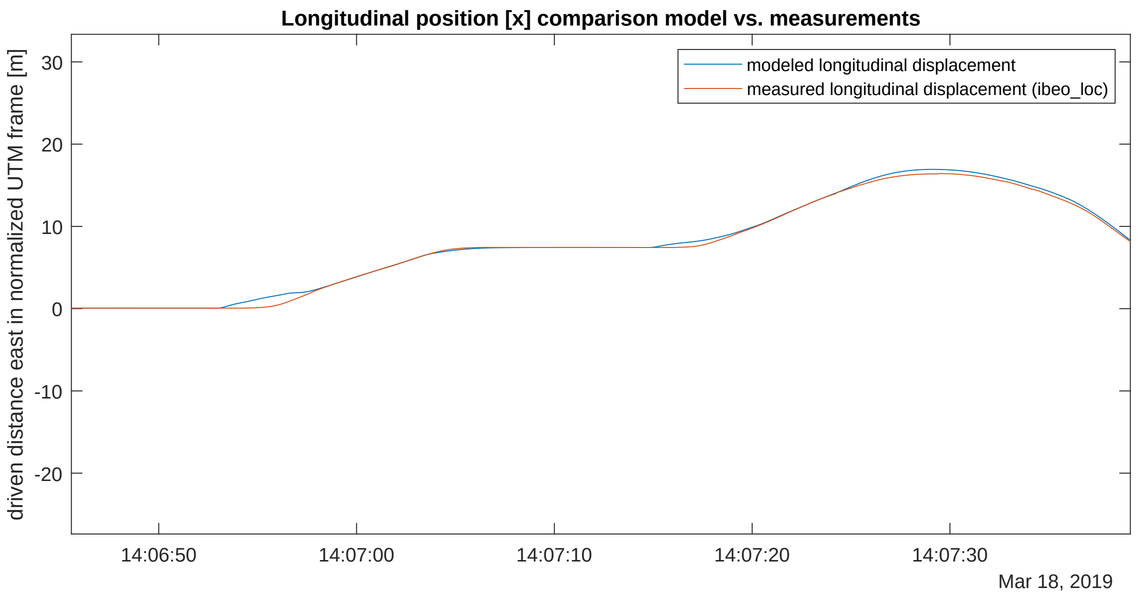


Figure 3.9: Comparison between measured x-position (normalized) and the closed loop model output for the x-position.

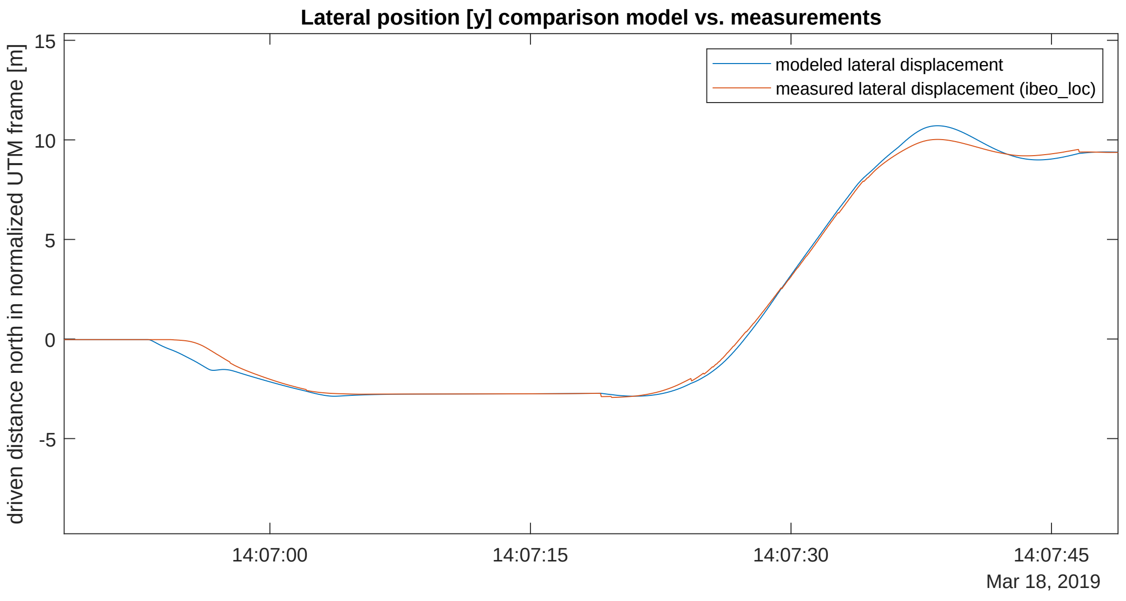


Figure 3.10: Comparison between measured y-position (normalized) and the closed loop model output for the y-position. Large jumps in the lateral position (y) are already visible between 14:07:15 and 14:07:30.

Changed sign in equations for orientation

While checking the model including the observer on validity above, an extra plot was generated for the orientation, as shown in Figure 3.11. It turned out that the model showed something odd for the orientation entry. When the measured orientation angle becomes larger, the model seems to become smaller first before correcting towards the expected value. Since the equation for the orientation is only based on the steering angle and the speed, the latter two were plotted against the unexpectedly shaped orientation to retrace the error. It is presumed that a minus sign is missing in the equations. This is shown in Figure 3.12.

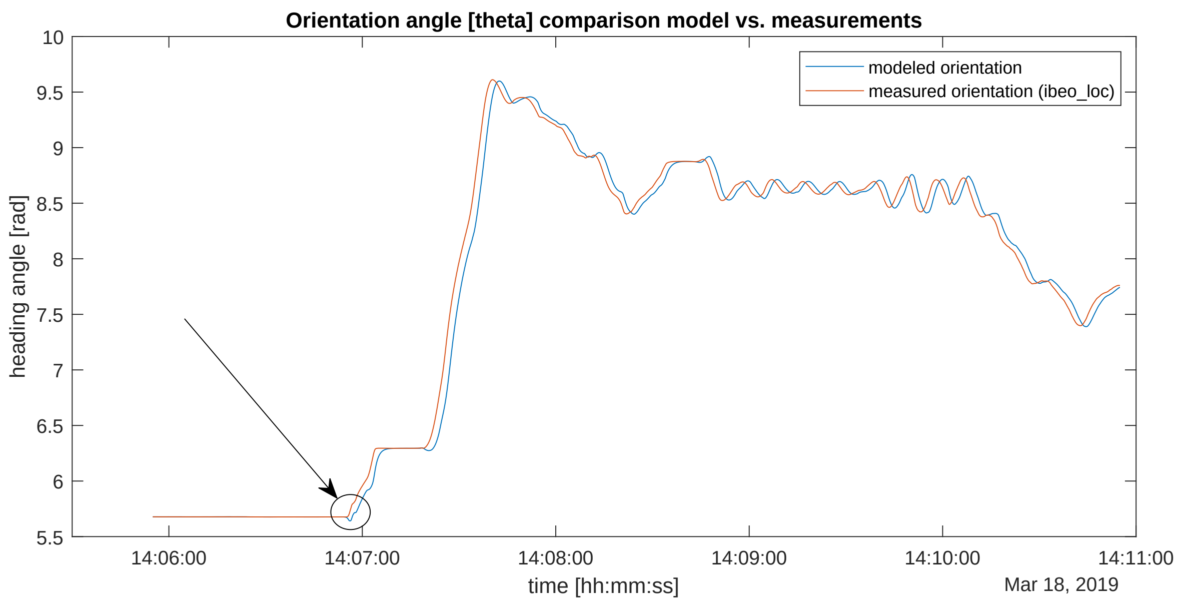


Figure 3.11: Comparison between orientation from the model and the IBEO localization output. The arrow and circle indicate the notable difference in orientation.

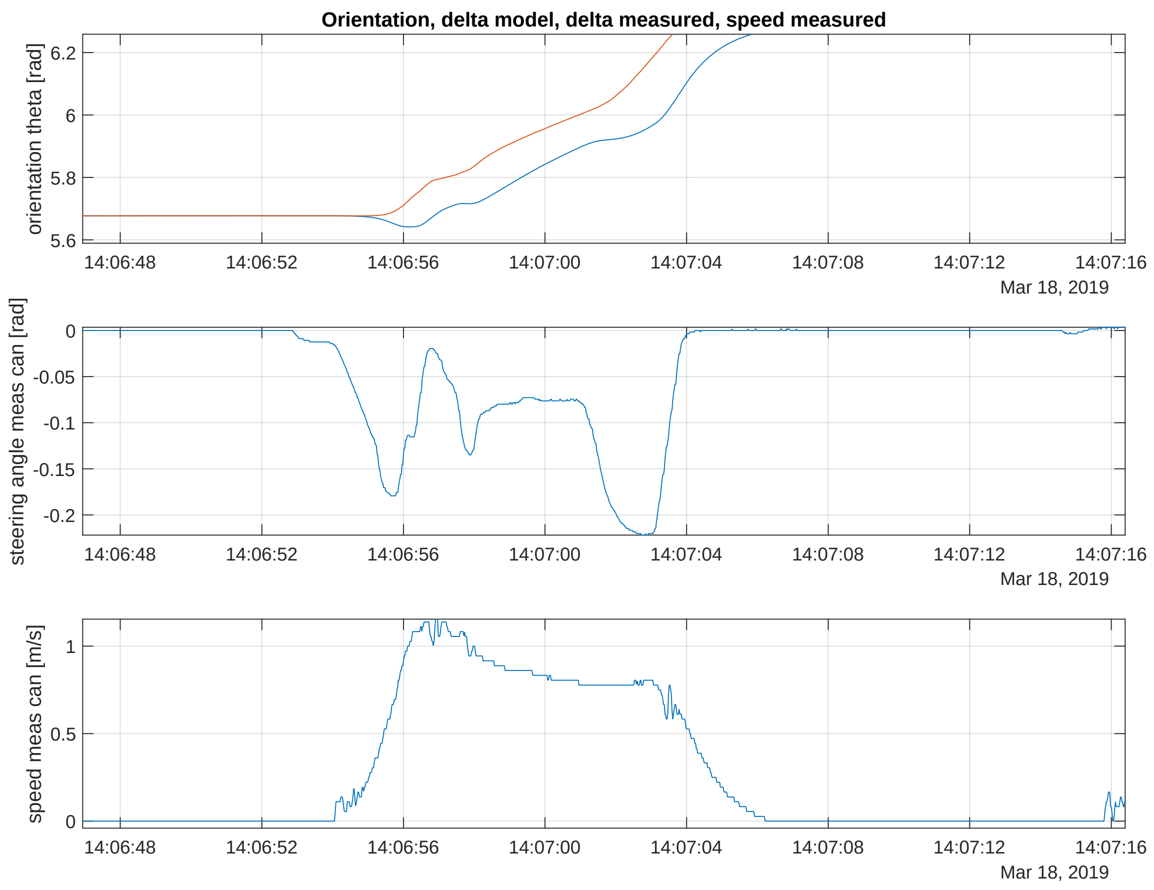


Figure 3.12: Comparison between orientation from the model and the IBEO localization output (top plot), complemented with steering angle (middle plot) and vehicle speed (bottom plot) coming from the can bridge measurements (meas on y-axis).

The relevant kinematic equation is $\dot{\theta} = \sin(\delta)/l * v$, which has been drawn up as one of entries in the A-matrix in the system description $A\mathbf{x} + B\mathbf{u}$. By inspecting this equation and noting that it is multiplied with v_des, it becomes clear that the output will be negative for the plotted steering angle (which remains below 0) and speed (which remains above 0). The steering angle is namely a small negative value, so the output of taking the sinus of that, is negative as well and the speed is positive. Because the length of the wheelbase remains constant and is obviously positive as well, the orientation could never become positive for this plotted time interval. This means that only the observer is responsible for pulling 'up' the orientation towards the correct value, so probably a minus sign is missing in the kinematic equations. This is tested by running the model in open loop (i.e. by setting the observer gain equal to 0). The result is shown in Figure 3.13.

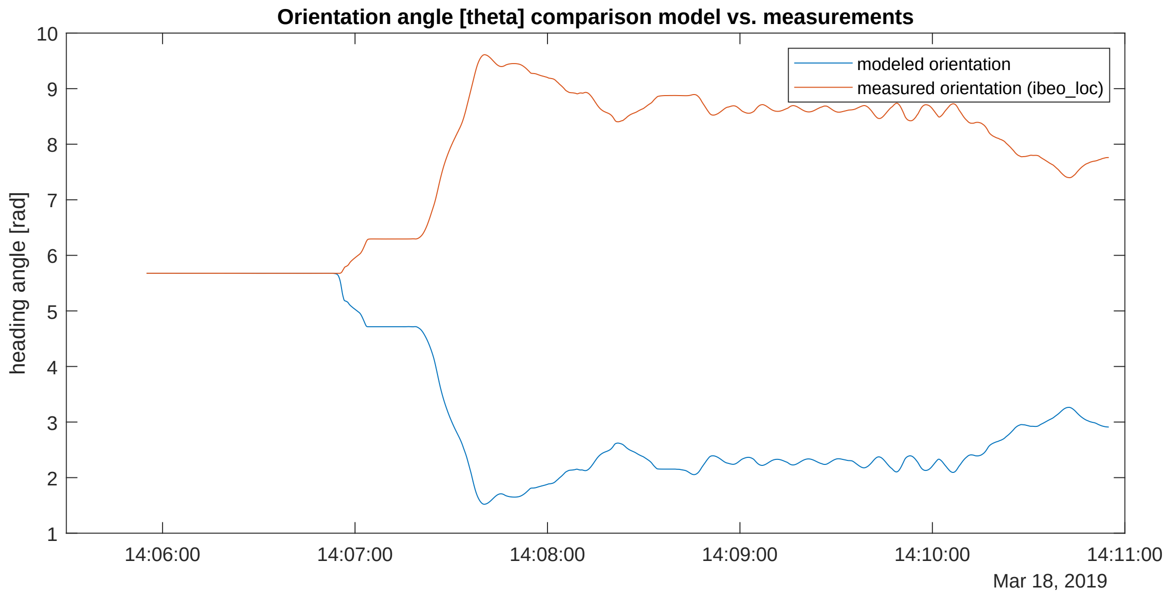
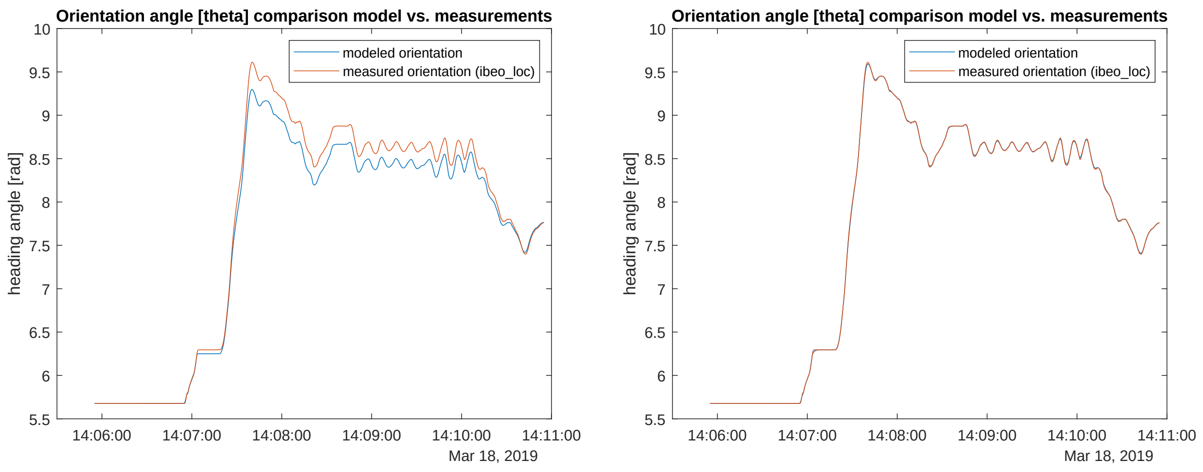


Figure 3.13: Open loop output for modeled orientation with unadapted equations.

This validated the presumption of having a mismatch in the sign. In Figure 3.14, the output for the orientation based on the model when the minus sign is added, are shown. First, in open loop form and then in closed loop, where correction by the observer is performed.



(a) Open loop output corrected sign orientation

(b) Closed loop output corrected sign orientation

Figure 3.14: Corrected sign orientation output, left: open loop, right: closed loop (observer).

From this point onwards, the equations have been updated by the addition of a minus sign:

```
function A = fcn(angles,par)
tau_s = par(1);     % time delay actuators [s]
tau_v = par(2);     % time delay actuators [s]
l = par(3);         % length of half the wheelbase

A = [0 0 0 cos(angles(2))*cos(angles(1)) 0;
     0 0 0 cos(angles(2))*sin(angles(1)) 0;
     0 0 0 -sin(angles(2))/l           0;
     0 0 0 -1/tau_v               0;
     0 0 0 0               -1/tau_s];
```

After updating the equations, the output of the model in x- and y-position slightly improved as well. Two zoomed-in plots of the comparison between the measurements and positions resulting from the model are included in Figures 3.15 and 3.16. Especially the lateral position is improved in comparison to Figure 3.10.

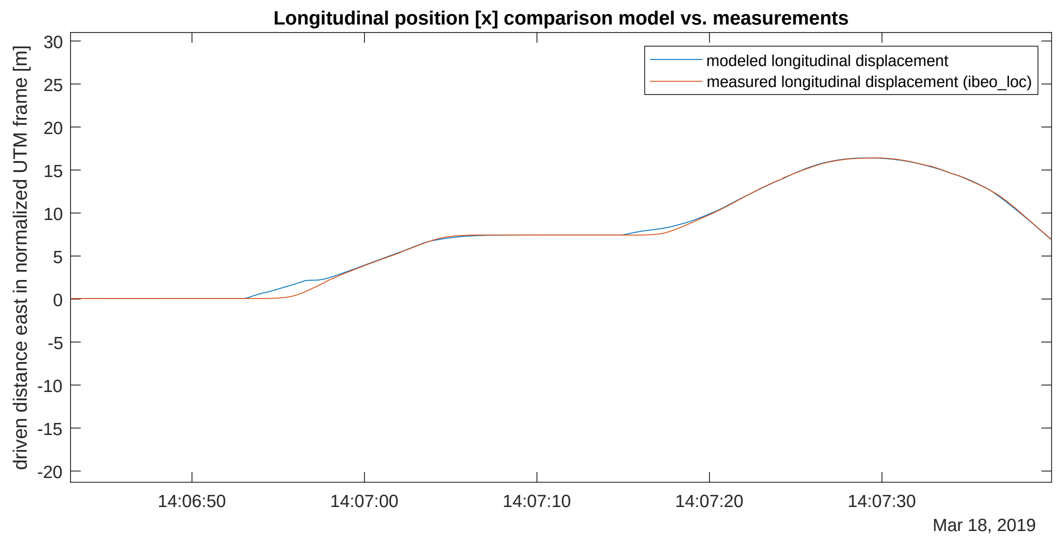


Figure 3.15: Comparison between measured x-position (normalized) and the closed loop model output with updated A-matrix.

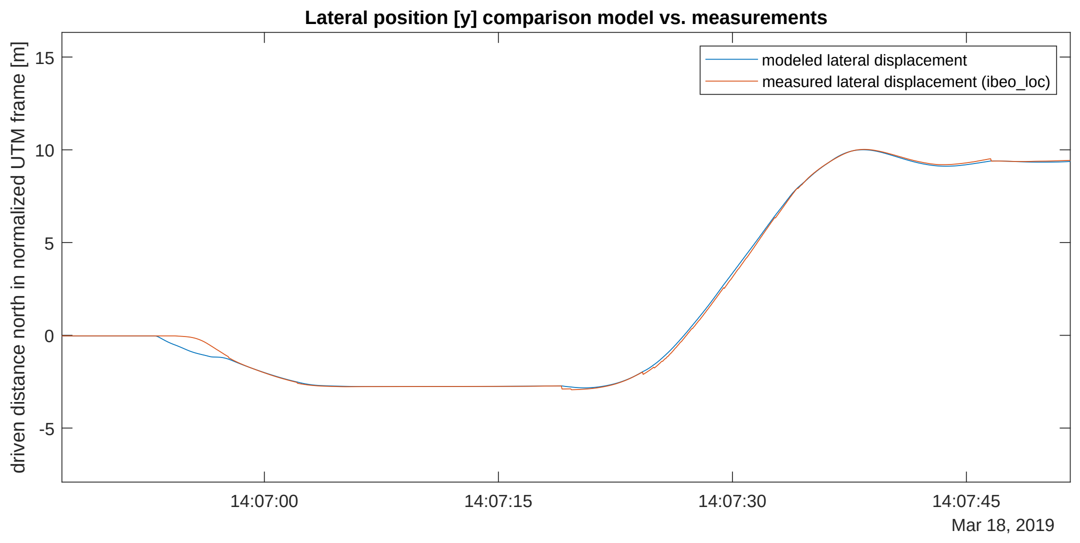


Figure 3.16: Comparison between measured y-position (normalized) and the closed loop model output with updated A-matrix.

3.4.4. Fault detection

Now that the model has been drawn up, validated and prepared accordingly, the actual fault detection can be implemented. For this, first the residuals will be generated that compare the model output to the measurements. As a second step, the model will be split into two observers, one for the localization entries (x, y and θ) and one for the speeds (v) and steering angles (δ). This is done to enable fault detection for the localization module only, without experiencing influence from other sensors and resources that might not be completely fault-free. Various residuals using different methods will be generated again once this update to the model is made. Finally, the threshold is designed which the residual needs to cross in order to detect a fault.

Residual as difference in position

The residual is in the first instance computed by taking the difference between the model and the measurements:

$$r = y - \hat{y}. \tag{3.8}$$

The residual r that emerges for all state entries (x, y, θ, v, δ) for the model versus the sensor used, in this case the IBEO localization, is shown in Figure 3.17.

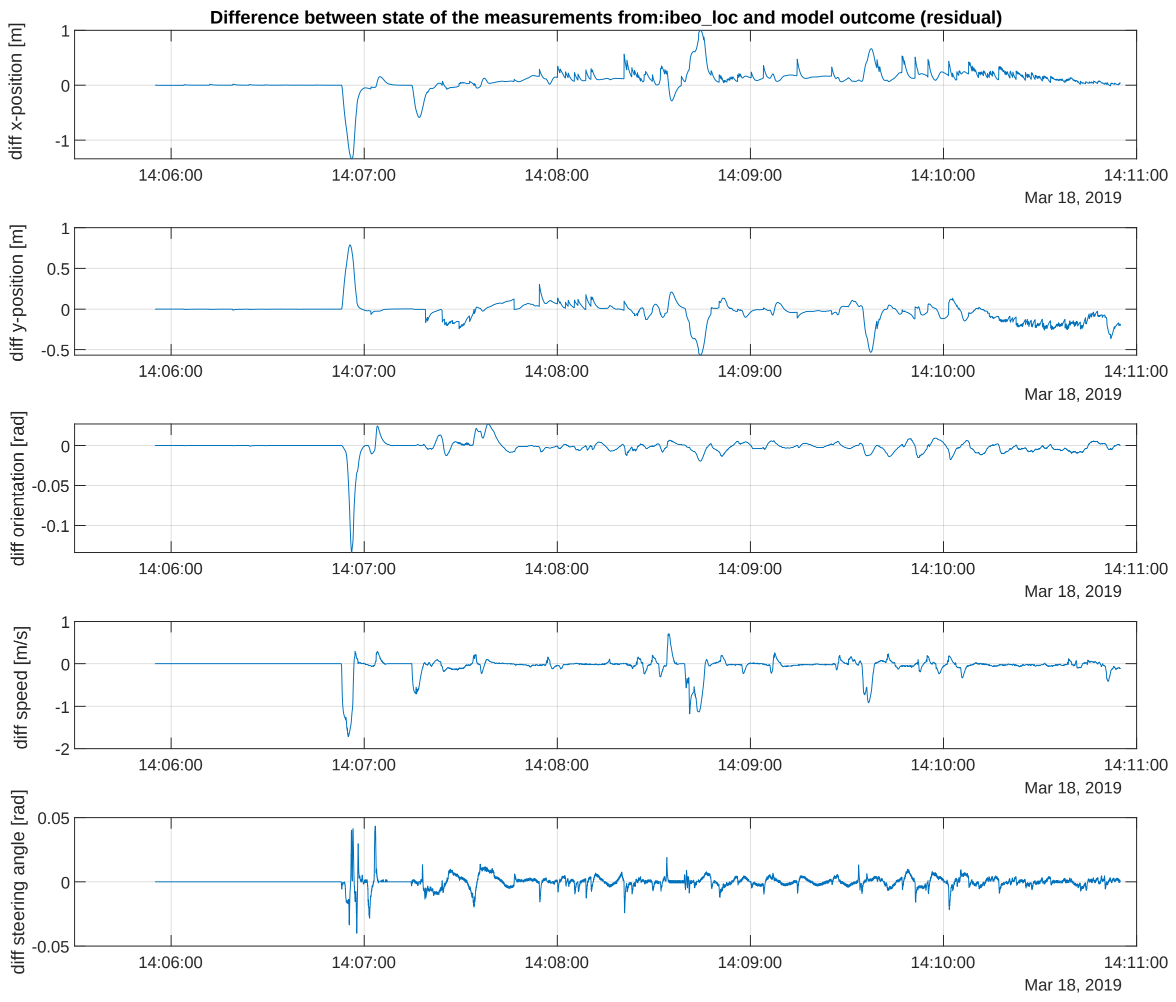


Figure 3.17: Residuals in all state entries (x, y, θ, v, δ) model versus IBEO localization sensor.

It is visible that the residuals for the x- and y-position stay relatively small. Only slightly before 14:07:00, something interesting occurs. There are large peaks visible for both the x- and y-position. By inspecting how the data behaves, it becomes clear that at that point in time the vehicle starts driving. The large difference in speed could mean that this is an unreliable value to use. In order to determine this, the speeds emerging from different sensors and the model are visualized in Figures 3.18 and 3.19 (zoomed in version of the plot in Figure 3.18) for further inspection.

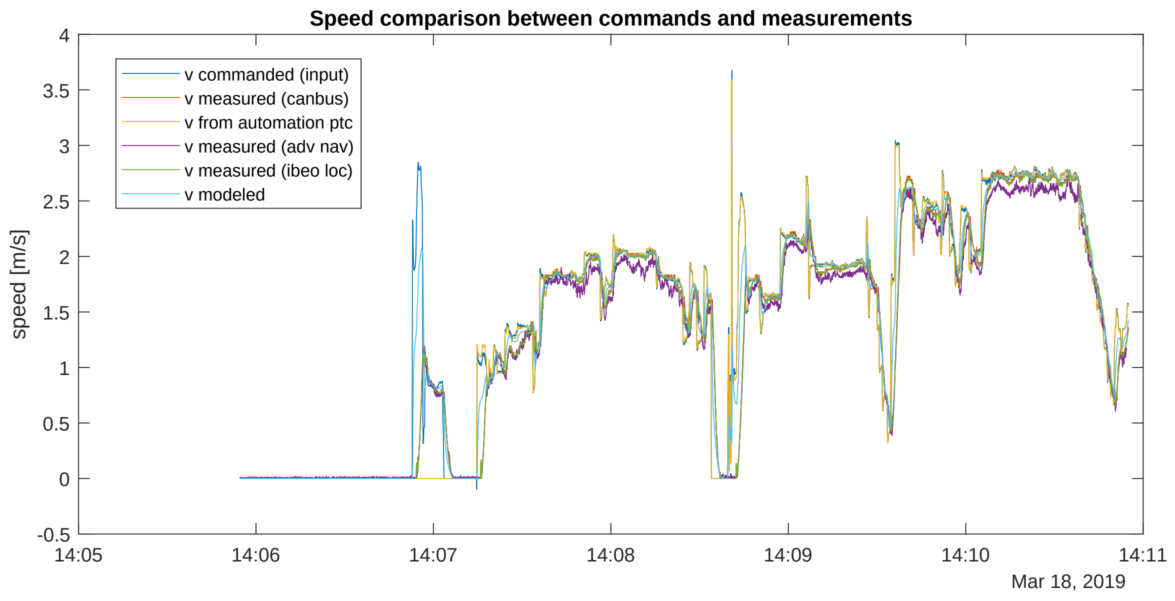


Figure 3.18: All measured speeds vs. model output.

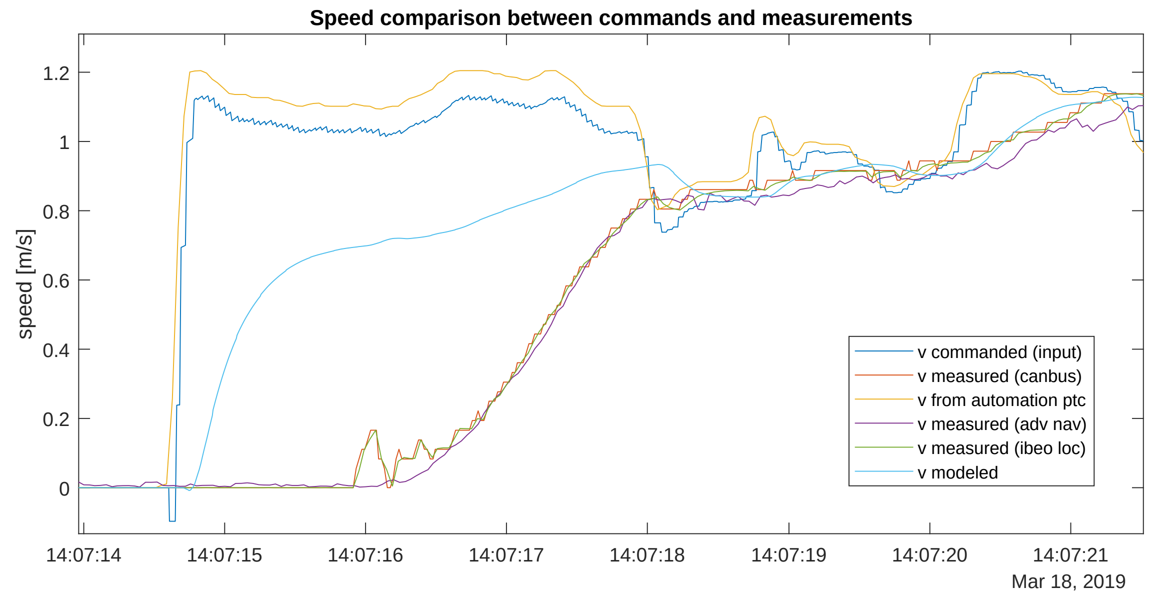


Figure 3.19: All measured speeds vs. model output zoomed in.

It is obvious from Figure 3.19 that something unexpected happens in the control loop of the vehicle: the desired speeds differ significantly from the measured speeds. The accelerations seem to be smoothed by a filter in order to avoid jerks and uncomfortable accelerations of the vehicle. A second influencing factor according to the team, is that the velocity controller only uses P(I) - Proportional (Integration) and does not include feedforward control but only acts when the error became sufficiently large. The controller is unfortunately a black box, so these assumptions have been made based on inspection. This is something that can not be influenced from the position of performing the research, therefore the decision has been made to split the observer into two parts: one for the localization, and one for the steering angle and speed. In order to prevent the localization residual to be heavily influenced by the unexpected speed output, the observer will from now on be two-fold: on the one hand there is the state of the localization (x, y and θ), on the other hand there are the speeds (v) steering angles (δ).

Update model to two observers

The equations for the split observer model are as follows:

$$x_2 = \begin{bmatrix} v \\ \delta \end{bmatrix}, A_2 = \begin{bmatrix} \frac{-1}{\tau_v} & 0 \\ 0 & \frac{-1}{\tau_s} \end{bmatrix}, B_2 = \begin{bmatrix} \frac{1}{\tau_v} & 0 \\ 0 & \frac{1}{\tau_s} \end{bmatrix}, u_2 = \begin{bmatrix} v_{des} \\ \delta_{des} \end{bmatrix}, y_2 = x_2 + \xi_2 \text{ (from can bridge)}. \tag{3.9}$$

$$x_1 = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix}, A_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B_1 = \begin{bmatrix} \cos(\delta_{meas})\cos(\theta_{gps}) & 0 \\ \cos(\delta_{meas})\sin(\theta_{gps}) & 0 \\ \sin(\delta_{meas})/l & 0 \end{bmatrix}, u_1 = \begin{bmatrix} v_{meas} \\ \delta_{meas} \end{bmatrix} (= y2), y_1 = x_1 + \xi_1 \text{ (from sensor)}. \tag{3.10}$$

The second observer is described first, because the output (y_2) represents the input of the first observer. The names 'second' and 'first' are chosen based on the different parts of the state entries that are used. The second observer, which is denoted by equation 3.9, contains the steering angle and speed of the model (which is the second part of the state), whereas the first observer, denoted by equation: 3.10, contains the position in x and y and the orientation angle (which is the first part of the state). For the second model, the input is defined as the desired (commanded) speeds, whereas the first observer takes the actual measured speeds as its input. This is done because there was some unknown conversion between the commanded and modelled speeds as identified above. In order to eliminate this unknown influence on computing the localization output of the model, the measured speeds are taken directly. Then, a choice was made on which steering and orientation angles to use in the B-matrix of the first observer (equation: 3.10). The steering angles are only provided by the can bridge measurements and can bridge commands. As it is desired to stay close to reality, the measurements are taken directly. Then, the orientation angle is retrieved from the advanced navigation GPS unit. It could have been taken from the IBEO GPS or IBEO localization as well, but because these use the advanced navigation GPS as an input and fuse this with their own information, the choice is made to stay as raw as possible and use the advanced navigation GPS output as an input to this B-matrix.

In the Simulink model, the approach using two observers looks as follows (Figure 3.20):

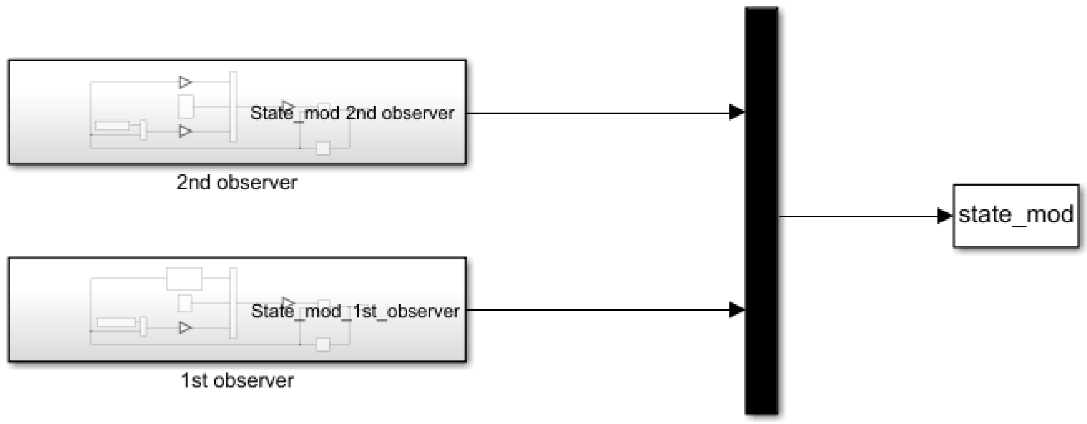


Figure 3.20: Two observer model in Simulink.

The next two Figures, 3.21 and 3.22 contain a detailed overview of the 2nd and 1st observer blocks respectively. A complete overview of all the subsystems for the two observers is included in Appendix I).

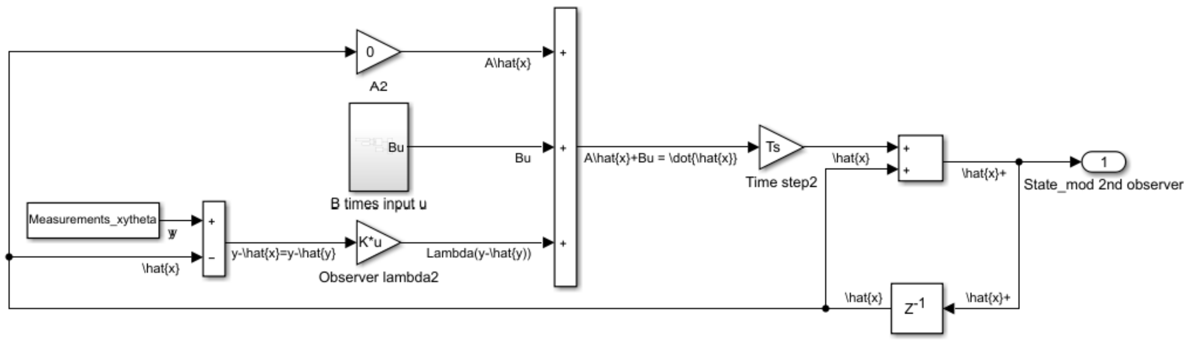


Figure 3.21: Second observer model in Simulink.

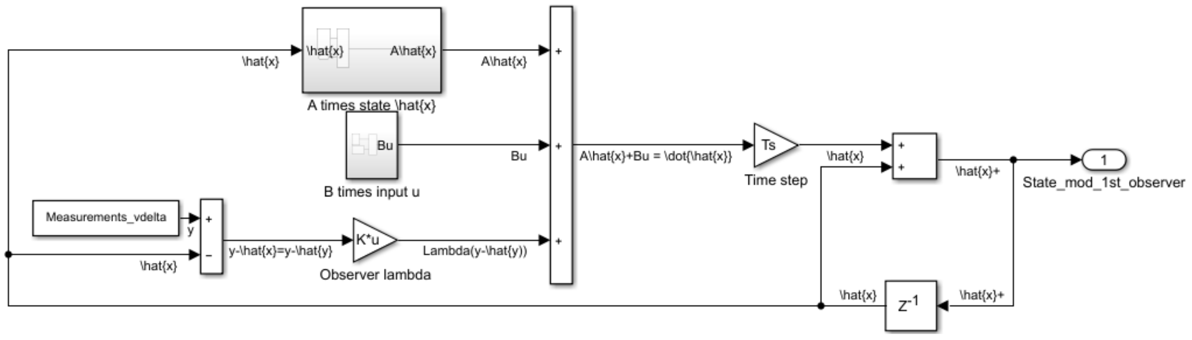


Figure 3.22: First observer model in Simulink.

Residual as difference in state output

After having split the model into two observers, the residuals indicating the difference between the model and the measurements are drawn up again for each of the state entries and shown in Figure3.23. The residuals for the localization (x, y and θ) are indeed smaller, as is visible in Figure 3.23 compared to Figure 3.17. This update is beneficial for the detectability of the faults. However, it is visible that at the second part of the log, there is a constant offset of about 0.2 above 0 in the x-position. This makes it hard to detect peaks and therewith faults over the whole scope of the residuals during one log. Therefore, several elaborations in computation of the residuals are made. The next three subsections are aimed to describe the following three approaches that are more in-depth than only taking the difference between the state outputs:

- Residual as difference in speed based on measurements only
- Residual as derivative of residual in position based on model and measurements output
- Mahalanobis distance as residual in order to statistically evaluate the previous residual

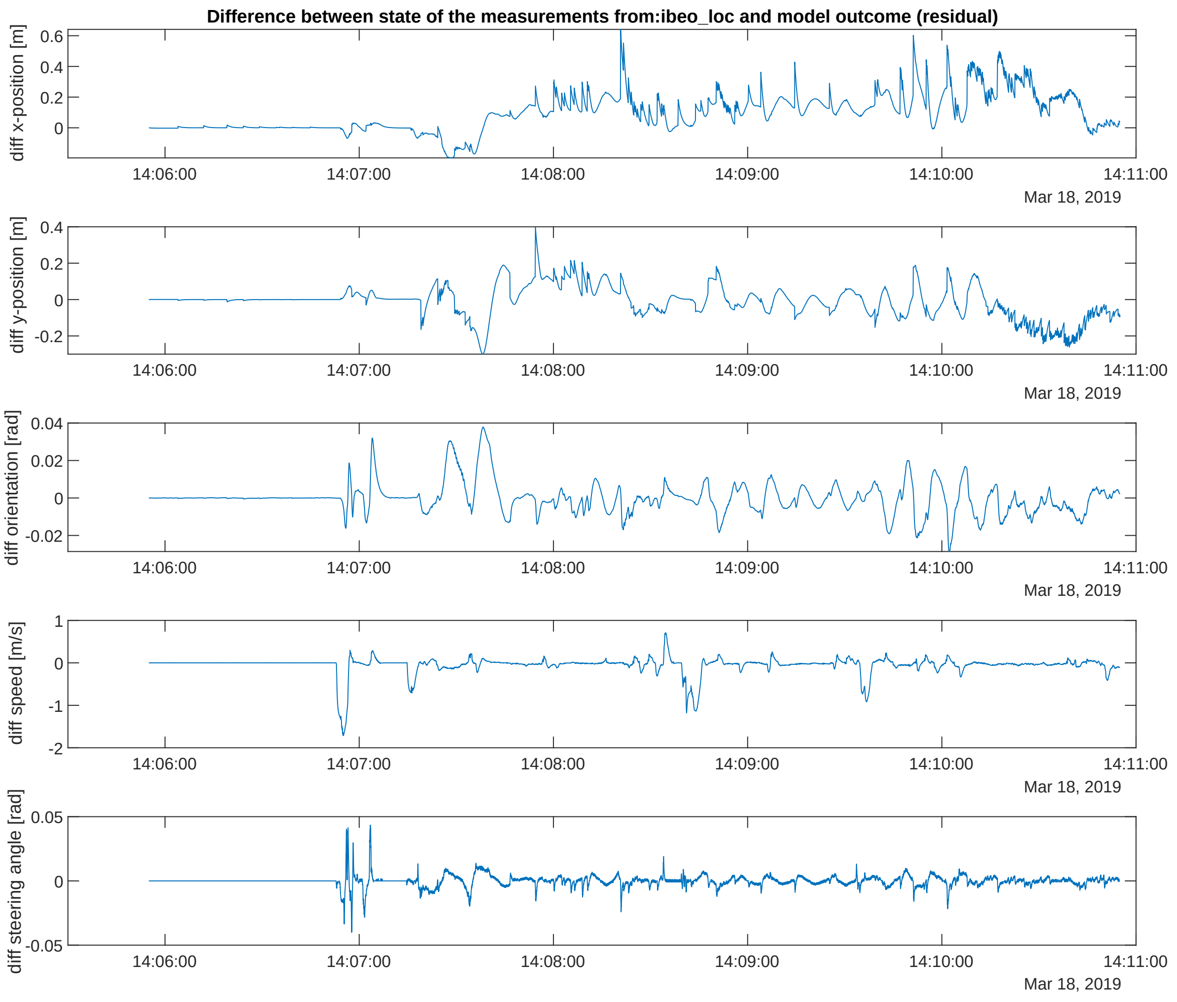


Figure 3.23: Residuals in all state entries (x, y, θ, v, δ) model vs. sensor two observer model.

Residual as difference in speed

The first approach to elaborate on the residual computation is to compute derivatives of the position resulting from the model (i.e. the speeds v_{comp}) and compare these with the speed measurements (v_{meas}):

$$v_{comp} = \sqrt{\Delta x^2 + \Delta y^2} / \Delta t. \tag{3.11}$$

$$r = v_{comp} - v_{meas} \tag{3.12}$$

Taking this approach is a form of applying a first order filter. One thing that has to be taken into account, is that the computed speeds will always be smaller than the actual speeds, because not the entire curvature is taken into account, but it is approximated only by small increments. This is valid to do, because the increments are small enough (i.e. we have enough data points). Fitting the ‘known’ curvature to the data could be a solution for obtaining more accurate results, if desired. First, an evaluation is made on the speeds as they have been computed by taking the increments of just 0.01 seconds, which seems already small enough. The result is shown in Figure 3.24.

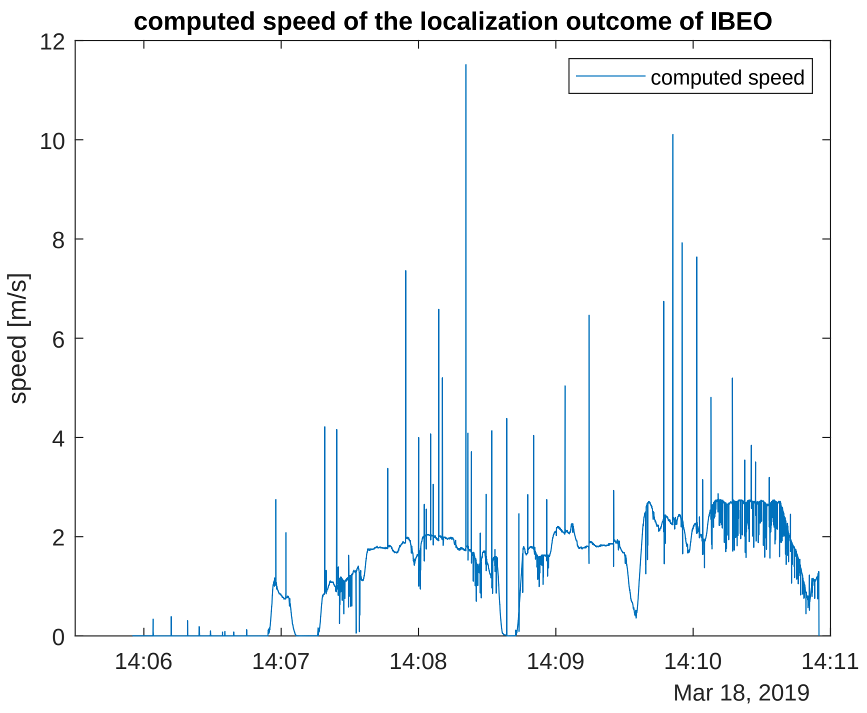


Figure 3.24: Speeds computed on localization output for position.

There are large peaks visible in the computed speeds shown in Figure 3.24. These peaks indicate the instances where a fault in the localization is present. Because of jumps in the positions, the derivative at that point is very high, so the visualization of the faults will be enlarged. A residual can be drawn up to subtract the actual speed from the computed speed, so that the faults remain nicely centered around 0, recall equation 3.12. The computed speeds can be compared (i.e. a residual can be drawn up) with the speed output of another sensor. The choice is made to do this comparison with the GPS output, as these seem to represent reality best. Recall that the modeled speed turned out to perform unexpectedly due to the difference between input speed (canbus) and actual measured speed (canbus) and the fact that there are some unknown effects emerging from the control. Also, it does not make much sense to compare a faulty sensor with its own output for another state entry (i.e. use the IBEO localization speeds). When plotting the both of them against each other, it turned out that the speed measurements from the advanced navigation GPS sensor are delayed with around 1 second, see Figure 3.25. Therefore, the computed speeds are delayed before subtracting them (i.e. before computing the residual). The resulting residual is shown in Figure 3.26.

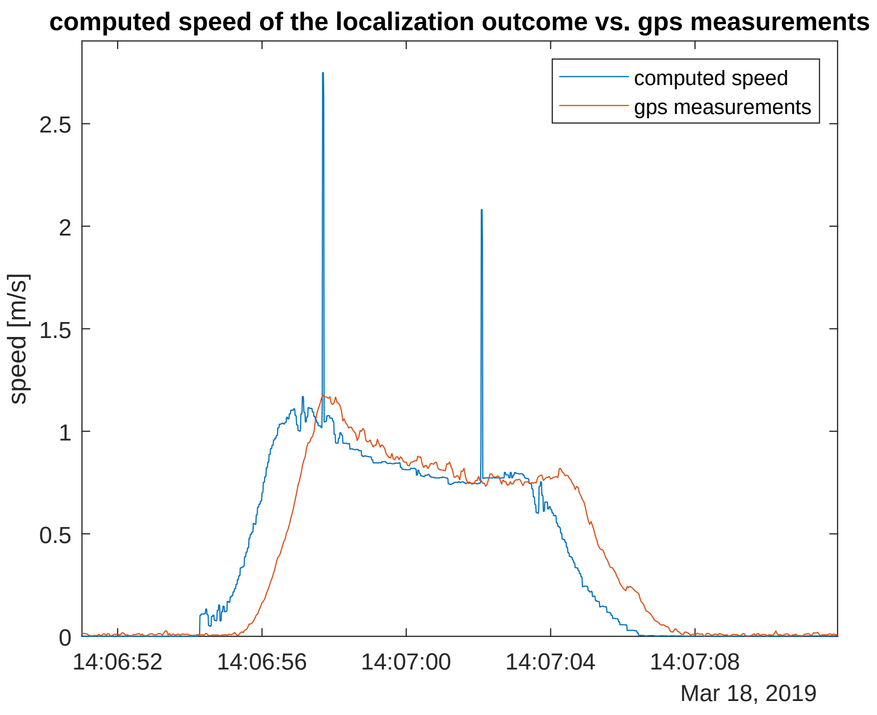


Figure 3.25: Speeds computed on localization output for position vs. 1 [s] delayed GPS measurements.

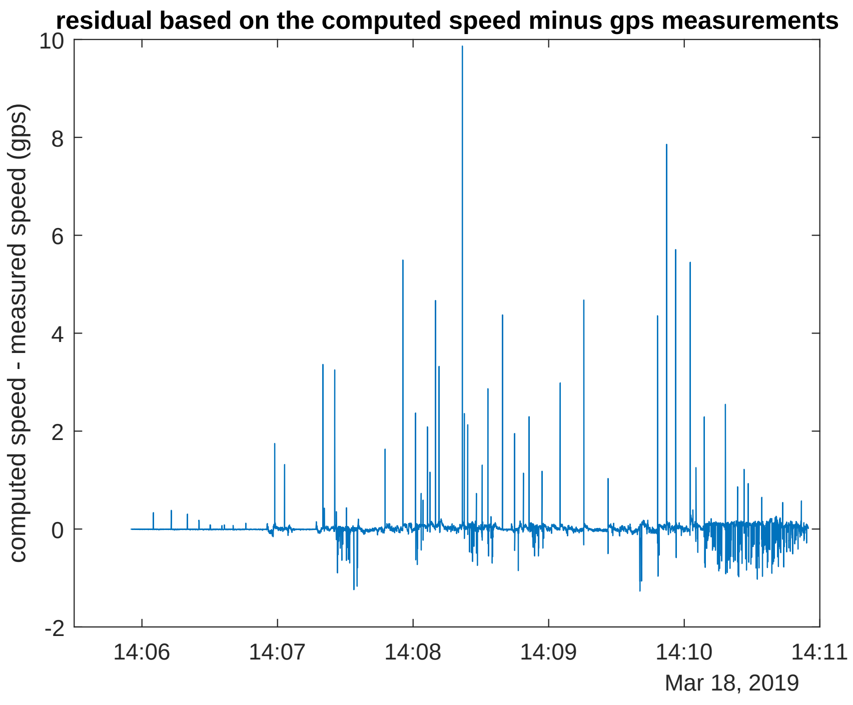


Figure 3.26: Residual of speeds computed on localization output minus shifted GPS measurements.

The peaks that show up in Figure 3.26, indicate now the fault occurences. Because it is nicely centered around 0, this residual would be very well eligible for fault detection when the absolute value of it is compared to a threshold. This approach is actually not model-based anymore, since only the measurements are taken in consideration. It is classified as a parity-checking approximation. A drawback of this method is that it only holds if, in this case, the GPS measurements are assumed to be always correct, which is clearly not always the case, because there could be many influencing external factors [22]. A second drawback of this approach is that the results will be delayed with one seconds when implemented on-line, because the GPS measurements have a delay of around 1 second. It depends on the desired time window in which the fault needs to be detected if this method is applicable or not. A similar approach, that does take the model into consideration so this uncertainty and immediate on-line computation inability can be avoided, can be defined as taking the derivative of the residual on the position.

Residual as derivative of residual in position

Instead of comparing the derivatives of the position from the IBEO localization with the measured GPS speed, another approach is to first compute the residual based on the position output of the model and the measurements of the IBEO localization and then take the derivative of that as a new residual. This can be seen as applying a simple first order filter to the residual. Another research that applied a filter on the output, however before computing a residual, can be found in [9]. There, a a stable linear time invariant (LTI) filter is applied to each of the measurement outputs and it is proved that it improved the detectability of faults. Note that in our case, the residual (i.e. difference between localization model output and measurements) is taken as an input to the filter, rather than only the measurements itself.

As a next step, the residual dynamics have to be computed. This is important because for threshold computation, the residual should be bounded in healthy condition [12]. The residual dynamics are derived as follows (recall: $r(k) = y(k) - \hat{y}(k)$):

$$\begin{array}{rcl} r(k+1) & = & y(k+1) - \hat{y}(k+1), \\ & = & f(x(k),u(k)) + \eta(k) + \xi(k+1) - \big(f(y(k),u(k)) + \Lambda(\hat{y}(k) - y(k))\big), \\ & = & f(x(k),u(k)) + \eta(k) + \xi(k+1) - f(y(k),u(k)) - \Lambda(\hat{y}(k) - y(k)), \\ & = & f(x(k),u(k)) + \eta(k) + \xi(k+1) - f(y(k),u(k)) + \Lambda(y(k) - \hat{y}(k), \\ & = & f(x(k),u(k)) + \eta(k) + \xi(k+1) - f(y(k),u(k)) + \Lambda r(k), \\ & = & \Lambda r(k) + f(x(k),u(k)) - f(y(k),u(k)) + \eta(k) + \xi(k+1), \\ & = & \Lambda r(k) + \delta(k) \triangleq \Sigma(r(k),\delta(k)). \end{array} \tag{3.13}$$

In which the observer matrix Λ can be replaced by the gain values λ (as it is a diagonal matrix: $\Lambda = \text{diag}\{\lambda_i\}$). $x(k)$ represents $y(k) - \xi(k)$, so uncertainty δ becomes:

$$\delta(k) \triangleq f(y(k) - \xi(k), u(k)) - f(y(k),u(k)) + \eta(k) + \xi(k+1). \tag{3.14}$$

This means that the measured dynamics are subtracted from the true dynamics (modelled). Then the non-linear dynamics of the vehicle which were not modelled, end up in η and the measurement noise is indicated by ξ. η is assumed to be 0 in the model, since it represents uncertainty noise whereof the way of computating is unknown. An upper bound $\bar{\eta}$, however, can be defined as three times the variance (σ) of the measurement noise of the sensor that is used.

Instead of using the residual directly, the suggested approach takes the time derivative of this residual, therefore leading to a new residual, r'. A short notation is used for $r(k+1)$ as r_+, $r(k)$ as r and $r(k-1)$ as r_-:

$$r' = r_+ - r. \tag{3.15}$$

λ represents the observer gain used in the model. The actual residual dynamics are now equal to:

$$\begin{array}{rcl} r' & = & \lambda r + \delta - (\lambda r_- + \delta_-), \\ & = & \lambda r + \delta - \lambda r_- - \delta_-, \\ & = & \lambda(r - r_-) + \delta - \delta_-, \\ & = & \lambda r'_- + \delta - \delta_-. \end{array} \tag{3.16}$$

An upper bound can be written on the residual, which is denoted by $\bar{r}'_+$, as in the following:

$$\bar{r}'_+ = \lambda \bar{r}' + 2\bar{\delta}. \tag{3.17}$$

Where $\bar{\delta}$ is an upper bound on δ and the fact is used that in the worst case δ and δ_- have opposite signs. The upper bound uncertainty $\bar{\delta}$ is defined as the maximum on both the model noise η and measurement noise ξ:

$$\overline{\delta}_{(i)}(k) \triangleq \max_{\eta} \max_{\xi} | \delta_{(i)}(k) |. \tag{3.18}$$

In order to upper bound the absolute value of each component of the residual, advantage can be taken of the triangular inequality [12] in the following way:

$$\begin{aligned} | r_{(i)}(k+1) | &\leqslant \lambda_i | r_{(i)}(k) | + | \delta_{(i)}(k) |, \\ &\leqslant \lambda_i | r_{(i)}(k) | + \bar{\delta}_{(i)}(k). \end{aligned} \tag{3.19}$$

Then, by using the Comparison Lemma [14], the adaptive threshold can be defined as:

$$\bar{r}_{(i)}(k+1) = \lambda_i \bar{r}_{(i)}(k) + \bar{\delta}_{(i)}(k) \geqslant | r_{(i)}(k+1) | . \tag{3.20}$$

Statistical evaluation of residual

When the residuals are obtained, one more step can be taken before proceeding to the threshold computation. This step is a statistical evaluation of the residuals. By taking this step, the detectability of a fault will be improved and an assessment can be made about with which probability a fault will be detected. There are several approaches that can be used [11]:

- **Probability density function (pdf) of the Gaussian fit**
  This test aims to determine under a computable certainty whether a certain sample is produced by the nominal probability density function. If this hypothesis is rejected, a fault is detected;
  Pdf of a Gaussian is assumed to have zero mean for the sensors, otherwise a constant offset is visible;
  Prerequisite: normal distribution of the residual;
  Will lead to: tighter threshold;

- **Student t-test**
  Testing the mean (when there is a change in mean, we assumed it to be 0);
  Prerequisite: normal distribution of the residual and computation of the mean and variance;
  Will lead to: tighter threshold;
  Computed as follows:
  $$t = \frac{\hat{\mu}(N) - \mu}{\hat{\sigma}/\sqrt{N}}. \tag{3.21}$$
  With N as a window of a certain number of samples z that is large enough, but also not too big. The samples z are in this case the residuals that are evaluated. The sample mean $\hat{\mu}(N)$ is defined as follows:
  $$\hat{\mu}(k) = \hat{\mu}(k-1) + \frac{1}{k}[z(k) - \hat{\mu}(k-1)]. \tag{3.22}$$
  The variance $\hat{\sigma}$ is determined as follows:
  $$\hat{\sigma}^2(k) = \frac{k-2}{k-1}\hat{\sigma}^2(k-1) + \frac{1}{k}[z(k) - \hat{\mu}(k-1)]^2. \tag{3.23}$$

- χ^2
  Testing the variance (when there is a change in covariance assumed/observed);
  Prerequisites: normal distribution of the residual and computation of the variance;
  Will lead to: tighter threshold;
  Computed as follows:
  $$\chi^2 = \frac{(N-1)\hat{\sigma}_1^2}{\sigma_0^2}. \tag{3.24}$$

- **Mahalanobis distance**
  The Mahalanobis distance represents how far a sample (z) is away from the nominal distribution. Computation can be done for data of which the covariance noise is known in form of a covariance matrix C;
  Prerequisite: none regarding the type of distribution (however, more suitable for multivariate data), computation of the mean and known covariance matrix;
  Will lead to: over-conservative threshold;
  Computed as follows:
  $$d(z) = \sqrt{(z-\mu)^T C^{-1} (z-\mu)}. \tag{3.25}$$

Because the decision was made to proceed with the residual that is computed by differentiating the residual (i.e. difference) of the localization output for the x- and y-position of the model and the measurements (recall: $r' - \lambda r'_{-}$), it will be checked for these residuals whether they are normally distributed or not. This is done by computing histograms for the residuals and compare the result with a Gaussian fit. They are drawn up by using the *histogram* and *histfit* function in MATLAB in the master m-file *run_Simulink.m*. This m-file is excluded from the appendices for the sake of brevity. The histograms are shown in Figure 3.27.

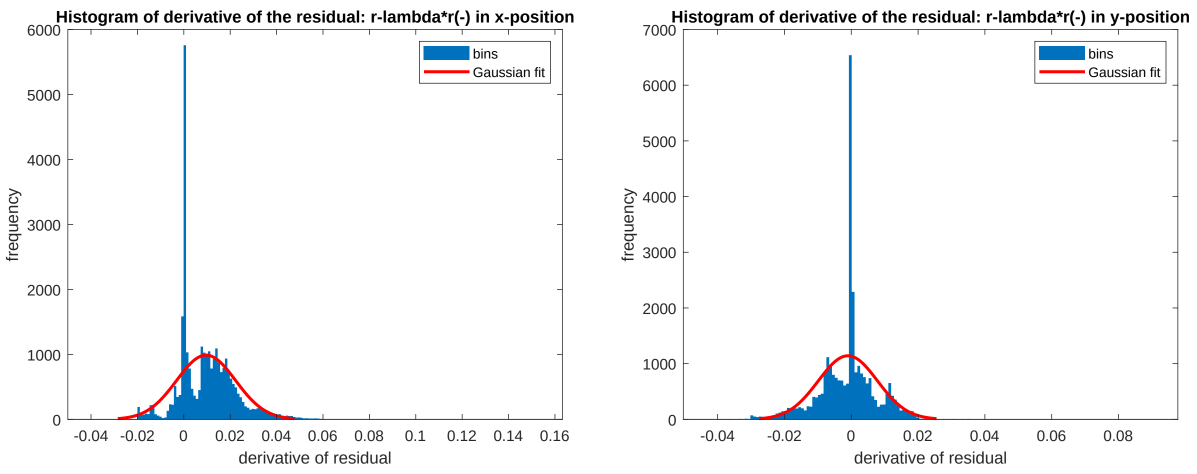


Figure 3.27: Histograms of the derivative of the residual in x-position (left) and y-position (right.)

As the residuals turn out not to be distributed normally, the Mahalanobis distance will be calculated, because this holds for every type of distribution. A big advantage of this approach is also that it can be incorporated in an on-line fashion (when zero mean is assumed). Because the other probabilistic tests require a sufficiently high number of samples, the outcomes can only be computed off-line (although computation of the mean can be done in a recursive way if desired). Other solutions that have an on-line character are likelihood based algorithms [11], but these are left out of the scope of this research.

Mahalanobis distance

The Mahalanobis distance on the 'derivative of the residual' can now be calculated. Recall equation 3.25. The samples z are now represented by a 2x1 vector containing the residual: $r' - \lambda r'_{-}$ in x-position and y-position for each time step and the mean μ is assumed to be zero (otherwise the sensor has a constant offset). The covariance matrix is a symmetric matrix that is defined as follows:

$$C = \begin{bmatrix} cov\_xx & cov\_xy \\ cov\_xy & cov\_yy \end{bmatrix} \tag{3.26}$$

The IBEO localization sensor provides a measurement of these three covariances for each sampling time. However, these covariances are given in the local coordinate frame of the vehicle. Therefore, it is required to include a rotation over the orientation angle θ_i of the vehicle at that time instance:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{3.27}$$

Rotating a matrix can be done using the following approach: RCR'. Instead of rotating the covariance matrix, it is also possible (and easier) to rotate the samples (z) and fill in the inverse of C directly:

$$z'(RCR')z = z'R'C^{-1}Rz = (Rz)'C^{-1}Rz. \tag{3.28}$$

Therefore, the actual implementation of the Mahalanobis distance becomes:

$$d = \sqrt{(Rz)'2C^{-1}Rz}. \tag{3.29}$$

As is visible in equation 3.29, two times the covariance matrix is used. This is in order to achieve the upper bound defined before (recall $2\bar{\delta}$). The covariance represents the measurement noise ξ. Because the noise

η that was not modelled, is set to 0, the covariance can be used as δ immediately. All can be filled in and computed off-line based on the logged information. Because the covariance is returned by IBEO every time step, this could be easily extended to an on-line implementation as well, as still only information at each time instance is used. In order to use the covariance of a single realization of a stochastic process in place of the covariance over time, the stochastic process is assumed to be stationary and ergodic during a single experiment.

The first time the Mahalanobis distance was computed by implementing equation 3.29 in Matlab, it turned out that something was wrong. The outcomes of the Mahalanobis distance were complex numbers. The fact that the covariance matrix is a positive definite matrix by definition [6], indicates that the covariance matrix used for computation is incorrect. Therefore, again some data cleaning had to be performed. The way this, and new emerging inconsistencies, were handled are added to Appendix F. After having all the data fixed in an off-line way for now, the outcome of the Mahalanobis distance compared with the initial residual and the derivative thereof, is shown in Figure 3.28. The new residuals have a smaller offset than the initial residual based on the difference only, which is advantageous in terms of detectability. Also, using the Mahalanobis distance makes detection easier, because the peaks are (at some points at least slightly) higher than the residual that is based on the derivative of the difference. This is due to the incorporation of covariance noises in this approach. They will give an extra boost or hold-back when the noise is low or high respectively.

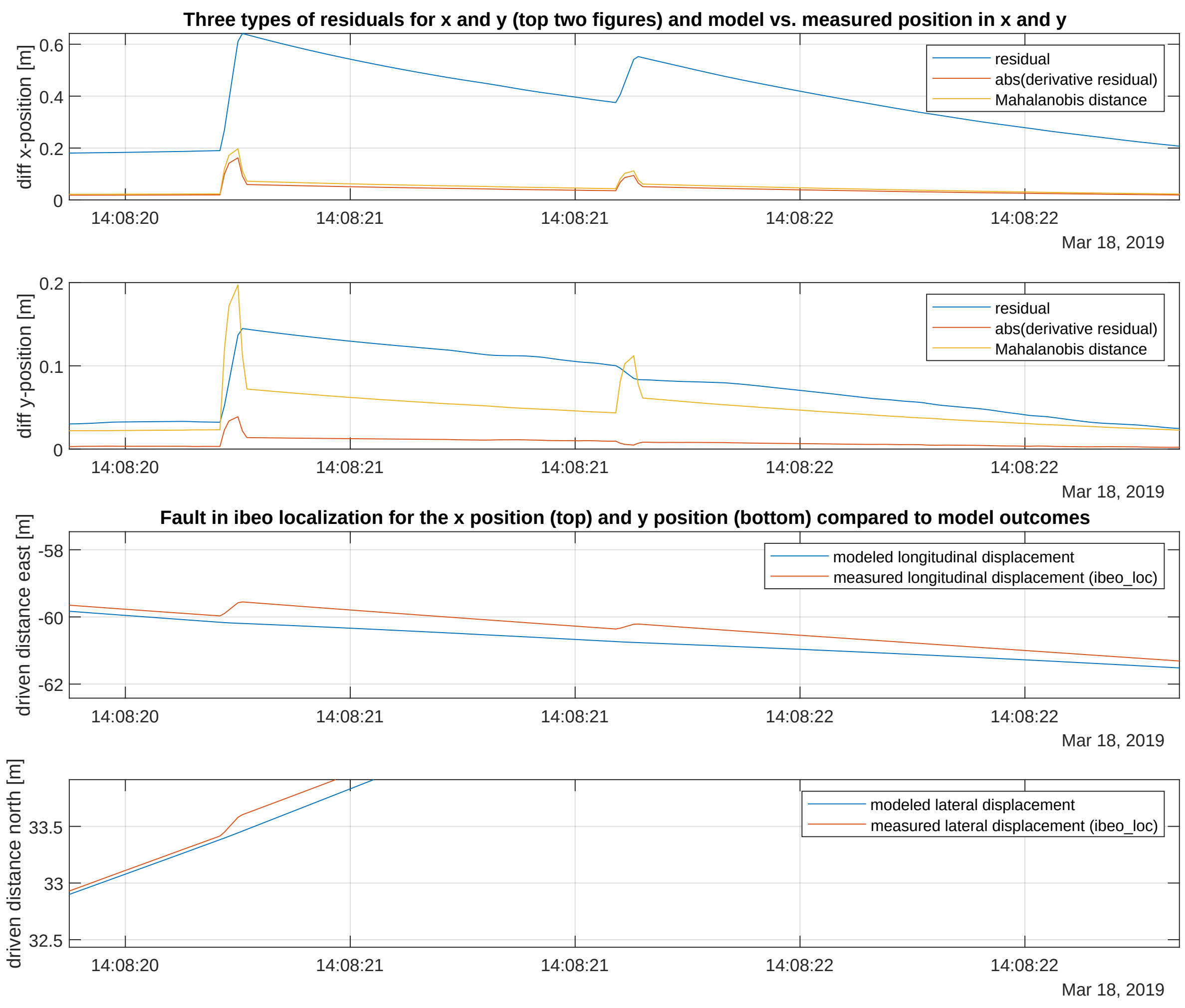


Figure 3.28: Top two figures: residual with its derivative and the Mahalanobis distance thereof, bottom two: localization output.

Threshold design - Chebyshev inequality

Once the Mahalanobis distance is obtained, the threshold $\bar{d}$ can be determined using the Chebyshev inequality as stated below [11].

$$\bar{d} \triangleq \frac{n}{\alpha} \Rightarrow \mathbb{P}[d(z) > \bar{d}] < 1 - \alpha \tag{3.30}$$

This term indicates the probability of having a correct rejection: the probability that sample z for which the Mahalanobis distance has been calculated (in this research z is set equal to the derivative of the residual in x and y, therefore dimension n = 2), has been produced by the nominal distribution. α has to be chosen as a value between 0 and 1 (excluding those bounds), because a probability is determined based on $1 - \alpha$. A scaling factor for the threshold is assumed, because the covariance is taken as the small values only, which are a decade in the order of magnitude smaller than they were initially returned by IBEO. In Figure 3.29, a plot is shown of the Mahalanobis distance versus a threshold that is determined with α equal to 0.01, so probability 1-0.01*100 = 99%. Because it was found that the covariance is subject to some scaling issues (by assessing the covariance of fault-free experimental data), and therewith the Mahalanobis distance outcome is scaled, the value of $\bar{d}$ is scaled by 1e-4.

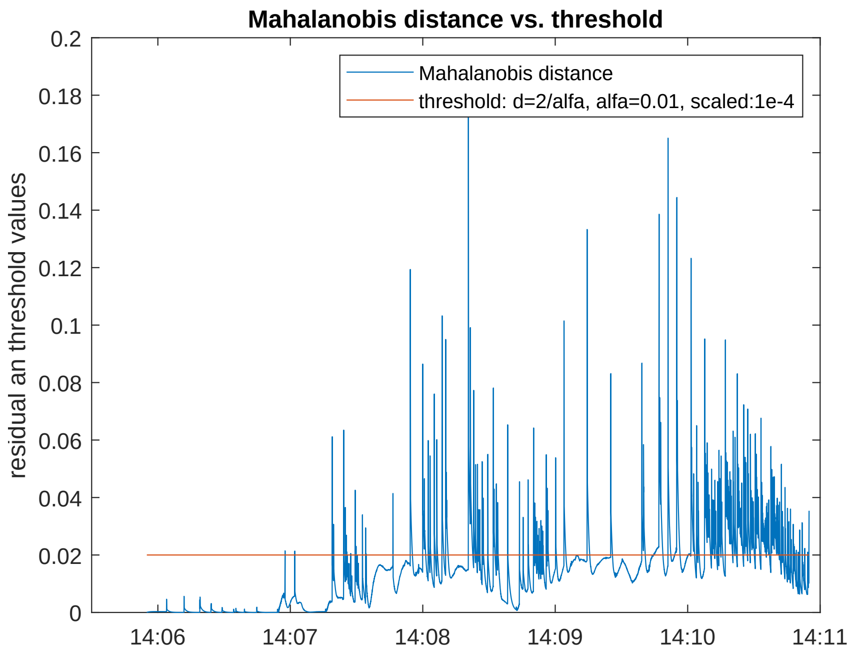


Figure 3.29: Mahalanobis distance versus threshold.

3.4.5. Fault isolation

In order to isolate a fault, multiple single observers (as introduced in section 3.4.2) can be used for each of the four sensors: advanced navigation GPS, advanced WGS84, IBEO GPS and IBEO localization. A combination of each of the sensors can be run for generating residuals. When one of the residuals appears to be significantly lower than the other combination of sensors, the sensor that is excluded from this particular combination, can be isolated as the faulty one, because it apparently has a negative influence on the overall residual. In progress towards fault tolerant control, this sensor can be temporarily disabled or will be trusted less in comparison to the other sensors, so the path planner will take only valid information into account while computing its next time step. For this master's thesis, the focus has been placed on the detection of faults in the black box localization software rather than isolating the fault first. This is because at this moment, the faulty sensor was already known and finding a solution for detecting faults in this particular sensor and providing information about the if and when of these occurrences is a more pending problem than providing information now that can identify that there is a fault and to isolate which sensor is the faulty one. But as an extension of this research, the fault isolation can be implemented easily with the resources that are already provided by this thesis.

4 Experiments and Results

This chapter discusses the experimental setup of the research based on the methods that were described in Chapter 3 and provides the results. Also, an evaluation is provided that assesses the performance of the overall model-based fault diagnosis approach.

4.1. Experiments

Two experiment cases have been drawn up based on the model-based approach. The first one tests the ability of detecting faults in the IBEO localization module using an empirical threshold in combination with the derivative of the residual. The second one combines the residual that has been determined by the Mahalanobis distance with a threshold emerging from the Chebyshev's inequality. Both cases are tested on three measurement days that showed localization data containing faults: the 5th, 6th and 18th of March, 2019. A baseline is provided for a log in which no faults were present in order to validate that the model does not provide false positives (i.e. provides a message that there is a fault, although there is none [13]). The two experimental cases of the model-based approach are compared to the parity approximation approach in order to determine the detectability rate in the performance section.

Every detected fault is reported uniquely with a message for the purpose of analysis, for example:

"fault in the IBEO localization localization y-position at: 18-Mar-2019 14:08:33"

The detected fault is defined as unique, when it is the first in the sequence to cross a certain threshold. Indices that come directly after, belong to the same fault instancce. This is implemented in the *run_Simulink.m*-file as:

```
idx = find(abs(res)>threshold);
D = diff([0,(diff(idx)==1),0]);
fault = idx(D>0);                % find the 'index of occurence' of a fault
if fault
    faultmess = strcat('fault in the IBEO localization position at:',{' '}, datestr(
        date_and_time+seconds(idx*Ts)));
    secs  = idx*Ts;              % determine time instance of fault occurence
    res_f = res_c(idx);          % determine residual values where the threshold is
        crossed
    unmes = string(unique(faultmess));
    unmes = uniquemes(unmes)     % display unique (one for each time) fault message
end
```

4.1.1. Derivative of the residual combined with empirical threshold

For this experiment, the model and the IBEO localization measurements are used. The outputs of the position in the x- and y-direction will be subtracted as a definition of the residuals:

$$r_x = x_{pos,IBEO} - x_{pos,model}, \tag{4.1}$$

$$r_y = y_{pos,IBEO} - y_{pos,model}. \tag{4.2}$$

The derivative is taken of these residuals in order to retrieve the new residuals. Actually, the derivative is not a strict derivative, because the time difference is subjected to observer gain λ. However, this approach will still be named 'derivative of the residual', because it summarizes the used method best.

$$r'_x = r_{x,+} - \lambda r_x, \tag{4.3}$$

$$r'_y = r_{y,+} - \lambda r_y. \tag{4.4}$$

The implementation in MATLAB looks as such:

```matlab
lambda = 0.9;
residual_x = tr(:,1)-x_mod;          % compute the residual in x-position
difres_x = diff(residual_x)';        % compute direct derivative of the residual x
for i = 2:length(residual_x)
    z(i-1)=residual_x(i)-lambda*residual_x(i-1);      % include observer gain
end
z(length(tr)) = z(length(tr)-1);
difres_x = z;                        % overwrite the derivative of the residual x
residual_y = tr(:,2)-y_mod;          % compute the residual in y-position
difres_y = diff(residual_y)';        % compute direct derivative of the residual y
for i = 2:length(residual_y)
    z(i-1)=residual_y(i)-lambda*residual_y(i-1);      % include observer gain
end
z(length(tr)) = z(length(tr)-1);
difres_y = z;                        % overwrite derivative of the residual y
```

An empirical threshold is defined by inspection of the residual and set to 0.03. This indicates that the rate of change of the model needs to be higher than 0.03 in order to exceed the threshold. Therefore, quick changes in the position will be detected as a jump and thus a fault. Jumps of 9 cm and more can be detected when the algorithm uses a threshold of 0.03, see Figure 4.1.

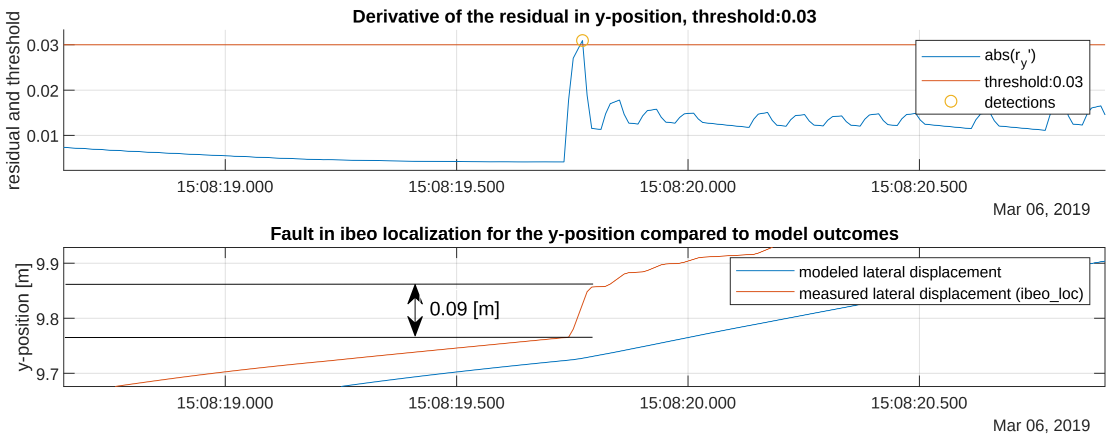


Figure 4.1: Empirical threshold determination for residual as derivative of the offset.

A visualization of the absolute value of the residual in x- and y-direction versus the threshold, is provided in Figure 4.2. This shows the barrier which the residual has to exceed in order to be detected as a fault. The detections itself are shown in section 4.2.1.

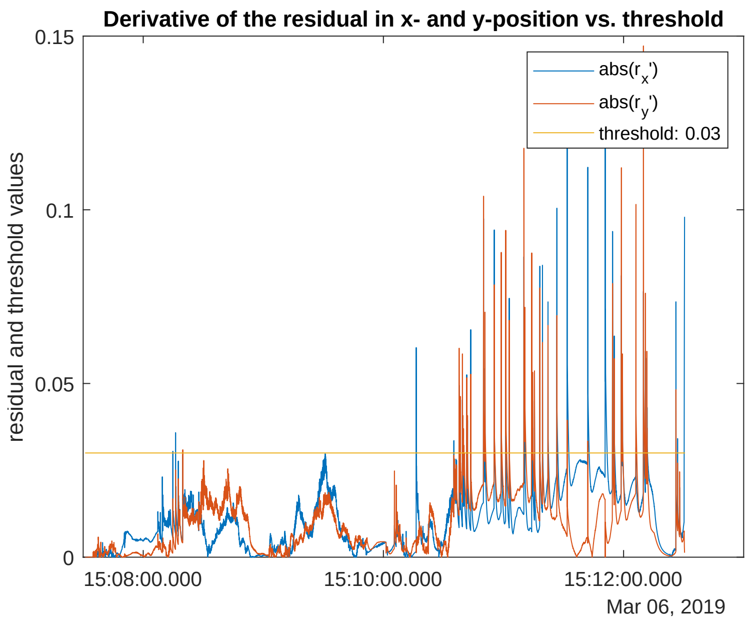


Figure 4.2: Absolute derivatives of the position differences as residuals vs. empirical threshold of 0.03, March 6, 2019.

4.1.2. Mahalanobis distance residual combined with Chebyshev threshold

This experiment is based on an adaptation of the previous residual. The r_x' and r_y' are combined by means of the Mahalanobis distance into one new residual, r''. An adaptation to the conventional definition as introduced in equation 3.25 is made in order to be able to use it for the application at hand:

$$r'' = \sqrt{(Rz)'2C^{-1}Rz}. \tag{4.5}$$

Z is a 2x30,000 vector (based on the length of the inputs) containing the r_x' as the first entry and r_y' as the second entry. R is a rotation matrix based on the orientations obtained by the measurements. C is the covariance matrix containing the noise of the position measurements in x and y direction (i.e. the localization output). The implementation in MATLAB is shown below, in which *covmin* is the function written in order to obtain the correct covariance values, as explained in section 3.4.4, see also Appendix F.

```
cov_min = covmin(xx,yy,xy); % calculate covariance matrix based on local minima cov

for i = 1:length(z)
    d_min(i) = sqrt(([cos(theta(i)), -sin(theta(i));
                      sin(theta(i)), cos(theta(i))]*[z(:,i)])'*2*inv(cov_min{i})*...
                     [cos(theta(i)), -sin(theta(i));
                      sin(theta(i)), cos(theta(i))]*[z(:,i)]);
end
```

The threshold is defined based on the Chebyshev inequality. A slightly higher probability is desired than the 99% that was drawn up in section 3.4.4 and is set to 99.33%, Therewith, a detection threshold of 1e-4*2/0.0067 = 0.02985. In Figure 4.3, the residual as the Mahalanobis distance is shown versus this Chebyshev threshold. The instances where the residual exceeds the thresholds, fault detection will take place. The actual detections are shown in section 4.2.2.

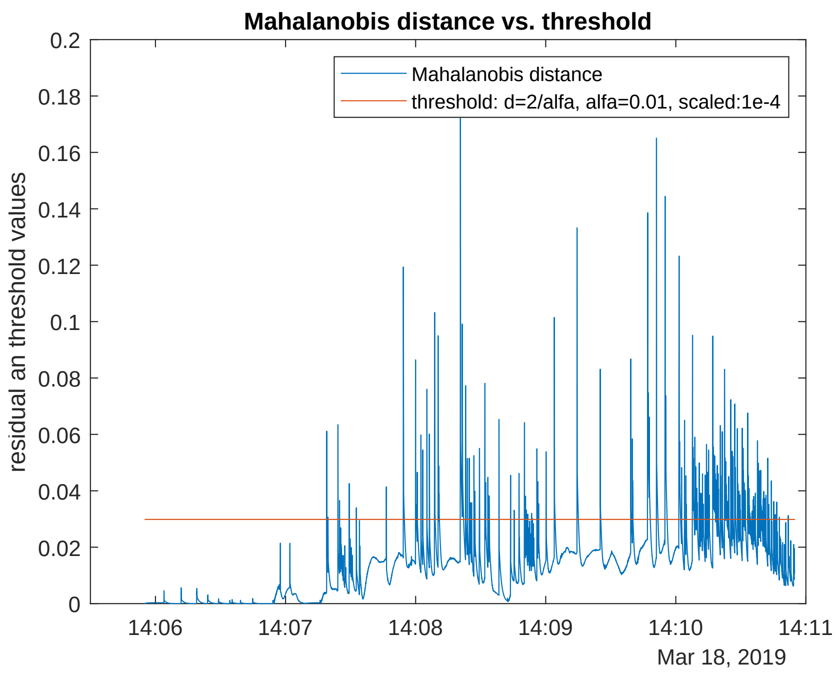


Figure 4.3: Mahalanobis distance residual versus threshold, March 18, 2019.

4.1.3. Baseline

It is important to ensure that the proposed approach does not generate false positives, i.e. outputs a message that a fault occurred, although in reality, no fault was present. In this case, the vehicle might aim to adapt its behavior or come to a safe stop, although this would have been unnecessary. This way, the algorithm might pose more disadvantages than it improves the overall level of autonomy, at least delay in the overall time for completing the track is introduced. This has to be avoided. A reference log is chosen where no big jumps in the localization data occurred (i.e. that performed fault-free). The chosen log is also on the 6th of March, but at the beginning of the day. Generally it turned out while testing, that the localization faults mainly occurred later during the day. First, a plot is included as Figure 4.4 that shows the difference between the model and the measurements. Although there is sometimes an offset visible, no clear jumps can be distinguished. The peaks might be due to some other noise or variance in the system. It can be visualized by inspecting the covariances. This is done in Figure 4.5 for the covariances in x- and y-direction. Indeed, at mainly 09:05:00, it is visible that there is a drop in the covariance where the offset is small as well.

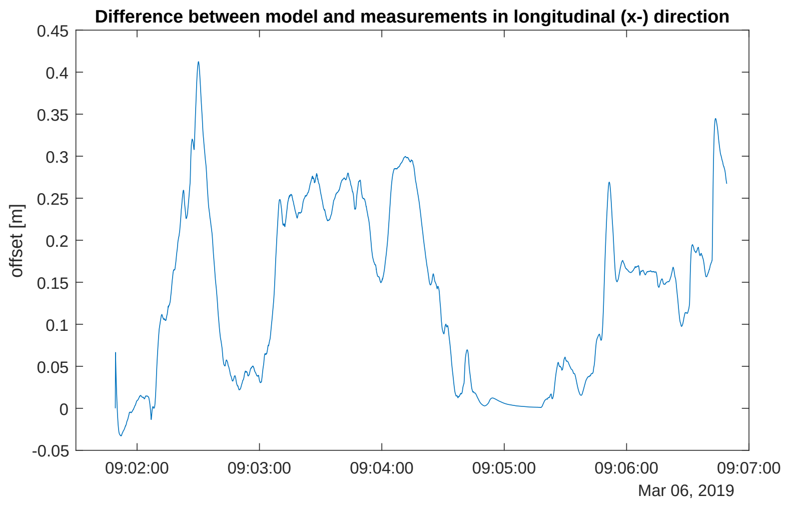


Figure 4.4: Log that is free from jumps in the localization module position (x) output, March 06, 2019.

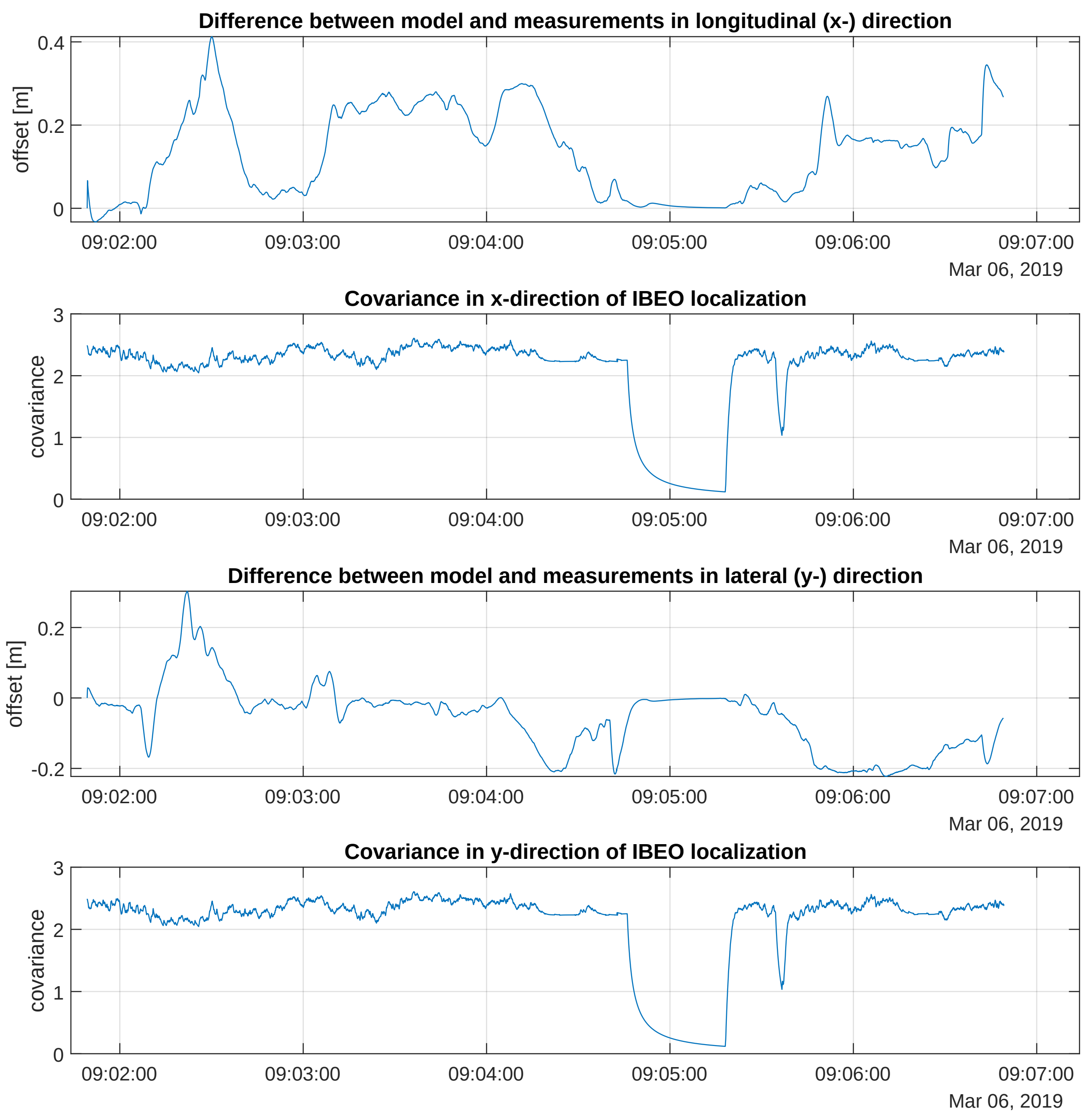


Figure 4.5: Covariances in the x- and y- direction provided by the IBEO localization versus the differences between the model and measurements.

The baseline log has been used to generate the residuals for both the derivative of the residual and the Mahalanobis distance approach and compare them with the according thresholds. When no detections are made, the approach functions as desired. The results are shown in section 4.2.3.

4.1.4. Performance

In order to determine the performance of the developed methods, experiments have been set up that assess this. A measure of validation is used in form of the parity approximation approach as explained in 3.4.4. First, the hit rate will be defined in which it is calculated what percentage of faults can be detected by the model-based approach. Then, the computational speed is computed because it is important to find out if the methods is eligible for an extension towards applying in an on-line manner.

Parity approximation

As a measure of validation, the residual is used that is defined by the difference between the derivatives of the position of the measurements and the speed of the measurements.

$$r = \left(\sqrt{\Delta x^2_{IBEO} + \Delta y^2_{IBEO}} \Delta t\right) - v_{GPS,shifted} \tag{4.6}$$

The implementation is included in the master m-file *run_Simulink.m* and looks as follows:

```
%% Compute speed
dt = diff(tv(1:2));
v_comp = sqrt(diff(tr(:,1)).^2+diff(tr(:,2)).^2)/dt;
v_comp2 = [zeros(105,1);v_comp];
figure
tn = [];
for i = 1:length(tv)+104
    if i >= length(tv)
        tn(i)=tn(i-1)+0.01;
    else
        tn(i)=tv(i)+0.01;
    end
end
res = v_comp2(1:end-104)-tr_adv_gps(:,4);
```

The difference in speeds is used as a residual, because it stays close to what happened in reality. This approach has a better detectability than the other two approaches (identified by inspection of Figure 4.6 of how the residual is distributed around 0 with clear peaks at each fault occurrence. These peaks only represent jumps in the position, because they are defined as the direct derivatives of these positions.

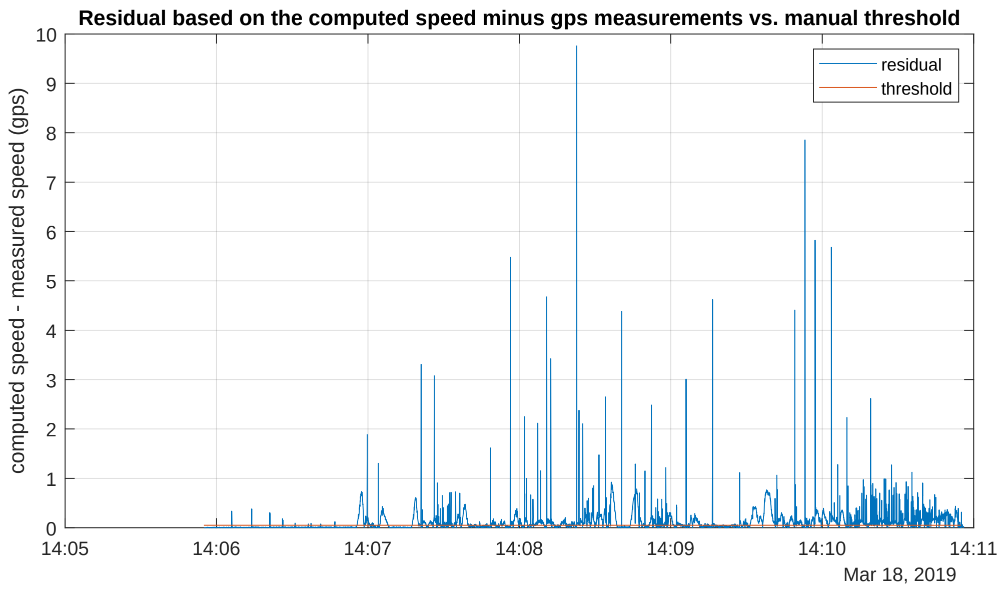


Figure 4.6: Absolute value of parity approximation residual vs. empirical threshold of 0.05, March 18, 2019.

The threshold in Figure 4.6 is randomly picked at 0.05. It is visible that this value is too low. Therefore, again an empirical evaluation has been made, as is shown in Figure 4.6. The value is set to 0.05, therefore able to detect faults with an offset from 2 centimetres upwards. This is determined by checking the jump that was visible from the position output of the measurements on the according first fault occurrence of this sequence. The choice could be made to set a less strict threshold, for example to allow a maximum offset of almost 7.5 centimeters (threshold ~0.5). The way the empirical threshold is determined, is shown in Figure 5.1.

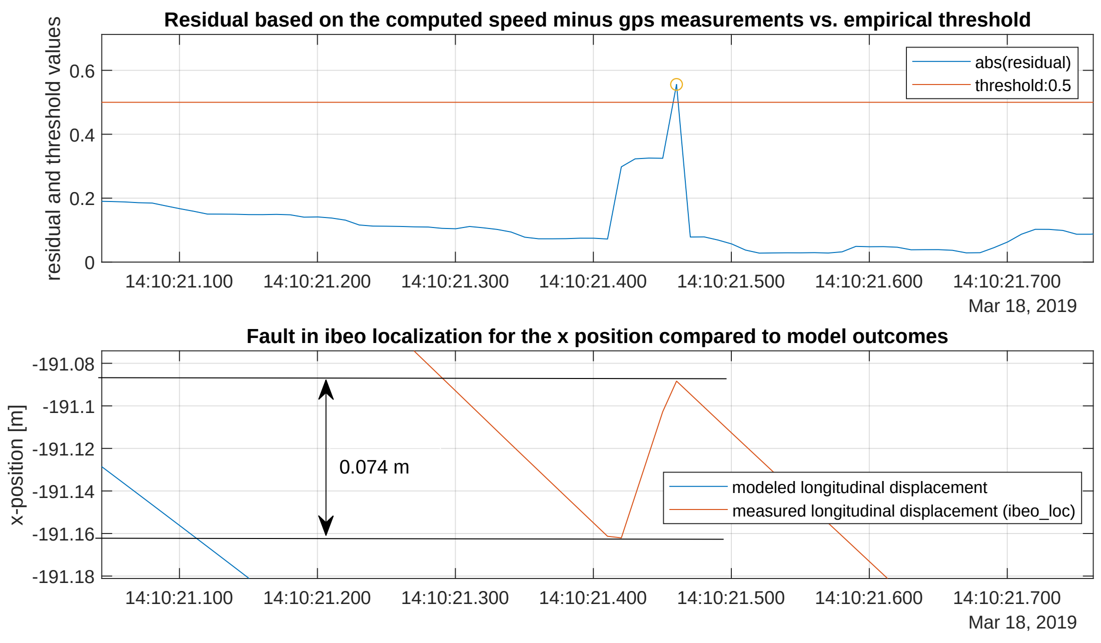


Figure 4.7: Empirical threshold determination for parity approximation approach.

Hit rate

The detection (or: hit) rate can be determined as the amount of true positives (correct classifications) divided by the sum of the true positives (TP) and false negatives (incorrect rejection: the fault has not been classified as such, FN) [13]:

$$\text{Detection rate} = \frac{TP}{TP+FN}. \tag{4.7}$$

The false alarm rate is defined as the amount of false positives (i.e. a false alarm), divided by the sum of the true negatives (correct rejections, TN) and the false positives (FP):

$$\text{False alarm rate} = \frac{FP}{TN+FP} \tag{4.8}$$

The number of detected faults will be compared, so a hit rate percentage can be computed of the model-based approach, assuming that this parity approximation reaches a hit rate of 100% for the given accuracy.

Computational speed

It will be determined for each of the computational steps, how much time has elapsed using the *tic toc* functionality in MATLAB. The sections are divided as follows (and listed along with the responsible m-files if any):

- Read in, convert and save data from .rosbag to .mat files - *read_in_data_from_rosbags.m*: *determine_noise.m*;
- Read in the data from .mat files;
- Prepare the data as input for simulation - *prepare.m*;
- Simulate;
- Derivative of residual - *faultmessage.m*: *uniquemes.m*;
- Mahalanobis distance - *Mahalanobis.m*: *vectorfourtimes.m*, *covmin.m* and *uniquemes.m*;
- Parity approximation - *uniquemes.m*.

The first functionality does not have to be run each time, only once to convert and save new log data from .rosbag to .mat files.

4.2. Results

First, an overview is provided for the detection using the derivative of the residual combined with a empirical threshold in section 4.2.1. Then, the residual based on the Mahalanobis distance which is combined with a Chebyshev threshold is visualized in section 4.2.2. Next, a baseline fault-free log is included to ensure that no false positives are generated in section 4.2.3. Finally, the performance of the model-based diagnosis approach is assessed in section 4.2.4.

4.2.1. Derivative of the residual combined with empirical threshold

First, an image is included in Figure 4.8 that contains a zoomed-in plot of a fault in the localization output for March 5, 2019. The bottom two figures represent the model (blue) and the measurements (red). The measurements show an unexpected jump between 11:16:48 and 11:16:49. The top two figures contain the difference between the model and the measurement outputs for the position in x-direction and the position in y-direction.

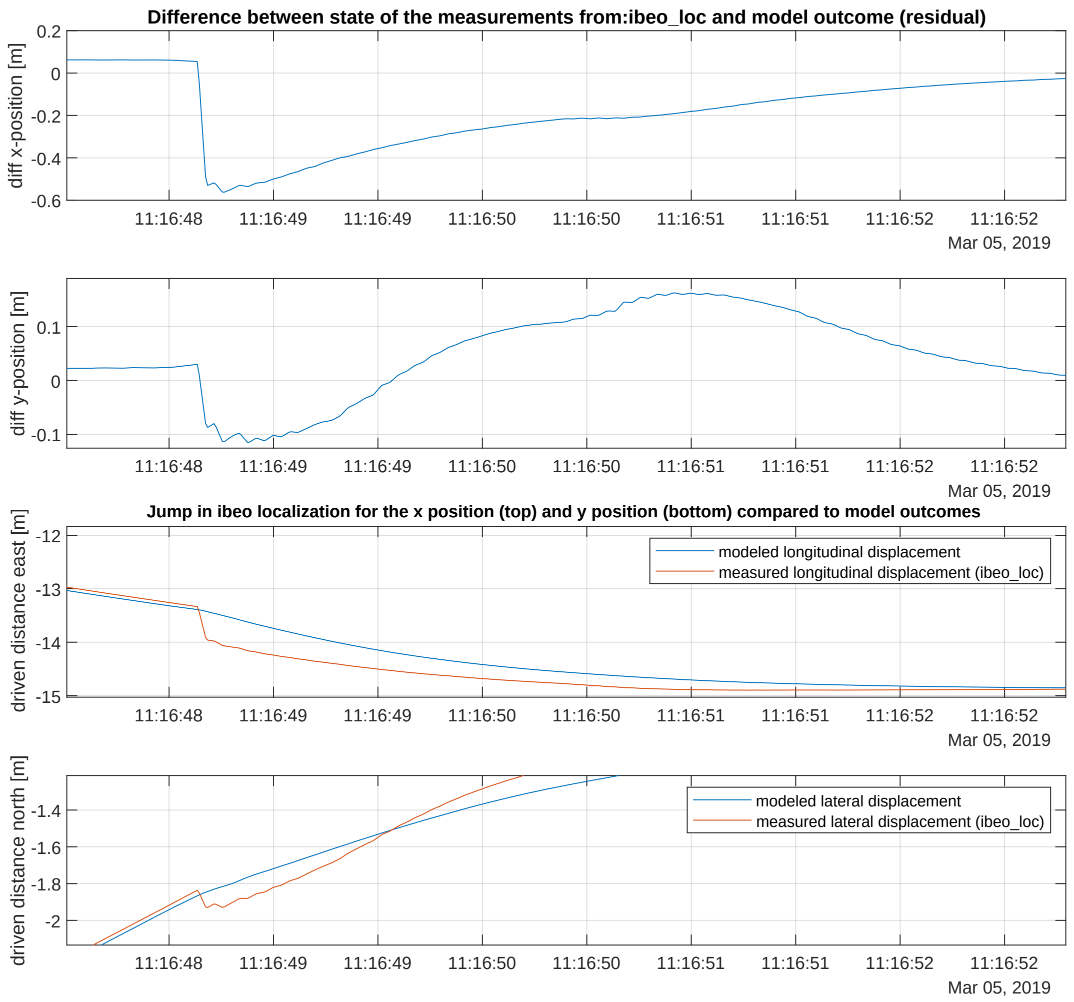


Figure 4.8: Detected fault in IBEO localization March 05, 2019.

The algorithm determines where the residuals exceed the empirical threshold. The instances where this is the case, are indicated by the red dots in Figure 4.9. I.e. at these instances, the derivative is large enough to make the detection. The bottom two figures represent the modelled and measured displacements in addition of black dotted lines. These black lines indicate the initial moment in time of the occurrence of a fault. It is visible that indeed the measurements (red line) performs a jump at the position of the black dotted line. The small sawtooth that is visible in the y-position offset is not detected by the algorithm. This is because these jumps lie in an order of magnitude of centimetres, while the threshold can detect from 9 centimetres upwards (recall section 4.1.1). In Table 4.1, a summary is provided of the date, threshold value and number of detections made. First, the total number of detections (which are represented by red dots in Figure 4.9) is given and then the number of 'unique' detections, defined as the first time a fault occurs (the black dotted lines).

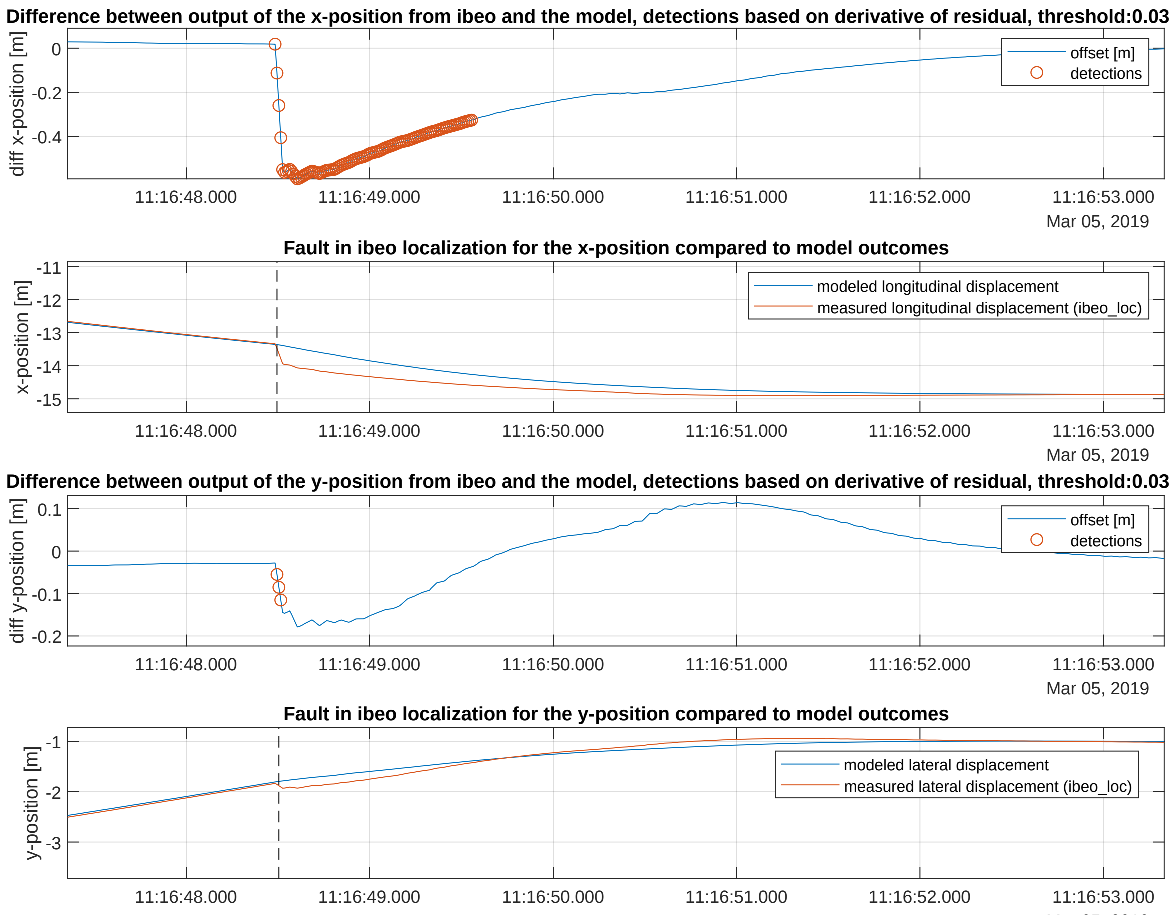


Figure 4.9: Detected fault in IBEO localization March 05, 2019.

Table 4.1: Fault detection ID derivative residual March 05, 2019

Date	**March 05, 2019**
Threshold value	0.03
# faults detected in x-position	108, of which unique: 1
# faults detected in y-position	24, of which unique: 5

A similar Figure is provided for March 6, 2019 in Figure 4.10. Due to the size of the axes, the jumps can not be inspected by eye in the plots showing the x- and y-positions emerging from the measurements vs. the model. However, the derivatives of the positions (residuals) do clearly show the sawtooth behavior. The overview of the fault detection is provided in Table 4.2. Comparing the results of this 6th of May to those of the 5th of May, it becomes clear that the logs of the former contain way more faults and therefore more outings of unexpected behavior by the IBEO localization unit.

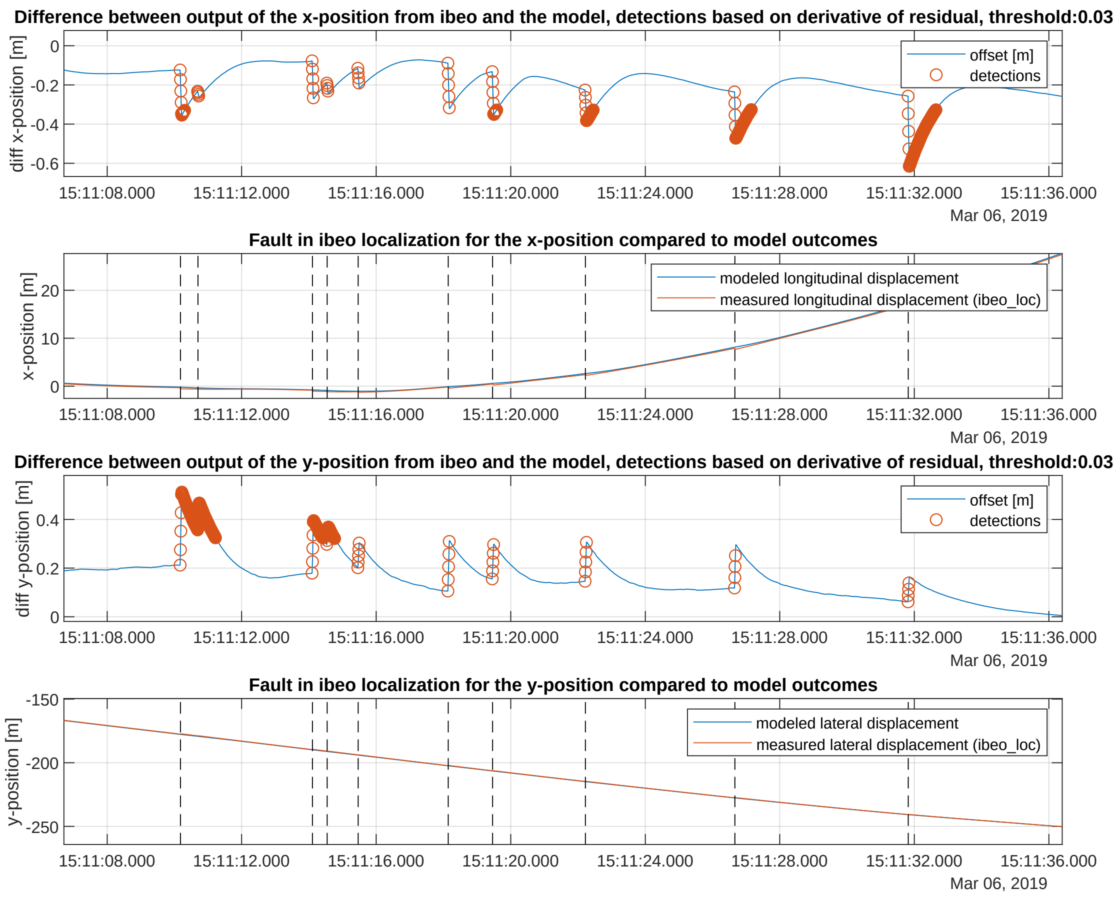


Figure 4.10: Detected fault in IBEO localization March 06, 2019.

Table 4.2: Fault detection ID derivative residual March 06, 2019

Date	**March 06, 2019**
Threshold value	0.03
# faults detected in x-position	715, of which unique: 39
# faults detected in y-position	709, of which unique: 37

Finally, the third testing day is also evaluated with the residual as derivative of the difference between the modelled and measured positions. This is shown in Figure 4.11 and the information is provided in Table 4.3.

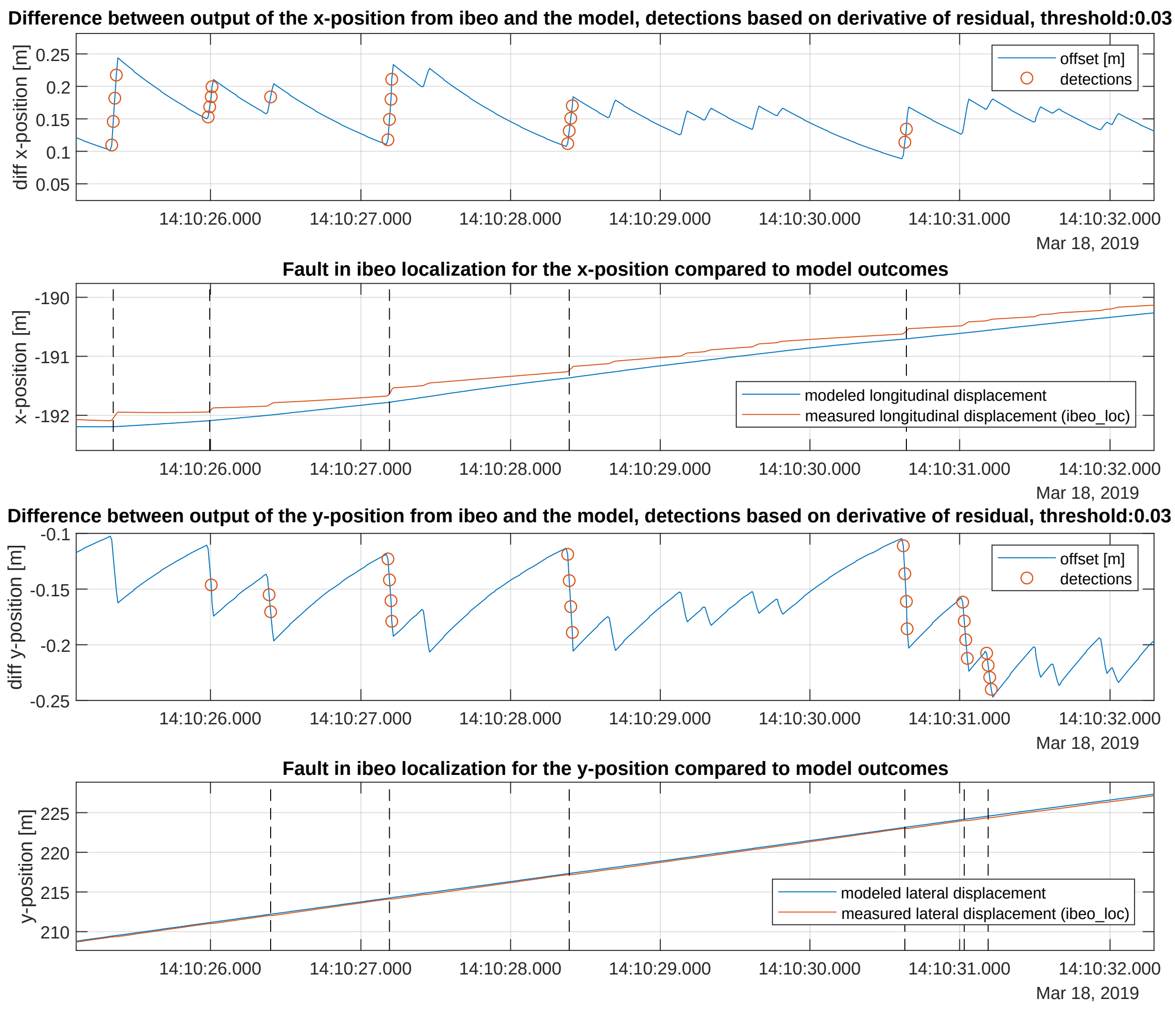


Figure 4.11: Detected faults with threshold 0.03 in IBEO localization March 18, 2019.

Table 4.3: Fault detection ID derivative residual March 18, 2019

Date	**March 18, 2019**
Threshold value	0.03
# faults detected in x-position	776, unique: 82
# faults detected in y-position	96, unique: 25

Before moving on to the next approach (residual based on Mahalanobis distance), the overall fault detection of one log is presented. Figure 4.12 contains the absolute values of the residuals, the threshold and the instances where the residual exceeds the threshold (represented by the yellow dots) for the 18th of March.

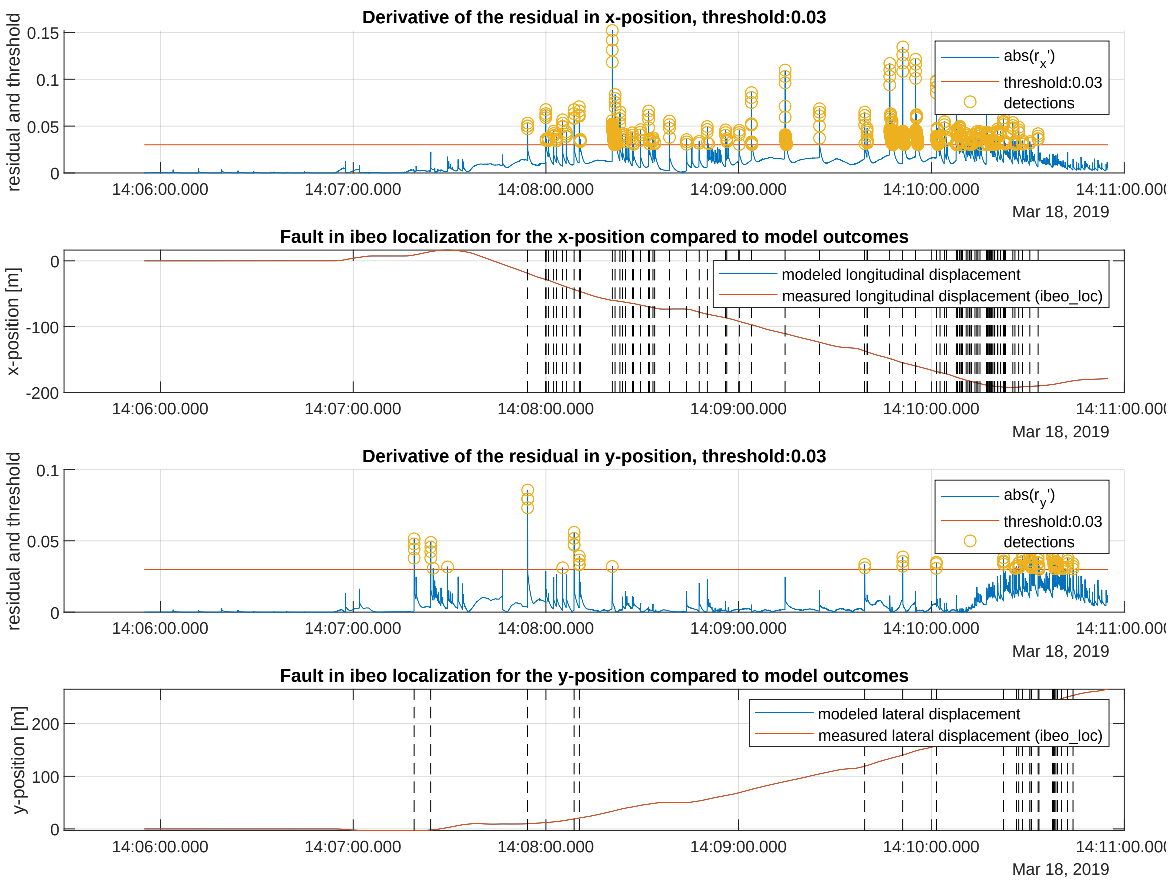


Figure 4.12: Detected faults with threshold 0.03 in IBEO localization March 18, 2019.

4.2.2. Mahalanobis distance residual combined with Chebyshev threshold

The same test data has been inspected by the second approach, using the Mahalanobis distance as a residual in combination with the Chebyshev threshold as well. Rather than only showing the difference in x- and y-position, now, the Mahalanobis distance residual is visualized, along with the threshold. The height of the threshold is determined by the Chebyshev inequality for a probability of 99.9%. Due to the scaling and computation, the threshold itself has a value of 0.029851. The detections that are performed are visualized on the residual as well. For the first testing day, March 05, 2019, see Figure 4.13. The overview of the fault detection is provided in Table 4.4.

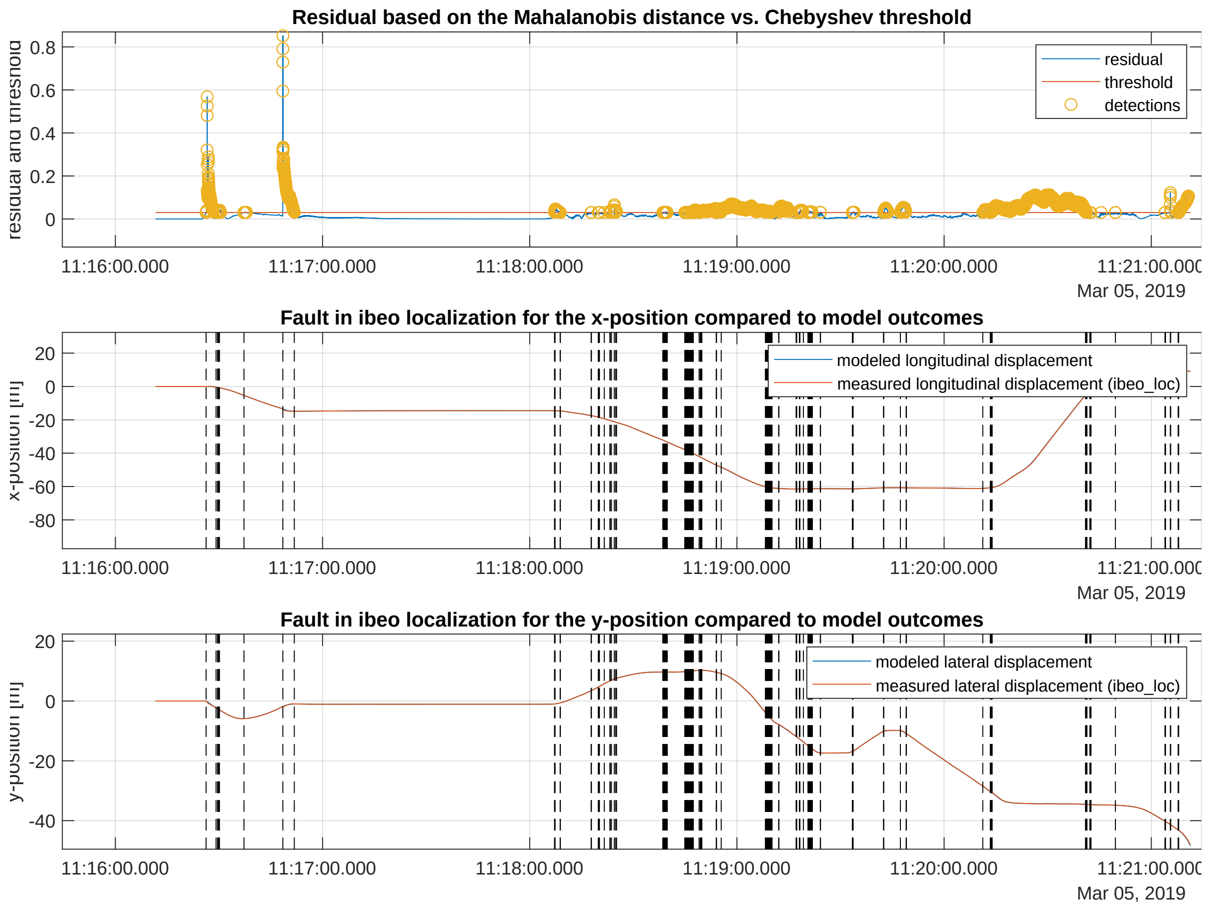


Figure 4.13: Detected fault in IBEO localization with Mahalanobis distance as residual March 05, 2019.

Table 4.4: Fault detection ID Mahalanobis distance residual March 05, 2019

Date	**March 05, 2019**	**March 05, 2019**
Threshold value	0.03	0.029851
# faults detected in x-position	170	7403, of which unique: 170
# faults detected in y-position	170	7403, of which unique: 170

On March 06, 2019, the results shown in Figure 4.14 and in Table 4.5 are achieved. The amount of unique detections for some former manually inserted threshold values are included in this table as well. The lower threshold results in a lower amount of detections, which does not seem correct at first. Actually, this is due to the way the unique faults are computed. If two detections are following up on each other, the first will be indicated as a unique fault and the second as a detection that belongs to that specific fault instance. If this occurs for multiple detections in a row, they will be gathered as one unique fault, although there might be more. This is why in the latter column (and in the other tables throughout this results section) both the 'raw' amount of detections is provided and the amount of unique faults.

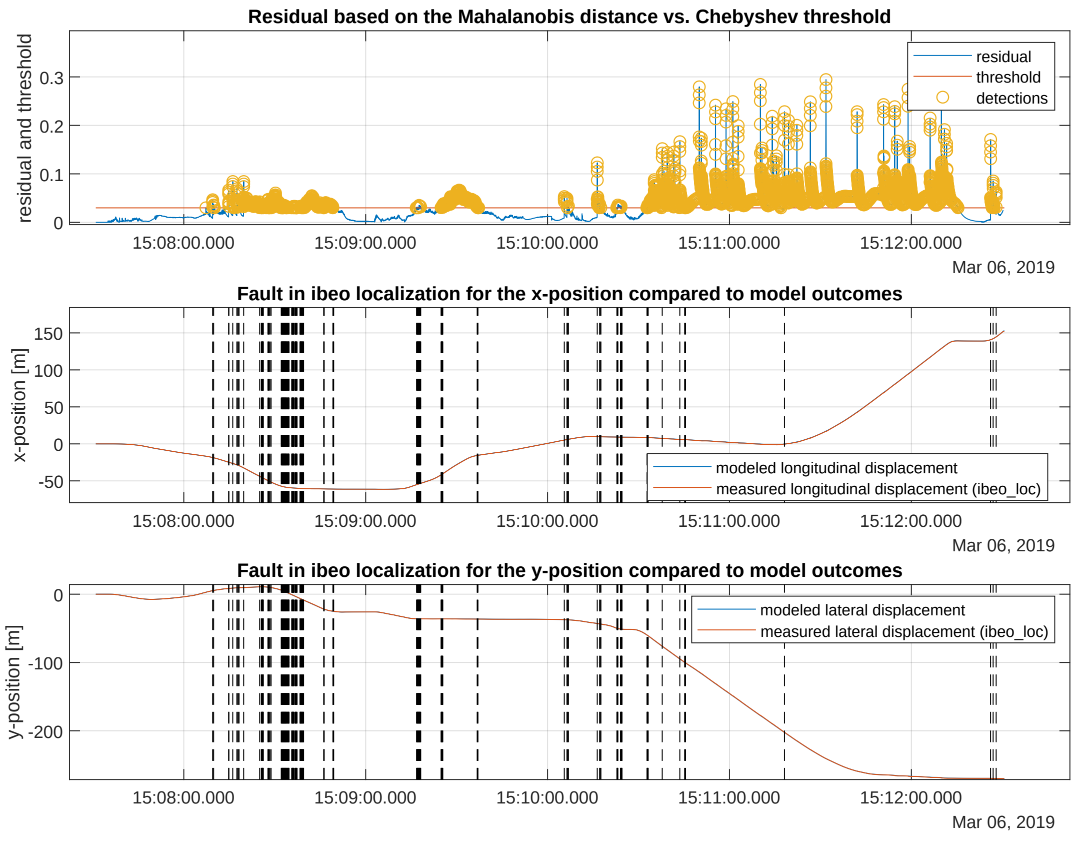


Figure 4.14: Detected fault in IBEO localization with Mahalanobis distance as residual March 06, 2019.

Table 4.5: Fault detection ID Mahalanobis distance residual March 06, 2019

Date	**March 06, 2019**	**March 06, 2019**	**March 06, 2019**
Threshold value	0.02	0.03	0.029851
# faults detected in x-position	134	161	14122, of which unique: 160
# faults detected in y-position	134	161	14122, of which unique: 160

A zoomed-in plot is provided for the 18th of March in Figure 4.15 and the details are given in Table 4.6. By enlarging the plots, the jumps are again better visible. Due to axes size, the x-position can be inspected better. Not much can be interpreted from the top plot that indicates the residual based on the Mahalanobis distance, the threshold value and the detections. All values of the Mahalanobis distance that lie above the threshold, have gotten a yellow dot. More insightful is to inspect again the residual by means of the derivative of the difference and plot the detections on there.

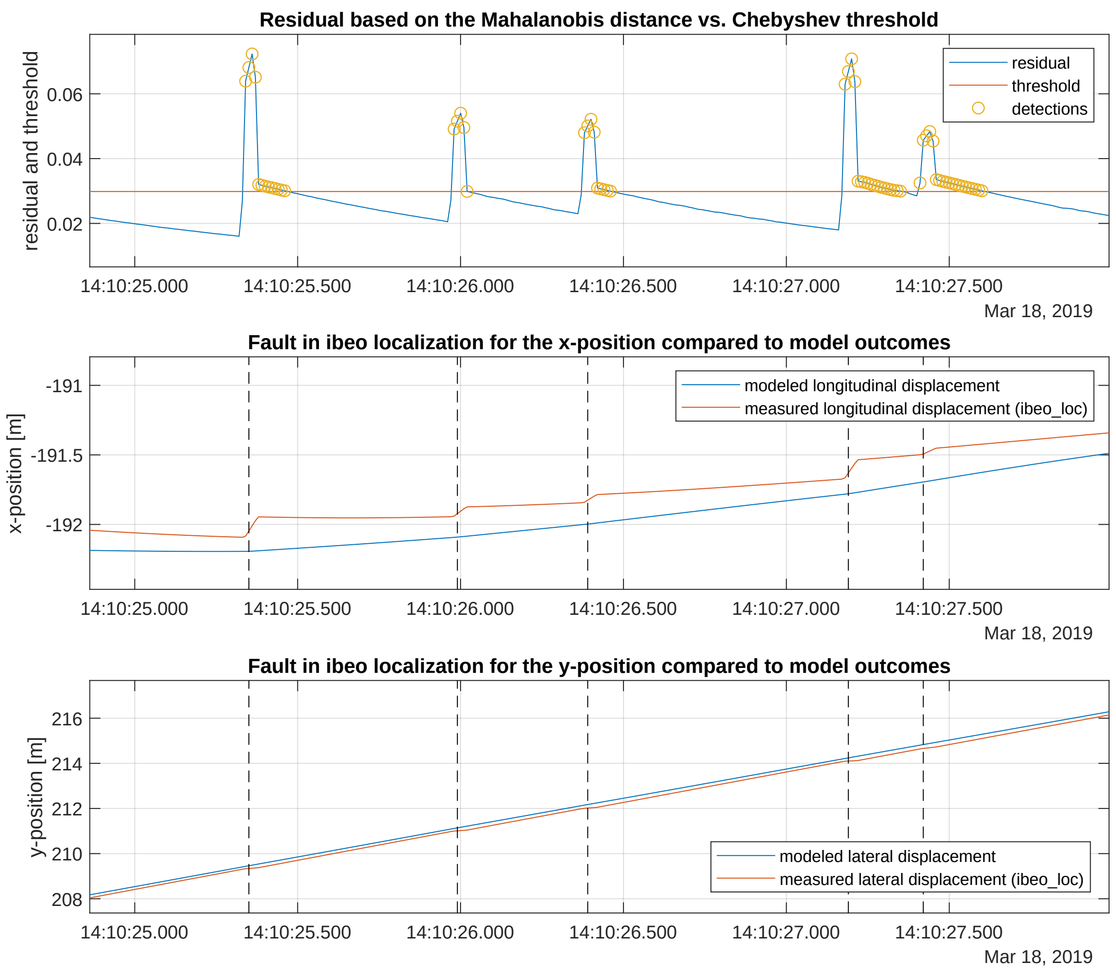


Figure 4.15: Detected fault in IBEO localization with Mahalanobis distance as residual March 18, 2019.

Table 4.6: Fault detection ID Mahalanobis distance residual March 18, 2019

Date	**March 18, 2019**
Threshold value	0.029851
# faults detected in x-position	1871, unique: 139
# faults detected in y-position	1871, unique: 139

4.2.3. Baseline

First, in Figure 4.16, the parity approximation method is tested for the baseline log (March 6th, 2019). It is visible that the absolute value of the residual stays below the threshold of 0.5 (i.e. faults of 8 centimetres or bigger, discussed in section 4.2.4) for the complete log. This means that the parity approximation algorithm performs as expected for a fault-free log. In Figures 4.17 and 4.18, the model-based approaches are shown. The Mahalanobis distance performs as expected, but the derivative of the residual is still subject to some false positives. Because the noise is high at the instances where the peak occurs, the residual is hold back slightly by the approach in which the Mahalanobis distance is taken.

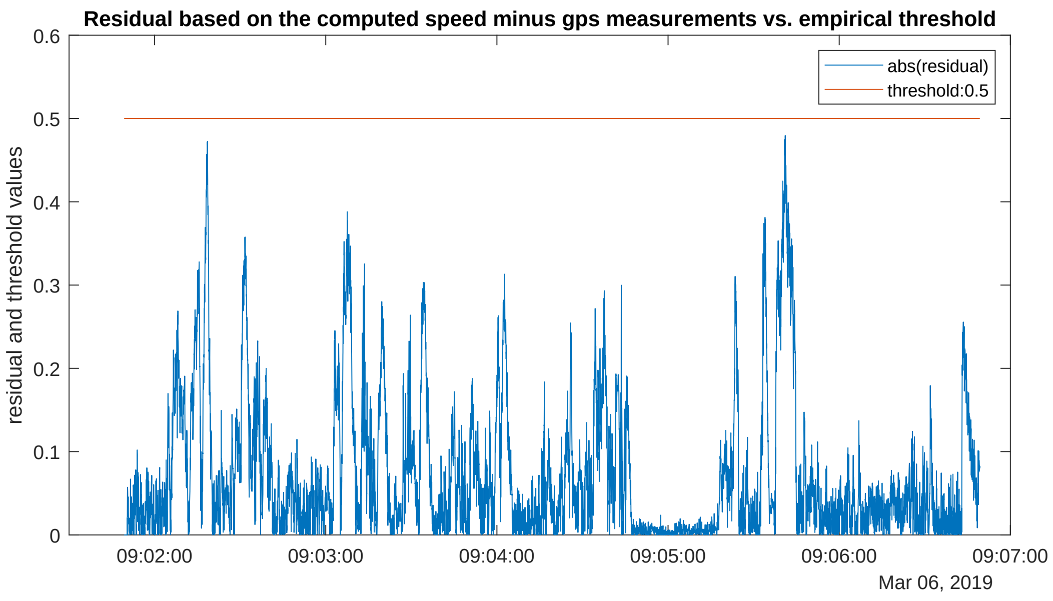


Figure 4.16: No detected faults for fault-free case using parity approximation March 06, 2019.

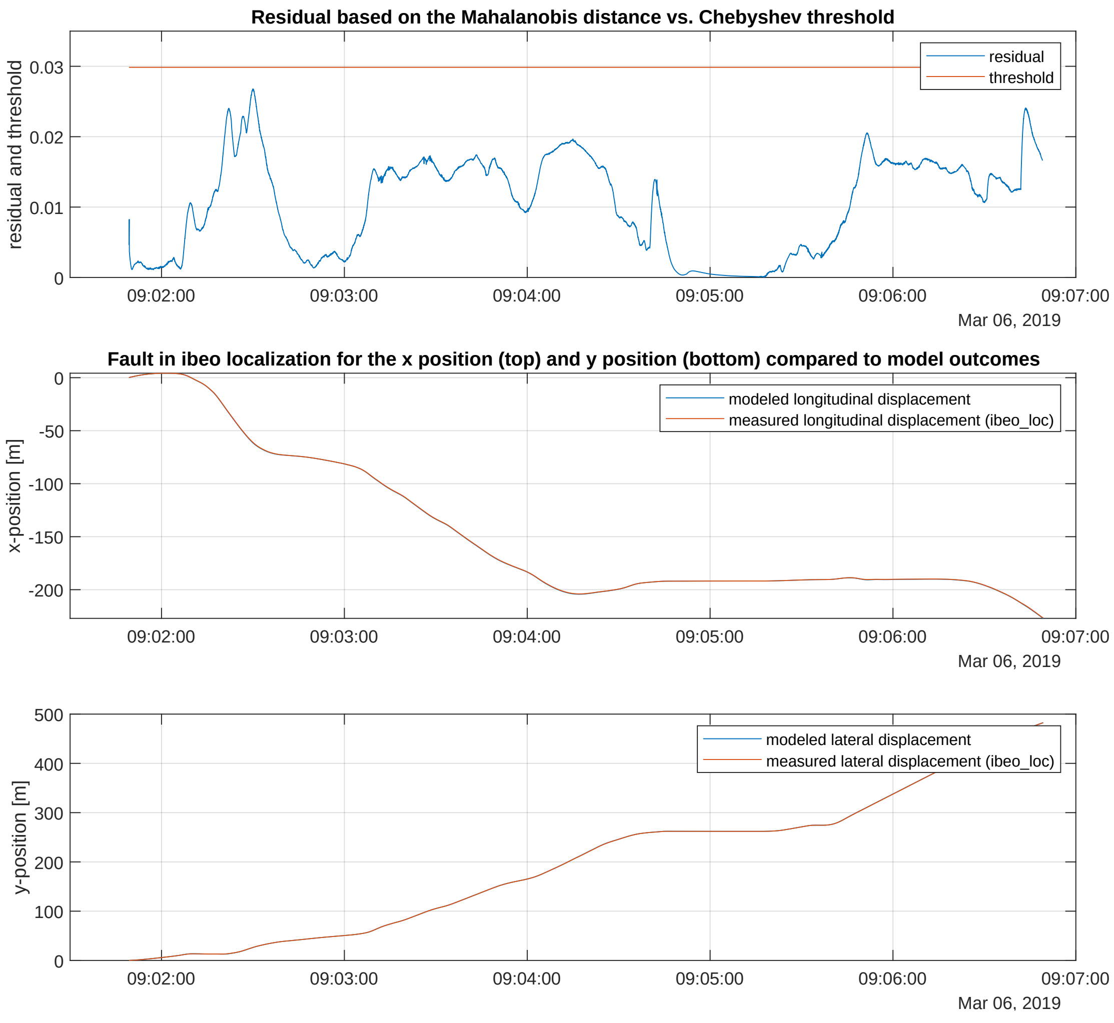


Figure 4.17: No detected faults for fault-free case using Mahalanobis distance March 06, 2019.

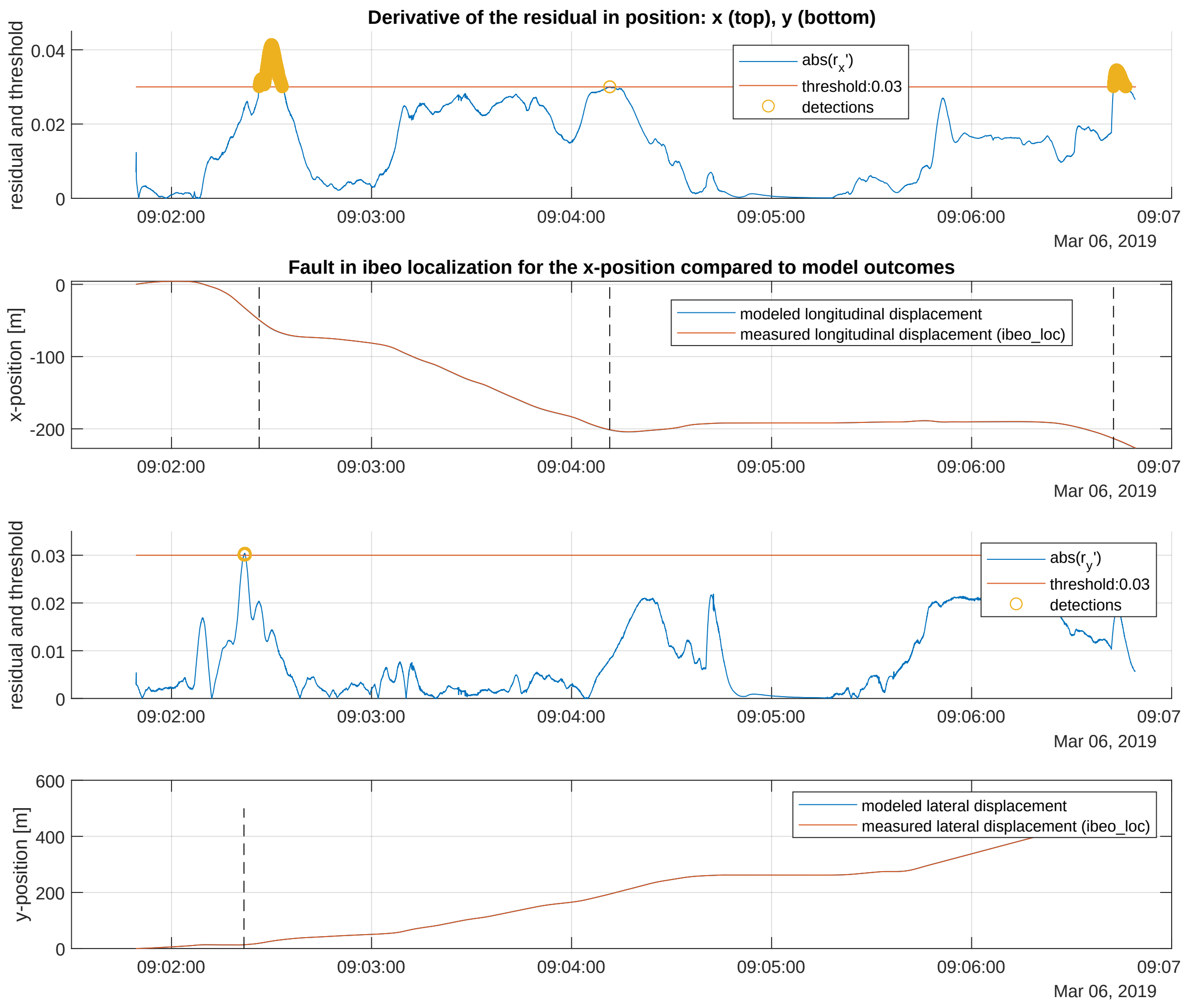


Figure 4.18: Some detected faults for the intendedly fault-free case using derivative of residual March 06, 2019.

In the discussion, section 5.1, the detections made by the derivative of the residuals made, will be briefly addressed.

4.2.4. Performance

For the parity approximation that will be used to assess several measures of performance of the remaining two methods, also an empirical threshold can be defined. A visualization of the detection using the parity approximation is shown in Figure 4.19 for a threshold of 0.05. It is visible that using this setting, detections are made already for faults smaller than 0.02 [m] in the longitudinal (x) direction and for the lateral (y) direction, even faults of half a centimeter can be identified. It is good that this method is capable of detecting faults of this size and not get deceived by other forms of noise. This is visible in the upper plot, because in the sections between the detections, the residual is more or less zero. Nevertheless, in order to be able to compare the other two methods well, a threshold should be found that lies closer to the 9 centimetres defined as a minimum offset requirement for a detection using the derivative of the difference as a residual. In section 4.1.4, an appropriate threshold is defined, with value 0.5. In Figures 4.20-4.22, the detections made for each of the used log days are shown. After that, three Tables (4.7-4.9) are provided that contain the number of faults detected for both the thresholds of a value of 0.05 and the other of 0.5.

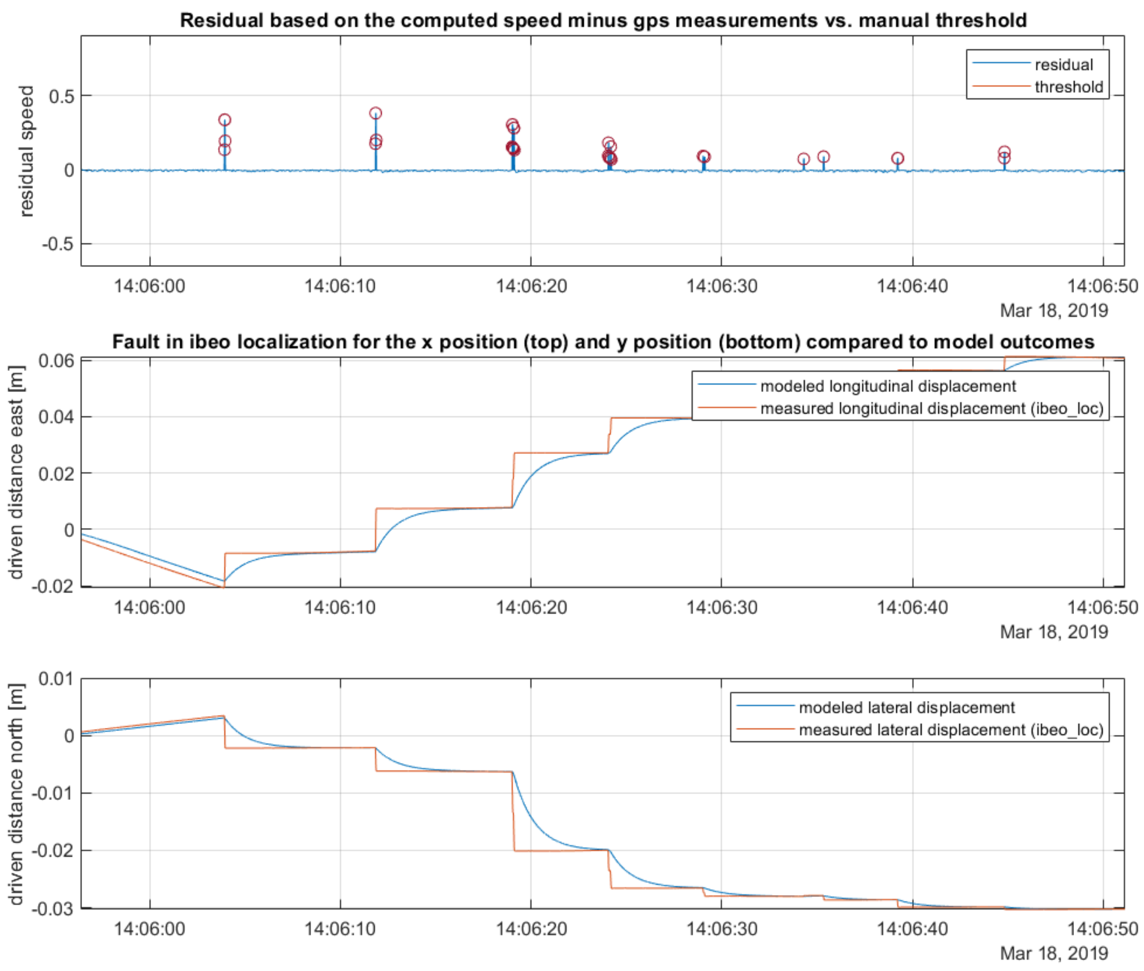


Figure 4.19: Detections based on parity approximation approach with a threshold value of 0.05. March 18, 2019.

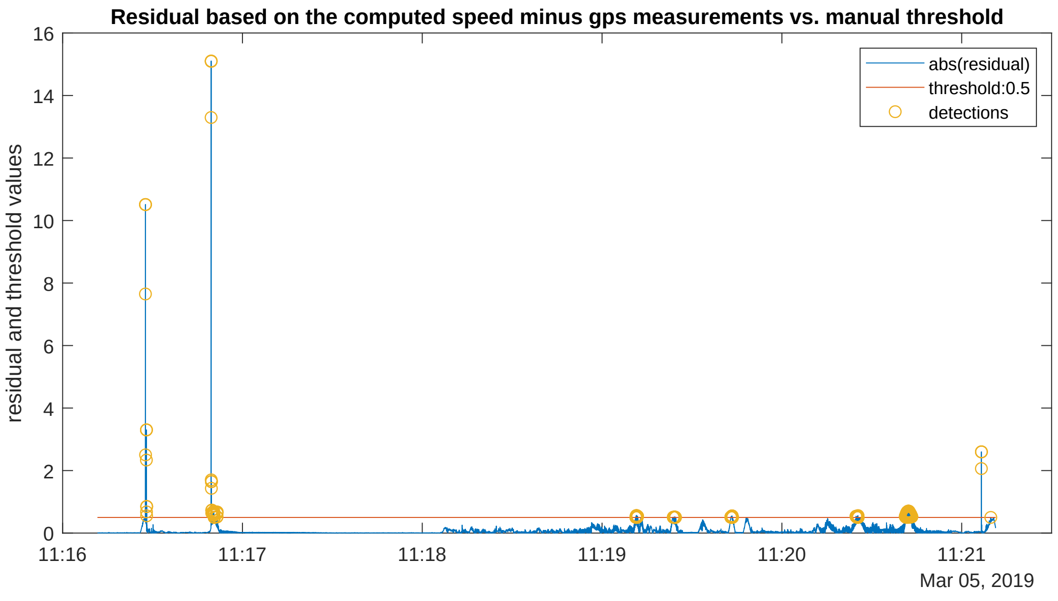


Figure 4.20: Absolute value of the residual based on parity approximation with a threshold of 0.5. March 05, 2019.

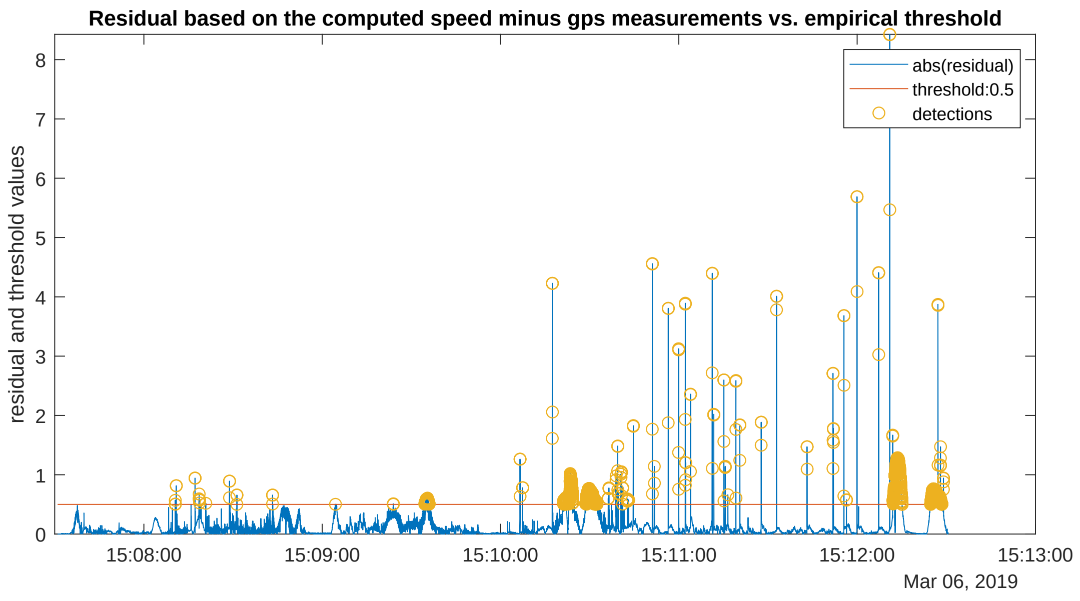


Figure 4.21: Absolute value of the residual based on parity approximation with a threshold of 0.5. March 06, 2019.

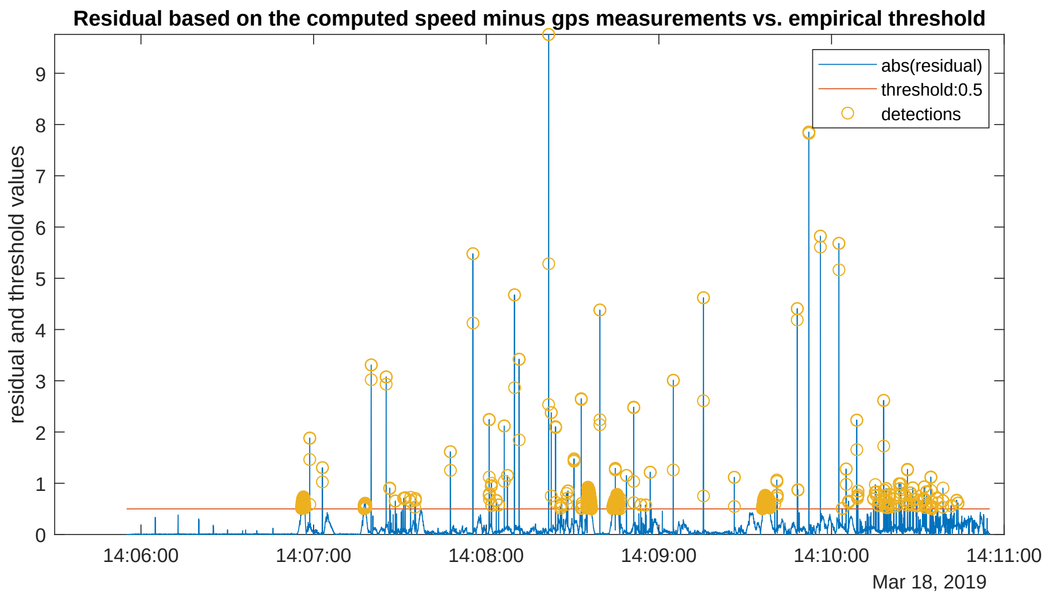


Figure 4.22: Absolute value of the residual based on parity approximation with a threshold of 0.5. March 18, 2019.

Table 4.7: Fault detection ID parity approximation residual March 05, 2019

Date	**March 05, 2019**	**March 05, 2019**
Threshold value	0.05	0.5
# faults detected	10520, of which unique: 1083	287, of which unique: 64

Table 4.8: Fault detection ID parity approximation residual March 06, 2019

Date	**March 06, 2019**	**March 06, 2019**
Threshold value	0.05	0.5
# faults detected	16331, of which unique: 823	1433, of which unique: 93

Table 4.9: Fault detection ID parity approximation residual March 18, 2019

Date	**March 18, 2019**	**March 18, 2019**
Threshold value	0.05	0.5
# faults detected	15922, of wich unique: 359	1138, of which unique: 101

Hit rate

The amount of faults with a jump of 2 centimetres or higher, that has been successfully detected by the model-based approach on the derivative of the residual only and a threshold of 0.03, is therefore for the x-position on March 18, 2019 : 28/359*100%=7.8%. When the Mahalanobis distance is used, however, with a threshold of 0.02985 is 139/359*100%=38.7%. When a threshold of 0.5 is chosen, 101 faults are detected by the parity approximation approach. This means that around 25% is detected with only the derivation of the threshold, and even more than 100% with the Mahalanobis distance. So, it can be said that all faults with an offset of 8cm and higher, can be detected by the Mahalanobis distance.

Using this line of thought, the table provided below is set up. The thresholds of 0.03, 0.02985 and 0.5 are taken respectively for the derivative of the difference (Der_of_diff), the Mahalanobis distance (Mah_dist) and the parity approximation approach in order to compare. In Table 4.10, the three log dates are visualized at the top, with the number of detections for each of the three approaches underneath. Then, the last two columns are dedicated to the hit rate percentages based on comparison with the parity approximation approach for both the raw detections and the unique ones (o.w.u.).

Table 4.10: Fault detection and hit rate comparison of both the model-based approaches and the parity approximation method

Date	**March 05, 2019**	**March 06, 2019**	**March 18, 2019**
Der_of_diff: # F in x	108, o.w.u.: 1	715, o.w.u.: 39	776, o.w.u.: 82
Der_of_diff: # F in y	24, o.w.u.: 5	709, o.w.u.: 37	96, o.w.u.: 25
Mah_dist: # F in x	7403, o.w.u.: 170	14122, o.w.u.: 160	1871, o.w.u.: 139
Mah_dist: # F in y	7403, o.w.u.: 170	14122, o.w.u.: 160	1871, o.w.u.: 139
Parity approximation: # F	287, o.w.u.: 64	1433, o.w.u.: 93	1138, o.w.u.: 101
Hit rate derivative x	37.6%, o.w.u.: 1.6%	49.9%, o.w.u.: 41.9%	68.2%, o.w.u.: 81.2 %
Hit rate derivative y	8.4%, o.w.u.: 7.8%	49.5%, o.w.u.: 39.8 %	8.4%, o.w.u.: 24.8%
Hit rate Mahalanobis x	2579.4%, o.w.u.: 265.6%	985.5%, o.w.u.: 172.0%	164.4%, o.w.u.:137.6%
Hit rate Mahalanobis y	2579.4%, o.w.u.: 265.6%	985.5%, o.w.u.: 172.0%	164.4%, o.w.u.:137.6%

A visual comparison of the number of detections of the three methods is given in Figure 4.23. An assessment of this figure is provided in the discussion, section 5.1.

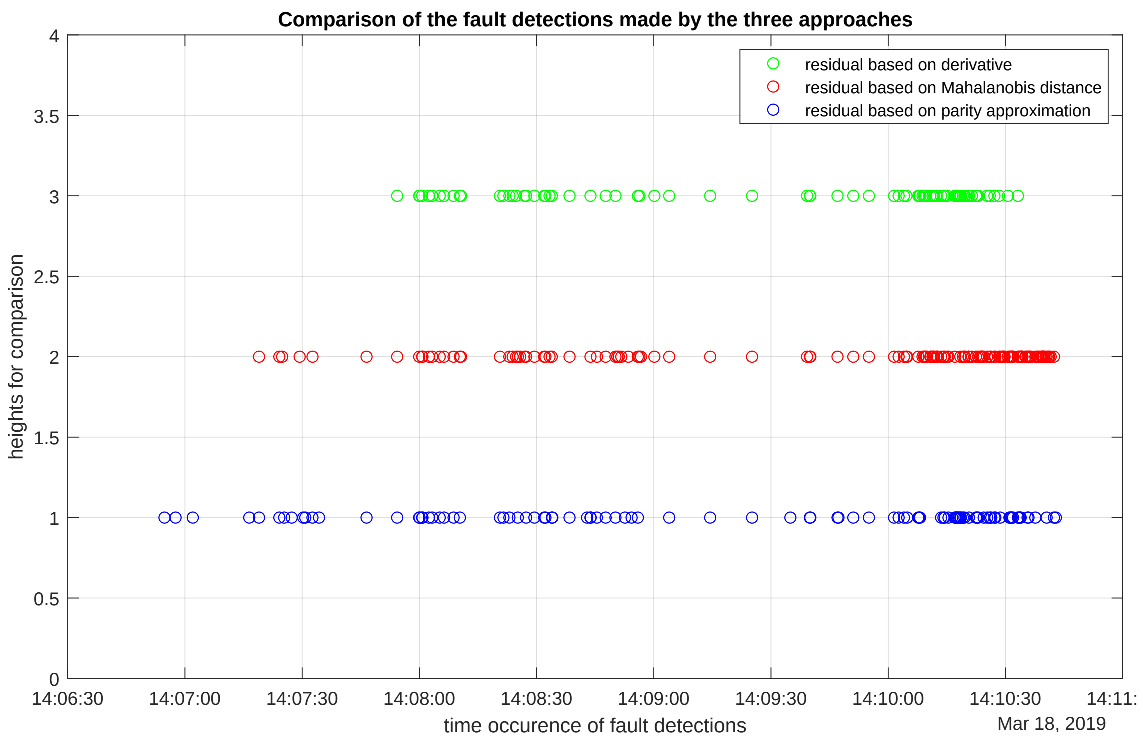


Figure 4.23: Detections made by the different methods compares, March 18, 2019.

Computational time

The following computational times have been achieved at an initial run of the *run_Simulink.m* file:

- Read in, convert and save data from .rosbag to .mat files: 22.31 seconds.
- Read in the data from .mat files: 4.16 seconds.
- Prepare the data as input for simulation: 0.28 seconds.
- Simulate: 1.56 seconds.
- Derivative of residual: 0.98 seconds.
- Mahalanobis distance: 26.95 seconds.

- Parity approximation: 0.09 seconds.

The total computational time that is required equals therefore 56.34 seconds, slightly under one minute. Note that reading in the data from the .rosbag and the .mat files depend on a GUI, so the speed with which the files are selected, adds up to the actual computational time. The calculation of the Mahalanobis distance takes relatively the most time.

Table 4.11: Computational times of the methods

Method	**Derivative of residual**	**Mahalanobis distance**	**Parity approximation**
Comp. time inc. saving data	29.30 [s]	56.25 [s]	28.41 [s]
Comp. time exc. saving data	6.99 [s]	33.94 [s]	6.10 [s]

5 Discussion and Conclusion

This chapter concludes the research based on the findings presented in the preceding chapters. A discussion is provided first, in which a critical view on the methods used and retrieved results is taken. Then, the actual conclusion is provided. At the end, some future recommendations are included of how to implement the methods found in an on-line way and an approach of how to extend the research towards actual fault tolerant control is suggested.

5.1. Discussion

In this part, a critical evaluation is made of the methods and results. Also, some of the results are elaborated on more in-depth in comparison to the results section (i.e. what has been observed and can be extracted from this). The discussion will be provided in a point by point form, following the chronological order of appearance in the report.

- The third method in residual determination that has been used for validation, is the parity approximation. This approach computes the speeds based on the localization measurements for the position in x- and y-direction and calculates the offset between this outcome and the speed measurements of the GPS sensor.

$$r = \left(\sqrt{\Delta x_{IBEO}^2 + \Delta y_{IBEO}^2}\Delta t\right) - v_{GPS,shifted}. \tag{5.1}$$

  The hit rate of this approach is assumed to be 100%. This is something that cannot be assumed for sure at any random given point in time. Also, there is a delay in the GPS measurements in comparison to the localization output, of around one second. It can be used as validation method for off-line inspection (or on-line while driving, taking a delay of 1 second into account).

- The parity approximation approach itself can function as an isolation method, because a GPS offset (which is very likely to occur due to environmental influences like weather, leaves from trees and tall buildings in narrow streets) will be reflected in the detectability of this parity checking approach.

- It turns out that the model is not a 1-to-1 representation of reality, therefore it would be advisable to update the model. One way of doing this is using another value for the model uncertainty η. This is now set to 0. More realistic would be to retrieve and implement some actual model noise values.

- At this point, the way the covariance matrix is computed for the Mahalanobis distance residual, is not strictly on-line compatible. A way to avoid this, is by determining fixed covariance matrices based on previous fault-free logs and use this database for on-line computations. Along with all the rosbag logs, the circumstances of the test day are provided as well. Several covariance matrices can be identified based on this information for different weather and/or surrounding density conditions. Once the vehicle recognises certain circumstances (this functionality needs to be implemented in addition in that case), it can distinguish the appropriate covariance matrix and compute the Mahalanobis distance based on that uncertainty.

- Robustness of threshold computation can be improved. Using an empirical threshold as is done in this model-based approach, is not the most robust way of determining an appropriate bound and the minimum size of the detected fault since it requires visual inspection.

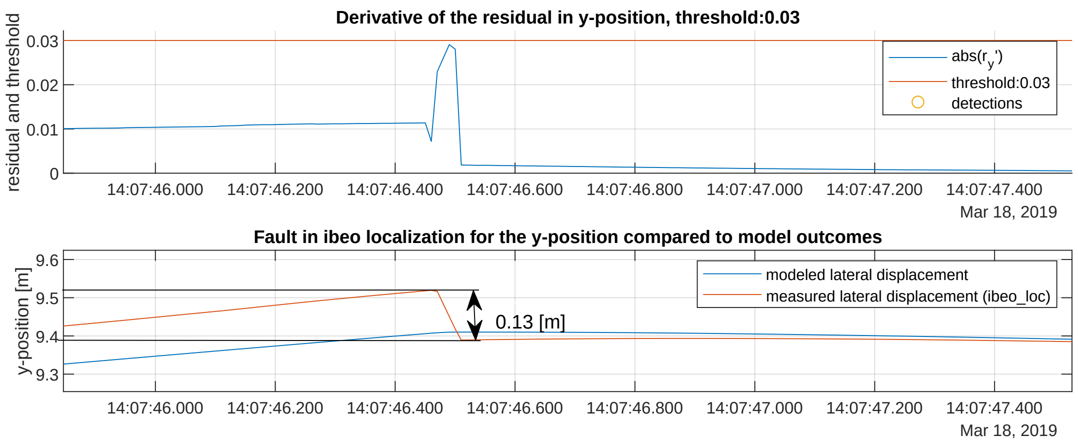


Figure 5.1: Empirical threshold determination for residual as derivative of the offset.

Therefore, it is better to use a dynamic or adaptive threshold that can take these edge-cases into account and a more robust threshold can be defined.

- When validating the choice of threshold, it turned out that this method does not hold for specific cases. Namely, in Figure 5.1, it is visible that even though the IBEO localization output jumps for over 10 centimetres, the threshold of 0.03, which is supposed to be able to detect faults bigger than 9 centimetres, is not exceeded. This is due to the fact that in this specific case, the jump of the IBEO localization 'crosses' the x-position output of the model. The sign flips and therefore the derivative of the difference between the measurements and the model is not able to reach the threshold.

- The log (6th of March, early morning) that has been chosen as a baseline, seems not to be completely fault free. Detections are even made by the derivative of the difference approach. This is due to noise in the log, rather than actual malperformance of the method used. The noise data (of the covariances) has been inspected with respect to the difference between the model output and the measurements in both the longitudinal (x-) and lateral (y-) direction. Indeed, the covariance noise is smaller at the instances where the vehicle model is deviating less from the actual measurements. By inspection of Figure 4.18, it becomes clear that the vehicle remained also in standstill position at that moment, so smaller covariances are already assumed because of that. This means that there are other external noises that influence the log of the 6th of March. This could be due to one of the factors described in section 3.3. Assessing these influences better and taking them into account while drawing up the residuals and/or threshold, will lead to more accurate results, that will perform less false positives.

- Another apparent influencing factor is the unexpected behavior of the model when a change in direction of the driven path takes place. This becomes clear at March 5th, halfway the morning, where two large peaks are visible at the beginning of the day (Figure 4.13). The rest of the log only contains small peaks that quickly follow up on each other. What is interesting, is that the large peaks represent changes in the driven direction. So the fault is detected correctly, only it addresses another type of fault: the unexpected behavior due to smoothing out of the control (as described in section 3.4.4). It is key to come up with an approach that is able to distinct between IBEO localization jumps and other type of faults (this is where isolation comes into play).

- Regarding the results section, multiple findings are retrieved that are open for discussion. First of all, the use of the 'unique fault'. When two detections follow up on each other, the first will be indicated as a unique fault and the second as a detection that belongs to that specific fault instance. If this occurs for multiple detections in a row, they will be gathered as one unique fault, although there might have occurred more faults during that specific time window. This is why in case of using a lower threshold (i.e. more detections) the number of unique detections might lie lower than for a higher threshold, which obviously incorrect: the lower the threshold, the higher the number of detections for the same data set. It is important to group the instances that belong to the same fault together, while keeping the fault occurrences 'unique'. A way to do this, is for example by comparing the values of the successive detections and select only the relatively big residuals (i.e. local maxima) to represent unique faults.

- The approach of taking the derivative of the difference as a residual, is capable of detecting faults in x- and y-direction separately, because two residuals are drawn up. The Mahalanobis distance only uses one residual that incorporates the noise in x- and y-direction within the covariance matrix directly. Therefore, the same residual is used for both the longitudinal and lateral direction, because it contains the relevant information for both of them. However, it is beneficial to be able to detect separately because it is not always the case that when a fault occurs, it has influence on both the longitudinal and lateral direction. The jumps due to the IBEO localization turned out to show this behavior: influencing both the x- and y-direction, however sometimes the jump is larger for one of the two directions. Therefore, using the derivative of the difference as a residual, can provide more information about the effects of the faults. However, in essence it is important that a fault is detected in the first place, and less what the effects thereof are. By extending the research to an actual fault tolerant approach, this becomes of higher interest, see also the recommendations about performance degradation in section 5.2.1.

- When comparing the number of detected faults by the three approaches taken, it becomes clear that the Mahalanobis distance residual detects significantly more faults than the other approaches. This is due to the fact that by taking this approach, the detectability is improved. However, using the Chebyshev inequality, a threshold almost equal to the one used for the derivative of the difference approach is computed. Therefore, more faults are detected. It is advisable to use a lower threshold, i.e. choose a smaller α in the Chebyshev inequality, to come up with a more realistic detection profile.

- It makes sense that the parity approximation approach also performs more detections than the first approach, because the threshold is set differently. It is chosen as 0.5, which leads to detections of 8 centimetres and more, while the other two methods can detect faults of 9 centimetres and larger only. A way of retrieving thresholds that achieve the same detection accuracy (i.e. minimum amount of centimetres before detections take place), needs to be implemented for valid comparison.

- The hit rate now is determined based on a comparison of percentages only. What is more important, however, is the detectability of the used approach. False positives (FP) can lead to unnecessary delays in the system, since they represent unjustified detections of faults. Especially when the future implementations (again, for example: performance degradation) are regarded, a safe-stop could be initiated, although nothing was wrong. Even more important to prevent are false negatives (FN), in which the vehicle assumes its localization module to be fault-free, while there is actually a deviation from reality present. This could lead to dangerous situations or a mismatch between the desired end location and the actual end of the driven track. Therefore it is important to select the method of sufficiently high detectability. Also, an optimal balance between true positives (TP) and false positives has to be found. Having the information as shown in Figure 4.23 available, it is possible to determine the false positives and false negatives. Assume the blue detections (bottom row in the plot), that represent the parity approximation approach, to be 100% correct. This way, the exact detection instances in seconds can be compared. The number of congruent indices has to be determined. This indicates the number of TPs in the approach. This amount can then be subtracted from the total amount of detections for the method used. Now, the number of FPs is obtained. By subtracting the number of TPs for the used method from the total amount of detections made by the reference method (in this case the parity approximation approach), the number of detections that has been missed, i.e. the amount of FNs, is obtained.

- Finally, the computational time can be reduced by first optimizing the code by amongst others, avoiding for-loops (by vectorizing) and preallocating data instead of using arrays of variable sizes. Also, more functions can be defined that will make the master m-file faster. Completely using another programming environment than MATLAB is another way of reducing the computational time, because MATLAB tends to take in general more computational effort than other programmes. Also, using another environment could lead to less dependency on expensive external software (and its updates), especially when the choice is made for one of the open source available software environments.

  The piece of code that is responsible for most of the elapsed time is the one where the Mahalanobis distance is calculated. Making use of the Mahalanobis distance is a trade-off between detectability and computational time. On the one hand, the detectability is improved when the Mahalanobis distance is used, but on the other hand, the computational speed is lower. When the more basic residuals already perform within the desired bounds of detectability, the choice could be made to continue only with these residuals, rather than incorporating the Mahalanobis distance as well. Especially when the desire of implementing the model-based fault diagnosis in an on-line way is kept in mind.

5.2. Conclusion

The model-based fault diagnosis approach that is used, consists of a model based on the kinematic and dynamic equations of the vehicles from the application at hand (WEpods). This model is enhanced with an observer that takes the actual measurements into account and prevents the model from an excessive drift between the system and the model. Innovative here is that the model is split into two observers, so undesired influence in the localization output (the x-position, y-position and orientation of the vehicle) of the model resulting from other faulty sensors or noise in the measurements of the speed and steering angle is prevented.

The difference between the model and the measurements based on the same input u, consisting of the desired speed and steering angle, functions as a preliminary residual of the approach:

$$r = y_{measurements} - y_{model}. \tag{5.2}$$

The two developed fault detection methods are listed here:

- *Derivative of the residual*
  The first approach is based on the derivative of this residual computed as such:

  $$r' = r_{+} - \lambda r. \tag{5.3}$$

  This approach is able to achieve a hit rate of (averaged over the two directions and three log days): 36%, hit rate of unique faults: 33% within a computational time of: 6.99 seconds. The hit rate for the longitudinal (x-) direction, however, lies a bit higher ~ 40% compared to the hit rate of ~ 20% for the lateral (y-) direction.

  Advantages of this method:
  - Ability to detect faults in longitudinal and lateral directions separately;
  - On-line implementation possible.
  Drawback of this method:
  - Less robust, prone to false positives

- *Mahalanobis distance*
  The second approach is based on the previous residual with an update. It includes the calculation of the Mahalanobis distance:

  $$d(z) = \sqrt{(z-\mu)^T C^{-1} (z-\mu)}. \tag{5.4}$$

  Implemented by (with R: rotation matrix, z: vector containing r'_x and r'_y, C: covariance matrix containing noise of the localization measurements):

  $$r'' = \sqrt{(Rz)' 2C^{-1} Rz}. \tag{5.5}$$

  This approach is able to achieve a hit rate of: 1243%, hit rate of unique faults: 189% within a computational time of: 33.94 seconds.

  Advantages of this method:
  - Relatively high detectability;
  - On-line implementation possible.
  Drawback of this method:
  - Higher computational time.

In Table 5.1, a summary is provided of the average performance of the different approaches.

Method	*Derivative of residual*	*Mahalanobis distance*	*Parity approximation*
Hit rate unique faults	33%	189%	100% (if fault-free GPS)
Computational time	6.99 [s]	33.94 [s]	6.10 [s]

Table 5.1: Performance of the different methods, left two: model-based, right: validation by parity approximation.

The parity approximation seems most beneficial in terms of hit rate and computational time. However, the assumption of a hit rate of 100% is only valid when the GPS output is completely fault-free. Using the Mahalanobis distance ensures a higher hit rate (in this case larger than 100%, because the amount of detected faults is compared to the number of detections in the parity approximation method for a slightly different accuracy as explained in the discussion, point 12, section 5.1), but comes at a higher computational cost. The approach in which the derivative of the difference is used, has the ability to detect faults separately for the longitudinal and lateral direction.

Using either one of the model-based fault detection approaches described above, the ability is provided of identifying the occurrence of a fault. By providing the ability to diagnose faults from within the automated vehicle itself, the measure of autonomy is extended. Instead of remaining in need of a steward that presses the emergency stop button when it inspects odd behavior (which indicates a system failure), the system can provide information about which sensor is wrong and detail this information with messages about time of fault occurrences. This will allow the vehicle to adjust its autonomous driving in such a way that it can handle these faults internally, and they do not result into system failures. Therefore, the research goal: *let the WEpods continue driving in autonomous mode more often than is currently the case* is met. Next to that does this research form the relevant scientific contribution of successfully incorporating a model-based fault diagnosis strategy into the localization module (also: self-localization) of an autonomous driving application.

5.2.1. Recommendations

In the future, the system can be extended with the functionality of affecting the path planner controller. By adjusting the control on-line based on faults present in the system, active fault tolerant control is reached. This could be done by putting less trust on a malfunctioning sensor. When this is done, the vehicle can either continue its original driving behavior, or, if necessary, be subjected to a new behavior objective within the safety limits. A performance degradation assessment can be made in order to find out which range of operation is still safe under the assumption that one (or multiple) of the sensors is (or are) not working correctly.

The methodology described in this research, has been applied in an off-line way: based on a pre-computed model and using localization data from logs. It is very well possible to extend it to an on-line implementation, by running the model simultaneously with the actual driving. The model computes the expected output and compares this with the localization data immediately. It is important to take computational time into account. This is in an ideal case no less than the speeds at which the measurement samples of the vehicle are acquired. Also, some of the values, that have been replicated or interpolated in order to match the highest frequency in the loop, should be kept on hold or interpolated over the according time steps. To be able to successfully run the model-based fault detection approach on the vehicle, only a conversion to runnable files on the WEpod system has to be done.

Bibliography

[1] Spatial dual reference manual. `https://www.advancednavigation.com/product/spatial-dual`, 2017.

[2] Universal transverse mercator coordinate system. `https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system`, 2019.

[3] Unix time. `https://en.wikipedia.org/wiki/Unix_time`, 2019.

[4] World geodetic system. `https://en.wikipedia.org/wiki/World_Geodetic_System`, 2019.

[5] Cartesian coordinate system. `https://en.wikipedia.org/wiki/Cartesian_coordinate_system`, 2019.

[6] Definiteness of a matrix. `https://en.wikipedia.org/wiki/Definiteness_of_a_matrix#Examples`, 2019.

[7] Karl Berntorp. Joint wheel-slip and vehicle-motion estimation based on inertial, gps, and wheel-speed sensors. *IEEE Transactions on Control Systems Technology*, 24(3):1020–1027, 2016.

[8] Karl Berntorp and Stefano Di Cairano. Tire-stiffness and vehicle-state estimation based on noise-adaptive particle filtering. *IEEE Transactions on Control Systems Technology*, (99):1–15, 2018.

[9] Francesca Boem, Riccardo MG Ferrari, Christodoulos Keliris, Thomas Parisini, and Marios M Polycarpou. A distributed networked approach for fault detection of large-scale systems. *IEEE Transactions on Automatic Control*, 62(1):18–33, 2017.

[10] Peter de Bakker. Localization system design description. *TU Delft*, 2016.

[11] Riccardo M G Ferrari. *Lecture 5: Change Detection Algorithms*. Lecture slides for the course Fault Diagnosis and Fault Tolerant Control - Delft University of Technology, 2018.

[12] Riccardo M G Ferrari. *Lecture 7: Model Based Fault Diagnosis: Detection*. Lecture slides for the course Fault Diagnosis and Fault Tolerant Control - Delft University of Technology, 2018.

[13] Fabio A González and Dipankar Dasgupta. Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines*, 4(4):383–403, 2003.

[14] Lj Grujic and D Siljak. On stability of discrete composite systems. *IEEE Transactions on Automatic Control*, 18(5):522–524, 1973.

[15] Jose E Guivant and Eduardo Mario Nebot. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *IEEE transactions on robotics and automation*, 17(3): 242–257, 2001.

[16] Elwan Héry, Philippe Xu, and Philippe Bonnifait. Lidar based relative pose and covariance estimation for communicating vehicles exchanging a polygonal model of their shape. In *10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles*, 2018.

[17] Remco de Lange Hugo van den Brand and Dragan Kostić. Ssd wepod high level control design. *Sogeti*, 2016.

[18] Remco de Lange Hugo van den Brand and Dragan Kostić. Tsr wepod system identification august 2016. *Sogeti*, 2016.

[19] Charles Jeffrey. *An Introduction to GNSS - GPS, GLONASS, BeiDou, Galileo and other Global Navigation Satellite Systems*. NovAtel Inc., 2015.

[20] Pieter Jonker and Jan Willem van der Wiel. System architecture description & safety report. *Robot Care Systems*, 2018.

[21] R. V. Kossen. Literature survey - fault tolerant control. *Delft University of Technology*, 2018.

[22] Vijay Kumar. Effect of environmental parameters on gsm and gps. *Indian Journal of Science and technology*, 7(8):1183–1188, 2014.

[23] Peter de Bakker-Danny Suls Peter de Jager Jeroen Ploeg Jan Willem van der Wiel Tom Jansen Theo Tieman Henk van de Brink Pieter Jonker Koen Lekkerkerker Maja Rudinac, Dimitrios Kotiadis. Wepods - system architecture document. *Robot Care Systems*, 2016.

[24] Florian Netter. Künstliche intelligenz im auto—applikationen, technologien und herausforderungen. *ATZelektronik*, 12(1):20–25, 2017.

[25] Society of Automotive Engineers. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems j3016_201401. `https://www.sae.org/standards/content/j3016_201401/`, January 2014.

[26] Habib Oladepo. Ros graph concepts: Nodes. `http://wiki.ros.org/Nodes`, December 2018.

[27] Maja Rudinac and Koen Lekkerkerker. Technical safety report wepods. *Robot Care Systems*, 2016.

[28] Srishti Saha. Baffled by covariance and correlation??? get the math and the application in analytics for both the terms.. `https://towardsdatascience.com/let-us-understand-the-correlation-matrix-and-covariance-matrix-d42e6b643c22`, October 2018.

[29] Akshay Shetty and Grace Xingxin Gao. Adaptive covariance estimation of lidar-based positioning errors for uavs.

[30] Akshay Shetty and Grace Xingxin Gao. Covariance estimation for gps-lidar sensor fusion for uavs. In *Proceedings of the 30th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2017), Portland, OR, USA*, 2017.

[31] MapTiler team. Epsg:32632 - wgs 84 / utm zone 32n. `https://epsg.io/32632`, 2019.

[32] ISO 26262-1 to 9:2011(E). ISO 26262-10:2012(E). Road vehicles – functional safety. *International Organization for Standardization, Geneva, Switzerland.*

[33] I-AT Interreg Automated Transport. Zelfrijdend vervoer in de grensregio nederland-duitsland. `https://www.i-at.eu/`, November 2018.

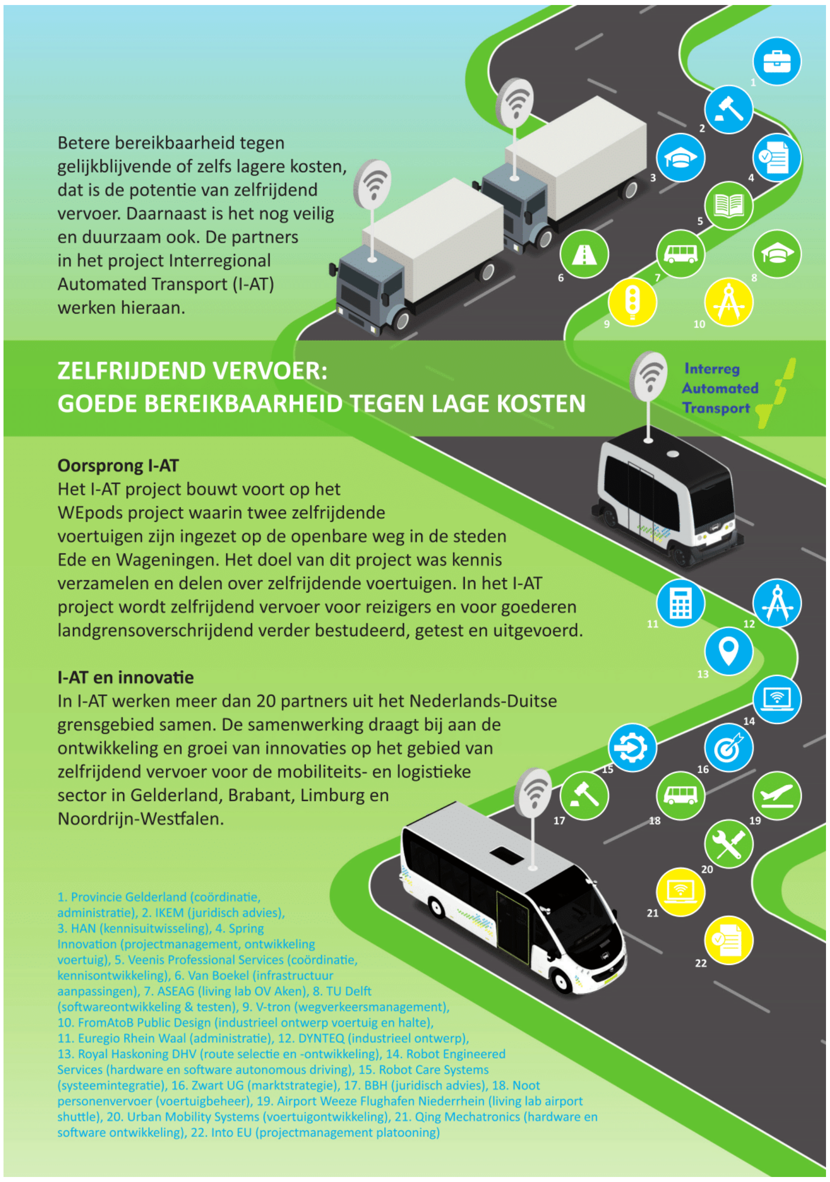
[34] Jan Verhaegh and Jeroen Ploeg. Wepods deliverable: Vehicle platform modelling, sensor fusion modelling, and control architecture design. *TNO*, 2015.

[35] Erik Vlasblom. Coordinate systems. *Robot Robots Company*, 2019.

[36] Youhanna William, Walid Oraby, and Sameh Metwally. Analysis of vehicle lateral dynamics due to variable wind gusts. *SAE International Journal of Commercial Vehicles*, 7(2014-01-2449):666–674, 2014.

[37] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, page 9, 2014.

A
Flyer Infographic I-AT



Betere bereikbaarheid tegen gelijkblijvende of zelfs lagere kosten, dat is de potentie van zelfrijdend vervoer. Daarnaast is het nog veilig en duurzaam ook. De partners in het project Interregional Automated Transport (I-AT) werken hieraan.

ZELFRIJDEND VERVOER:
GOEDE BEREIKBAARHEID TEGEN LAGE KOSTEN

Oorsprong I-AT
Het I-AT project bouwt voort op het WEpods project waarin twee zelfrijdende voertuigen zijn ingezet op de openbare weg in de steden Ede en Wageningen. Het doel van dit project was kennis verzamelen en delen over zelfrijdende voertuigen. In het I-AT project wordt zelfrijdend vervoer voor reizigers en voor goederen landgrensoverschrijdend verder bestudeerd, getest en uitgevoerd.

I-AT en innovatie
In I-AT werken meer dan 20 partners uit het Nederlands-Duitse grensgebied samen. De samenwerking draagt bij aan de ontwikkeling en groei van innovaties op het gebied van zelfrijdend vervoer voor de mobiliteits- en logistieke sector in Gelderland, Brabant, Limburg en Noordrijn-Westfalen.

1. Provincie Gelderland (coördinatie, administratie), 2. IKEM (juridisch advies), 3. HAN (kennisuitwisseling), 4. Spring Innovation (projectmanagement, ontwikkeling voertuig), 5. Veenis Professional Services (coördinatie, kennisontwikkeling), 6. Van Boekel (infrastructuur aanpassingen), 7. ASEAG (living lab OV Aken), 8. TU Delft (softwareontwikkeling & testen), 9. V-tron (wegverkeersmanagement), 10. FromAtoB Public Design (industrieel ontwerp voertuig en halte), 11. Euregio Rhein Waal (administratie), 12. DYNTEQ (industrieel ontwerp), 13. Royal Haskoning DHV (route selectie en -ontwikkeling), 14. Robot Engineered Services (hardware en software autonomous driving), 15. Robot Care Systems (systeemintegratie), 16. Zwart UG (marktstrategie), 17. BBH (juridisch advies), 18. Noot personenvervoer (voertuigbeheer), 19. Airport Weeze Flughafen Niederrhein (living lab airport shuttle), 20. Urban Mobility Systems (voertuigontwikkeling), 21. Qing Mechatronics (hardware en software ontwikkeling), 22. Into EU (projectmanagement platooning)

Figure A.1: Flyer of the Interregional Automated Transport project (Dutch). Retrieved from: [33].

B
Fault Tree Analyses (FTA) WEpods modules

Two fault tree analyses (FTA) are performed for the WEpod modules, namely for the object detection module and the localization module, as shown in Figure B.1 and Figure B.2 respectively.

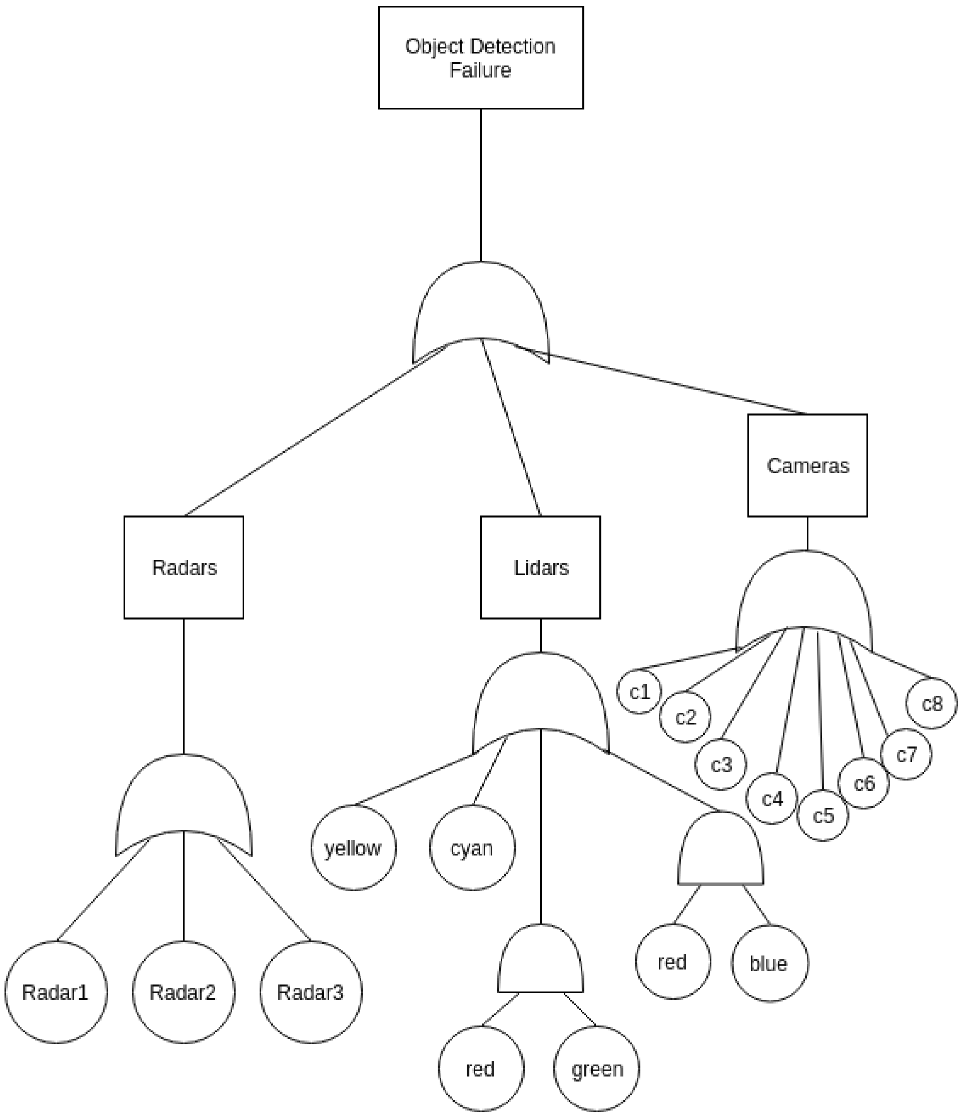


Figure B.1: FTA diagram object detection module in WEpod.

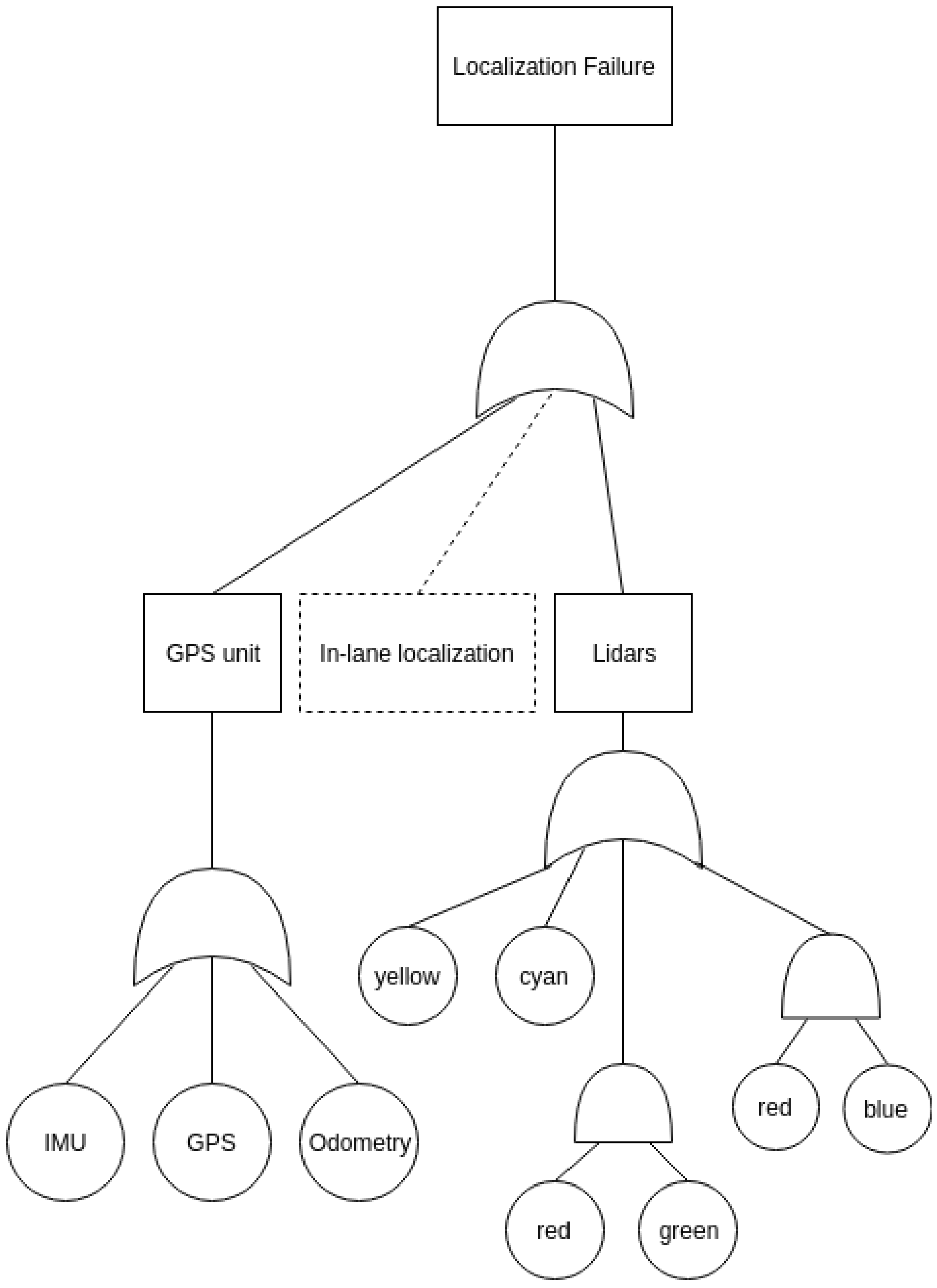


Figure B.2: FTA diagram localization module in WEpod.

C
Failure Mode Effects Analysis (FMEA)

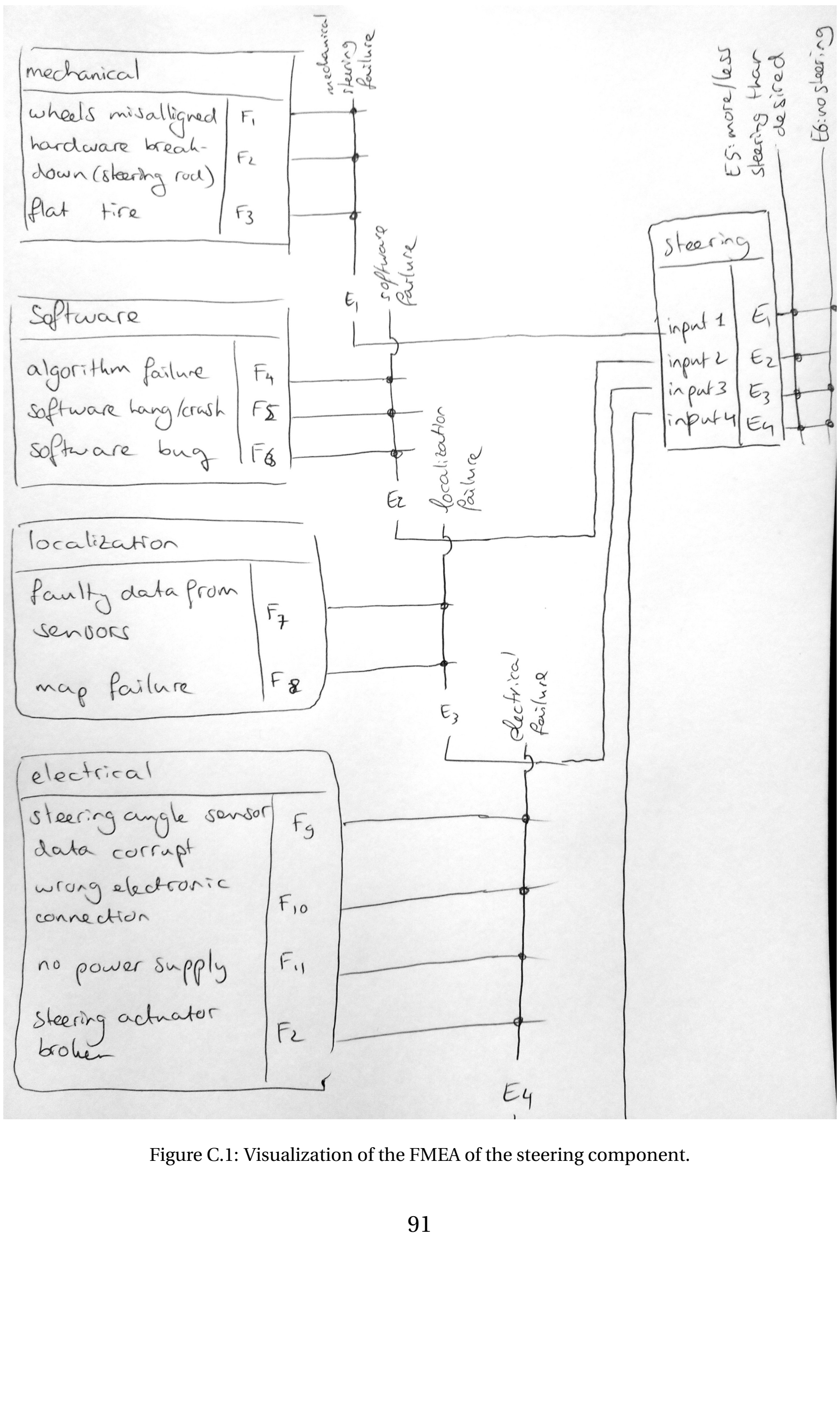


Figure C.1: Visualization of the FMEA of the steering component.

D
Relevant rosbag topics and messages

The relevant rosbag topics and messages for the localization module are listed below.

- /vehicle_can_bridge/commands - *weasy_can_msgs/Commands*
  SpeedMps
  SteerFrontRad = SteerBackRad
  Stamp.Sec and Stamp.Nsec
- /vehicle_can_bridge/measurements - *weasy_can_msgs/Measurements*
  SpeedMps
  SteerFrontRad = SteerBackRad
  Stamp.Sec and Stamp.Nsec
- /pathplanner/trajectory/controller - *wepods_msgs/Trajectory*
  Points(1).Pose.X, Points(1).Pose.Y, Points(1).Pose.Theta
  Header.Stamp.Sec and Header.Stamp.Nsec
  Points(1).Speed.X2, Points(1).Speed.Y2
- /advnav/gps_localization - *wepods_msgs/Localization*
  XEasting, YNorthing, Heading
  Header.Stamp.Sec and Header.Stamp.Nsec
  XXCov, YYCov, XYCov
  VxStd, VyStd, HeadingStd
  Vx, Vy
- /advnav/wgs84 - *sensor_msgs/NavSatFix*
  Latitude, Longitude
  PositionCovariance
  Header.Stamp and Header.NStamp
- /ibeo_republisher/gps_localization - *wepods_msgs/Localization*
  XEasting, YNorthing, Heading
  Header.Stamp.Sec and Header.Stamp.Nsec
  XXCov, YYCov, XYCov
  HeadingStd
  Vx, Vy
- /ibeo_republisher/localization - *wepods_msgs/Localization*
  XEasting, YNorthing, Heading
  Header.Stamp.Sec and Header.Stamp.Nsec
  XXCovLocal, YYCovLocal, XYCovLocal
  VxStdLocal, HeadingStd
  Vx, Vy

E

M-file: *read_in_data_from_rosbags.m*

```matlab
%% This script is intended to read in the relevant rosbag data from the
    pathplanner_trajectory_controller 2018-12-06-10-47-05_3
% Rebecca Kossen
% RRC, TU Delft
% 06-03-2019

close all
clear all
clc

[sel_vcbc_bag,path] = uigetfile('vehicle_can_bridge_commands*.bag','Select vehicle
    can bridge commands, use same number for all to open');
[~,name_vcbc,~]=fileparts([path,sel_vcbc_bag]);
bag_vcbc = rosbag([path,sel_vcbc_bag]);
bSel_vcbc = select(bag_vcbc,'Topic','/vehicle_can_bridge/commands');
msgStructs_vcbc = readMessages(bSel_vcbc,'DataFormat','struct');

[b,c] = strtok(name_vcbc,'2');

sel_vcbm_bag = dir(strcat(path,'vehicle_can_bridge_measurements_',c(1:end-5),'*',c(
    end),'.bag'));
name_vcbm = strtok(sel_vcbm_bag.name,'.');
bag_vcbm = rosbag([path,sel_vcbm_bag.name]);
bSel_vcbm = select(bag_vcbm,'Topic','/vehicle_can_bridge/measurements');
msgStructs_vcbm = readMessages(bSel_vcbm,'DataFormat','struct');

sel_ptc_bag = dir(strcat(path,'pathplanner_trajectory_controller*',c(1:end-5),'*',c(
    end),'.bag'));
name_ptc = strtok(sel_ptc_bag.name,'.');
bag_ptc  = rosbag([path,sel_ptc_bag.name]);
bSel_ptc = select(bag_ptc,'Topic','/pathplanner/trajectory/controller');
msgStructs_ptc = readMessages(bSel_ptc,'DataFormat','struct');

sel_adv_gps_bag = dir(strcat(path,'advnav_gps_localization*',c(1:end-5),'*',c(end),'
    .bag'));
name_adv_gps = strtok(sel_adv_gps_bag.name,'.');
bag_adv_gps = rosbag([path,sel_adv_gps_bag.name]);
bSel_adv_gps = select(bag_adv_gps,'Topic','/advnav/gps_localization');
msgStructs_adv_gps = readMessages(bSel_adv_gps,'DataFormat','struct');

```

```
sel_ibeo_gps_bag = dir(strcat(path,'ibeo_republisher_gps_localization*',c(1:end-5),'
    *',c(end),'.bag'));
name_ibeo_gps = strtok(sel_ibeo_gps_bag.name,'.');
bag_ibeo_gps = rosbag([path,sel_ibeo_gps_bag.name]);
bSel_ibeo_gps = select(bag_ibeo_gps,'Topic','/ibeo_republisher/gps_localization');
msgStructs_ibeo_gps = readMessages(bSel_ibeo_gps,'DataFormat','struct');

sel_ibeo_loc_bag = dir(strcat(path,'ibeo_republisher_localization*',c(1:end-5),'*',c
    (end),'.bag'));
name_ibeo_loc = strtok(sel_ibeo_loc_bag.name,'.');
bag_ibeo_loc = rosbag([path,sel_ibeo_loc_bag.name]);
bSel_ibeo_loc = select(bag_ibeo_loc,'Topic','/ibeo_republisher/localization');
msgStructs_ibeo_loc = readMessages(bSel_ibeo_loc,'DataFormat','struct');

%bag_adv_odom = rosbag('D:\backup_rebecca_30_01_2019\Documents\thesis\MATLAB\data\
    advnav_odom_2018-12-06-11-27-05_11.bag');
%bSel_adv_odom = select(bag_adv_odom,'Topic','/advnav/odom');
%msgStructs_adv_odom = readMessages(bSel_adv_odom,'DataFormat','struct');

sel_adv_wgs84_bag = dir(strcat(path,'advnav_wgs84*',c(1:end-5),'*',c(end),'.bag'));
name_adv_wgs84 = strtok(sel_adv_wgs84_bag.name,'.');
bag_adv_wgs84 = rosbag([path,sel_adv_wgs84_bag.name]);
bSel_adv_wgs84 = select(bag_adv_wgs84,'Topic','/advnav/wgs84');
msgStructs_adv_wgs84 = readMessages(bSel_adv_wgs84,'DataFormat','struct');

%% Initialize empty matrices
x_ptc = [];               x_adv_gps = [];          x_ibeo_gps  =[];
x_ibeo_loc = [];          x_adv_wgs84 = [];
y_ptc = [];               y_adv_gps = [];          y_ibeo_gps  =[];
y_ibeo_loc = [];          y_adv_wgs84 = [];
theta_ptc = [];           theta_adv_gps = [];      theta_ibeo_gps  =[];
theta_ibeo_loc = [];

ts_ptc  = [];             ts_adv_gps = [];         ts_ibeo_gps  =[];
ts_ibeo_loc = [];         ts_adv_wgs84 = [];       ts_vcbc  =[];
ts_vcbm  = [];

xx_cov_adv_gps = [];      xy_cov_adv_gps = [];     yy_cov_adv_gps = [];
theta_std_adv_gps = [];  xx_cov_ibeo_gps = [];    xy_cov_ibeo_gps = [];
yy_cov_ibeo_gps = [];    theta_std_ibeo_gps = [];xx_lcov_ibeo_loc = [];
xy_lcov_ibeo_loc = [];   yy_lcov_ibeo_loc = [];   theta_std_ibeo_loc = [];
cov_pos_adv_wgs84 = []; vx_std_adv_gps = [];     vy_std_adv_gps = [];
vx_lstd_ibeo_loc = [];


v_adv_gps = [];           v_ibeo_gps = [];         v_ibeo_loc = [];
v_ptc   = [];             v_des = [];              delta_des = [];
v_meas = [];              delta_meas = [];

% Fill the arrays with data
% Vehicle_can_bus_commands
for j = 1:length(msgStructs_vcbc)
    v_des = [v_des msgStructs_vcbc{j,1}.SpeedMps];
    delta_des = [delta_des msgStructs_vcbc{j,1}.SteerFrontRad];
    ts_vcbc = [ts_vcbc double(msgStructs_vcbc{j,1}.Stamp.Sec)+1e-9*double(
        msgStructs_vcbc{j,1}.Stamp.Nsec)];
```

```
end
ts0_vcbc = ts_vcbc-ts_vcbc(1);

% Vehicle_can_bus_measurements
for j = 1:length(msgStructs_vcbm)
    v_meas = [v_meas msgStructs_vcbm{j,1}.SpeedMps];
    delta_meas = [delta_meas msgStructs_vcbm{j,1}.SteerFrontRad];
    ts_vcbm = [ts_vcbm double(msgStructs_vcbm{j,1}.Stamp.Sec)+1e-9*double(
        msgStructs_vcbm{j,1}.Stamp.Nsec)];
end
ts0_vcbm = ts_vcbm-ts_vcbm(1);

% Pathplanner_trajectory_controller
for j = 1:length(msgStructs_ptc)
        x_ptc = [x_ptc msgStructs_ptc{j,1}.Points(1).Pose.X];
        y_ptc = [y_ptc msgStructs_ptc{j,1}.Points(1).Pose.Y];
        theta_ptc = [theta_ptc msgStructs_ptc{j,1}.Points(1).Pose.Theta];
        ts_ptc = [ts_ptc double(msgStructs_ptc{j,1}.Header.Stamp.Sec)+1e-9*double(
            msgStructs_ptc{j,1}.Header.Stamp.Nsec)];
        v_ptc = [v_ptc sqrt((msgStructs_ptc{j,1}.Points(1).Speed.X^2)+(
            msgStructs_ptc{j,1}.Points(1).Speed.Y^2))];
end
ts0_ptc = ts_ptc-ts_ptc(1);

% Advnav_gps_localization
for j = 1:length(msgStructs_adv_gps)
        x_adv_gps = [x_adv_gps msgStructs_adv_gps{j,1}.XEasting];
        y_adv_gps = [y_adv_gps msgStructs_adv_gps{j,1}.YNorthing];
        theta_adv_gps = [theta_adv_gps msgStructs_adv_gps{j,1}.Heading];
        ts_adv_gps = [ts_adv_gps double(msgStructs_adv_gps{j,1}.Header.Stamp.Sec)+1e
            -9*double(msgStructs_adv_gps{j,1}.Header.Stamp.Nsec)];
        xx_cov_adv_gps = [xx_cov_adv_gps msgStructs_adv_gps{j,1}.XXCov];
        xy_cov_adv_gps = [xy_cov_adv_gps msgStructs_adv_gps{j,1}.XYCov];
        yy_cov_adv_gps = [yy_cov_adv_gps msgStructs_adv_gps{j,1}.YYCov];
        vx_std_adv_gps = [vx_std_adv_gps msgStructs_adv_gps{j,1}.VxStd];
        vy_std_adv_gps = [vy_std_adv_gps msgStructs_adv_gps{j,1}.VyStd];
        theta_std_adv_gps = [theta_std_adv_gps msgStructs_adv_gps{j,1}.HeadingStd];
        v_adv_gps = [v_adv_gps (sqrt((msgStructs_adv_gps{j,1}.Vx)^2+(
            msgStructs_adv_gps{j,1}.Vy)^2))];
end
ts0_adv_gps = ts_adv_gps-ts_adv_gps(1);
%for i = 1:length(msgStructs{1,1}.Points)
%    pos_y = [pos_y msgStructs{1,1}.Points(i).Pose.Y];
%end

%for i = 1:length(msgStructs{1,1}.Points)
%    theta = [theta msgStructs{1,1}.Points(i).Pose.Theta];
%end

% Ibeo_republisher_gps_localization  - in UTM coordinates
for j = 1:length(msgStructs_ibeo_gps)
        x_ibeo_gps = [x_ibeo_gps msgStructs_ibeo_gps{j,1}.XEasting];
        y_ibeo_gps = [y_ibeo_gps msgStructs_ibeo_gps{j,1}.YNorthing];
        theta_ibeo_gps = [theta_ibeo_gps msgStructs_ibeo_gps{j,1}.Heading];
        ts_ibeo_gps = [ts_ibeo_gps double(msgStructs_ibeo_gps{j,1}.Header.Stamp.Sec)
            +1e-9*double(msgStructs_ibeo_gps{j,1}.Header.Stamp.Nsec)];
        xx_cov_ibeo_gps = [xx_cov_ibeo_gps msgStructs_ibeo_gps{j,1}.XXCov];
        xy_cov_ibeo_gps = [xy_cov_ibeo_gps msgStructs_ibeo_gps{j,1}.XYCov];
        yy_cov_ibeo_gps = [yy_cov_ibeo_gps msgStructs_ibeo_gps{j,1}.YYCov];
        theta_std_ibeo_gps = [theta_std_ibeo_gps msgStructs_ibeo_gps{j,1}.HeadingStd
            ];
        if isnan(msgStructs_ibeo_gps{j,1}.Vx) || isnan(msgStructs_ibeo_gps{j,1}.Vy)
        else
            v_ibeo_gps = [v_ibeo_gps (sqrt((msgStructs_ibeo_gps{j,1}.Vx)^2+(
                msgStructs_ibeo_gps{j,1}.Vy)^2))];
        end
end
ts0_ibeo_gps = ts_ibeo_gps-ts_ibeo_gps(1);

%Ibeo_republisher_localization  - in UTM coordinates
for j = 1:length(msgStructs_ibeo_loc)
        x_ibeo_loc = [x_ibeo_loc msgStructs_ibeo_loc{j,1}.XEasting];
        y_ibeo_loc = [y_ibeo_loc msgStructs_ibeo_loc{j,1}.YNorthing];
        theta_ibeo_loc = [theta_ibeo_loc msgStructs_ibeo_loc{j,1}.Heading];
        ts_ibeo_loc = [ts_ibeo_loc double(msgStructs_ibeo_loc{j,1}.Header.Stamp.Sec)
            +1e-9*double(msgStructs_ibeo_loc{j,1}.Header.Stamp.Nsec)];
        xx_lcov_ibeo_loc = [xx_lcov_ibeo_loc msgStructs_ibeo_loc{j,1}.XXCovLocal];
        xy_lcov_ibeo_loc = [xy_lcov_ibeo_loc msgStructs_ibeo_loc{j,1}.XYCovLocal];
        yy_lcov_ibeo_loc = [yy_lcov_ibeo_loc msgStructs_ibeo_loc{j,1}.YYCovLocal];
        vx_lstd_ibeo_loc = [vx_lstd_ibeo_loc msgStructs_ibeo_loc{j,1}.VxStdLocal];
        theta_std_ibeo_loc = [theta_std_ibeo_loc msgStructs_ibeo_loc{j,1}.HeadingStd
            ];
        v_ibeo_loc = [v_ibeo_loc (sqrt((msgStructs_ibeo_loc{j,1}.Vx)^2+(
            msgStructs_ibeo_loc{j,1}.Vy)^2))];
end
ts0_ibeo_loc = ts_ibeo_loc-ts_ibeo_loc(1);

%Advnav_wgs84  - in WGS84 coordinates
for j = 1:length(msgStructs_adv_wgs84)
        x_adv_wgs84 = [x_adv_wgs84 msgStructs_adv_wgs84{j,1}.Latitude];
        y_adv_wgs84 = [y_adv_wgs84 msgStructs_adv_wgs84{j,1}.Longitude];
        cov_pos_adv_wgs84 = [cov_pos_adv_wgs84 msgStructs_adv_wgs84{j,1}.
            PositionCovariance];
        ts_adv_wgs84 = [ts_adv_wgs84 double(msgStructs_adv_wgs84{j,1}.Header.Stamp.
            Sec)+1e-9*double(msgStructs_adv_wgs84{j,1}.Header.Stamp.Nsec)];
end
ts0_adv_wgs84 = ts_adv_wgs84-ts_adv_wgs84(1);

% This script is intended to convert data to correct coordinate systems
utmstruct        = defaultm('utm');
utmstruct.zone   = '32N';
utmstruct.geoid = wgs84Ellipsoid;
utmstruct        = defaultm(utmstruct);

lon = y_adv_wgs84; lat = x_adv_wgs84;

[x_adv_wgs84_utm,y_adv_wgs84_utm]=mfwdtran(utmstruct,lat,lon);

%% Determine the noise
max_noise_adv_gps    = determine_noise(xx_cov_adv_gps,yy_cov_adv_gps,
    theta_std_adv_gps);
max_noise_ibeo_gps   = determine_noise(xx_cov_ibeo_gps,yy_cov_ibeo_gps,
    theta_std_ibeo_gps);
max_noise_ibeo_loc  = determine_noise(xx_lcov_ibeo_loc,yy_lcov_ibeo_loc,
    theta_std_ibeo_loc);
noise_adv_gps.xx_cov        = xx_cov_adv_gps;
noise_adv_gps.yy_cov        = yy_cov_adv_gps;
noise_adv_gps.theta_std     = theta_std_adv_gps;
noise_adv_gps.vx_std        = vx_std_adv_gps;
noise_adv_gps.vy_std        = vy_std_adv_gps;
noise_adv_wgs84             = cov_pos_adv_wgs84;
noise_ibeo_gps.xx_cov       = xx_cov_ibeo_gps;
noise_ibeo_gps.yy_cov       = yy_cov_ibeo_gps;
noise_ibeo_loc.xx_lcov      = xx_lcov_ibeo_loc;
noise_ibeo_loc.yy_lcov      = yy_lcov_ibeo_loc;
noise_ibeo_loc.xy_lcov      = xy_lcov_ibeo_loc;
noise_ibeo_loc.theta_std    = theta_std_ibeo_loc;
noise_ibeo_loc.vx_lstd      = vx_lstd_ibeo_loc;


%% Compare some of the data outputs
figure
hold on
plot(ts0_ptc,x_ptc)
plot(ts0_adv_gps,x_adv_gps)
plot(ts0_adv_wgs84,x_adv_wgs84_utm)
plot(ts0_ibeo_gps,x_ibeo_gps)
plot(ts0_ibeo_loc,x_ibeo_loc)
title('driven distance east in UTM coordinate frame')
xlabel('time [s]')
ylabel('distance [m]')
legend('desired trajectory','advnav gps localization','advnav wgs84 localization','
    ibeo gps localization','ibeo localization')

figure
hold on
plot(ts0_ptc,y_ptc)
plot(ts0_adv_gps,y_adv_gps)
plot(ts0_adv_wgs84,y_adv_wgs84_utm)
plot(ts0_ibeo_gps,y_ibeo_gps)
plot(ts0_ibeo_loc,y_ibeo_loc)
title('driven distance north in UTM coordinate frame')
xlabel('time [s]')
ylabel('distance [m]')
legend('desired trajectory','advnav gps localization','advnav wgs84 localization','
    ibeo gps localization','ibeo localization')

figure
hold on
plot(ts0_ptc,theta_ptc)
plot(ts0_adv_gps,theta_adv_gps)
plot(ts0_ibeo_gps,theta_ibeo_gps)
plot(ts0_ibeo_loc,theta_ibeo_loc)
title('orientation of the vehicle')
xlabel('time [s]')
ylabel('angle [rad]')
legend('desired trajectory','advnav gps localization','ibeo gps localization','ibeo
    localization')
```

```

figure
hold on
plot(ts0_ptc,v_ptc)
plot(ts0_vcbc,v_des)
plot(ts0_vcbm,v_meas)
plot(ts0_adv_gps,v_adv_gps)
plot(ts0_ibeo_loc,v_ibeo_loc)
if isempty(v_ibeo_gps)
else
plot(ts0_ibeo_gps,v_ibeo_gps)
end
title('actual speed of the vehicle')
xlabel('time [s]')
ylabel('speed [m/s]')
legend('desired trajectory automation','input:v\_des','can bus measurements','advnav
    gps localization','ibeo localization','ibeo gps localization')

% Vehicle can bridge comparison
figure
hold on
title('vehicle can bridge commands compared to measurements')
xlabel('time [s]')
ylabel('angle [rad]')
plot(ts0_vcbm,delta_meas)
plot(ts0_vcbc,delta_des)
legend('angle measured','angle command')

figure
hold on
title('vehicle can bridge commands compared to measurements')
xlabel('time [s]')
ylabel('speed [m/s]')
plot(ts0_vcbm,v_meas)
plot(ts0_vcbc,v_des)
legend('speed measured','speed command')

%% Save the data
saved = 0;      % initialize, if succesfully saved, this value changes to 1

old = "-";  % MATLAB recognizes - as a minus sign, so it needs to be changed
new = "_";  % the symbol used instead is an underscore _

% vehicle can bridge commands
assignin('base',strcat('data_',replace(name_vcbc,old,new)),[double(ts_vcbc)',double(
    v_des)',double(delta_des)']);
save(strcat('data_',replace(name_vcbc,old,new),'.mat'),strcat('data_',replace(
    name_vcbc,old,new)));

% vehicle can bridge measurements
assignin('base',strcat('data_',replace(name_vcbm,old,new)),[double(ts_vcbm)',double(
    v_meas)',double(delta_meas)']);
save(strcat('data_',replace(name_vcbm,old,new),'.mat'),strcat('data_',replace(
    name_vcbm,old,new)));

% pathplanner trajectory controller
```

```
oldptc       = "pathplanner_trajectory_controller";
newptc       = "ptc";
name_ptc     = replace(name_ptc,oldptc,newptc);
assignin('base',strcat('data_',replace(name_ptc,old,new),'_corr'),[double(ts_ptc)',
    double(x_ptc)', double(y_ptc)', double(theta_ptc)', double(v_ptc)']);
save(strcat('data_',replace(name_ptc,old,new),'_corr.mat'),strcat('data_',replace(
    name_ptc,old,new),'_corr'));
assignin('base',strcat('data_',replace(name_ptc,old,new),'_norm'),double([double(
    ts0_ptc)',double(x_ptc)'-double(x_ptc(1)), double(y_ptc)'-double(y_ptc(1)),
    double(theta_ptc)'-double(theta_ptc(1)), double(v_ptc)'-double(v_ptc(1))']));
save(strcat('data_',replace(name_ptc,old,new),'_norm.mat'),strcat('data_',replace(
    name_ptc,old,new),'_norm'));

% advnav gps
assignin('base',strcat('data_',replace(name_adv_gps,old,new),'_corr'),[double(
    ts_adv_gps)',double(x_adv_gps)',double(y_adv_gps)',double(theta_adv_gps)',double(
    v_adv_gps)']);
save(strcat('data_',replace(name_adv_gps,old,new),'_corr.mat'),strcat('data_',
    replace(name_adv_gps,old,new),'_corr'));
assignin('base',strcat('data_',replace(name_adv_gps,old,new),'_norm'),double([
    ts0_adv_gps',x_adv_gps'-x_adv_gps(1), y_adv_gps'-y_adv_gps(1), theta_adv_gps'-
    theta_adv_gps(1), v_adv_gps']));
save(strcat('data_',replace(name_adv_gps,old,new),'_norm.mat'),strcat('data_',
    replace(name_adv_gps,old,new),'_norm'));
assignin('base',strcat('max_data_noise_',replace(name_adv_gps,old,new)),double(
    max_noise_adv_gps));
save(strcat('max_data_noise_',replace(name_adv_gps,old,new),'.mat'),strcat('
    max_data_noise_',replace(name_adv_gps,old,new)));
assignin('base',strcat('data_noise_',replace(name_adv_gps,old,new)),noise_adv_gps);
save(strcat('data_noise_',replace(name_adv_gps,old,new),'.mat'),strcat('data_noise_'
    ,replace(name_adv_gps,old,new)));

% advnav wgs84
assignin('base',strcat('data_',replace(name_adv_wgs84,old,new),'_corr'),[double(
    ts_adv_wgs84)',double(x_adv_wgs84)',double(y_adv_wgs84)']);
save(strcat('data_',replace(name_adv_wgs84,old,new),'_corr.mat'),strcat('data_',
    replace(name_adv_wgs84,old,new),'_corr'));
assignin('base',strcat('data_',replace(name_adv_wgs84,old,new),'_norm'),double([
    ts0_adv_wgs84',x_adv_wgs84'-x_adv_wgs84(1),y_adv_wgs84'-y_adv_wgs84(1)]));
save(strcat('data_',replace(name_adv_wgs84,old,new),'_norm.mat'),strcat('data_',
    replace(name_adv_wgs84,old,new),'_norm'));
assignin('base',strcat('data_noise_cov_diag',replace(name_adv_wgs84,old,new)),double
    (noise_adv_wgs84));
save(strcat('data_noise_cov_diag',replace(name_adv_wgs84,old,new),'.mat'),strcat('
    data_noise_cov_diag',replace(name_adv_wgs84,old,new)));

% ibeo gps
oldibeogps = "ibeo_republisher_gps_localization";
newibeogps = "ibeo_gps";
name_ibeo_gps = replace(name_ibeo_gps,oldibeogps,newibeogps);
if isempty(v_ibeo_gps)
    assignin('base',strcat('data_',replace(name_ibeo_gps,old,new),'_corr'),[double(
        ts_ibeo_gps); double(x_ibeo_gps) ; double(y_ibeo_gps) ; double(theta_ibeo_gps
        ) ]');
    save(strcat('data_',replace(name_ibeo_gps,old,new),'_corr.mat'),strcat('data_',
        replace(name_ibeo_gps,old,new),'_corr'));
```

```matlab
    assignin('base',strcat('data_',replace(name_ibeo_gps,old,new),'_norm'),double([ts0_ibeo_gps; x_ibeo_gps-x_ibeo_gps(1); y_ibeo_gps-y_ibeo_gps(1); theta_ibeo_gps-theta_ibeo_gps(1)]'));
    save(strcat('data_',replace(name_ibeo_gps,old,new),'_norm.mat'),strcat('data_',replace(name_ibeo_gps,old,new),'_norm'));
else
    assignin('base',strcat('data_',replace(name_ibeo_gps,old,new),'_corr'),[double(ts_ibeo_gps); double(x_ibeo_gps) ; double(y_ibeo_gps) ; double(theta_ibeo_gps) ; double(v_ibeo_gps) ]);
    save(strcat('data_',replace(name_ibeo_gps,old,new),'_corr.mat'),strcat('data_',replace(name_ibeo_gps,old,new),'_corr'));
    assignin('base',strcat('data_',replace(name_ibeo_gps,old,new),'_norm'),double([ts0_ibeo_gps;x_ibeo_gps-x_ibeo_gps(1); y_ibeo_gps-y_ibeo_gps(1); theta_ibeo_gps-theta_ibeo_gps(1) ; v_ibeo_gps ]));
    save(strcat('data_',replace(name_ibeo_gps,old,new),'_norm.mat'),strcat('data_',replace(name_ibeo_gps,old,new),'_norm'));
end
assignin('base',strcat('max_data_noise_',replace(name_ibeo_gps,old,new)),double(max_noise_ibeo_gps));
save(strcat('max_data_noise_',replace(name_ibeo_gps,old,new),'.mat'),strcat('max_data_noise_',replace(name_ibeo_gps,old,new)));
assignin('base',strcat('data_noise_',replace(name_ibeo_gps,old,new)),noise_ibeo_gps);
save(strcat('data_noise_',replace(name_ibeo_gps,old,new),'.mat'),strcat('data_noise_',replace(name_ibeo_gps,old,new)));

% ibeo loc
oldibeoloc = "ibeo_republisher_localization";
newibeoloc = "ibeo_loc";
name_ibeo_loc = replace(name_ibeo_loc,oldibeoloc,newibeoloc);
if isempty(v_ibeo_loc)
    assignin('base',strcat('data_',replace(name_ibeo_loc,old,new),'_corr'),[double(ts_ibeo_loc); double(x_ibeo_loc) ; double(y_ibeo_loc) ; double(theta_ibeo_loc) ]');
    save(strcat('data_',replace(name_ibeo_loc,old,new),'_corr.mat'),strcat('data_',replace(name_ibeo_loc,old,new),'_corr'));
    assignin('base',strcat('data_',replace(name_ibeo_loc,old,new),'_norm'),double([ts0_ibeo_loc; x_ibeo_loc-x_ibeo_loc(1); y_ibeo_loc-y_ibeo_loc(1); theta_ibeo_loc-theta_ibeo_loc(1) ]'));
    save(strcat('data_',replace(name_ibeo_loc,old,new),'_norm.mat'),strcat('data_',replace(name_ibeo_loc,old,new),'_norm'));
else
    assignin('base',strcat('data_',replace(name_ibeo_loc,old,new),'_corr'),[double(ts_ibeo_loc); double(x_ibeo_loc) ; double(y_ibeo_loc) ; double(theta_ibeo_loc) ; double(v_ibeo_loc) ]);
    save(strcat('data_',replace(name_ibeo_loc,old,new),'_corr.mat'),strcat('data_',replace(name_ibeo_loc,old,new),'_corr'));
    assignin('base',strcat('data_',replace(name_ibeo_loc,old,new),'_norm'),double([ts0_ibeo_loc; x_ibeo_loc-x_ibeo_loc(1); y_ibeo_loc-y_ibeo_loc(1); theta_ibeo_loc-theta_ibeo_gps(1); v_ibeo_loc]));
    save(strcat('data_',replace(name_ibeo_loc,old,new),'_norm.mat'),strcat('data_',replace(name_ibeo_loc,old,new),'_norm'));
end
assignin('base',strcat('max_data_noise_',replace(name_ibeo_loc,old,new)),double(max_noise_ibeo_loc));
save(strcat('max_data_noise_',replace(name_ibeo_loc,old,new),'.mat'),strcat('
    max_data_noise_',replace(name_ibeo_loc,old,new)));
assignin('base',strcat('data_noise_',replace(name_ibeo_loc,old,new)),noise_ibeo_loc)
    ;
save(strcat('data_noise_',replace(name_ibeo_loc,old,new),'.mat'),strcat('data_noise_
    ',replace(name_ibeo_loc,old,new)));

[~, kp] = strtok(name_vcbc,'2');     % prepare name and date string
[~, km] = strtok(path, 'W');         % prepare name and date string
[ko,~] = strtok(km,'1');             % prepare name and date string

saved = 1;      % if succesfully saved, this value changes to 1
if saved        % show message that data has been saved succesfully
    show = strcat('data has been saved correctly for the following logs:',ko,kp)
end
```

F Data cleaning

Input data

The vehicle can bridge data, that is used as an input to the model, was inspected in order to prepare it well before using in the rest of the approach. The measurements turned out to be gathered at a lower frequency than the commands were given. Around twice as small: ~50Hz compared to 100Hz (sampling frequency of the commands). Therefore, either the data containing the commands needed to be down-sampled or the data containing the measurements needed to be sampled up in such way that the number of samples is congruent. The choice is made for sampling up, in order to retain as much information as possible. At first, the data was simply duplicated and plots were generated, as shown in Figures F.1-F.3. Because this is not a very neat way of handling data, an update is included in the cleaning steps that interpolates the data with the lower sampling frequency.

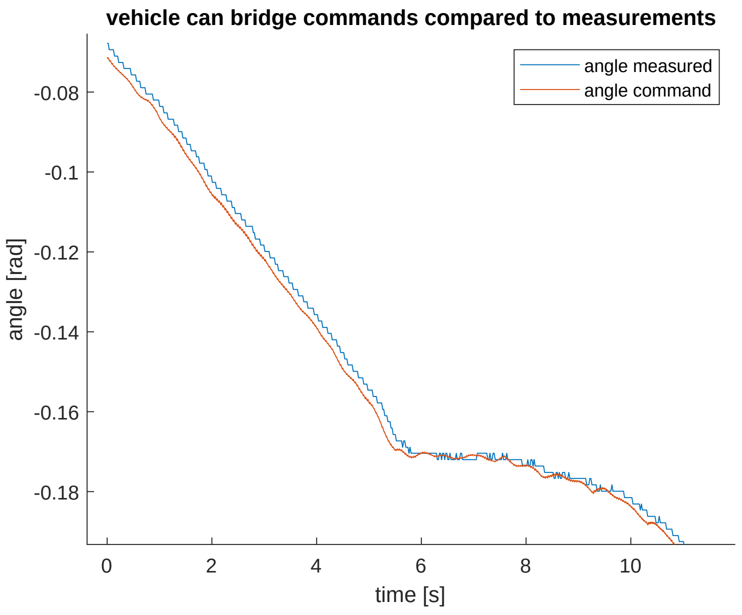


Figure F.1: Comparison between vehicle can bridge commands and measurements beginning of sequence.

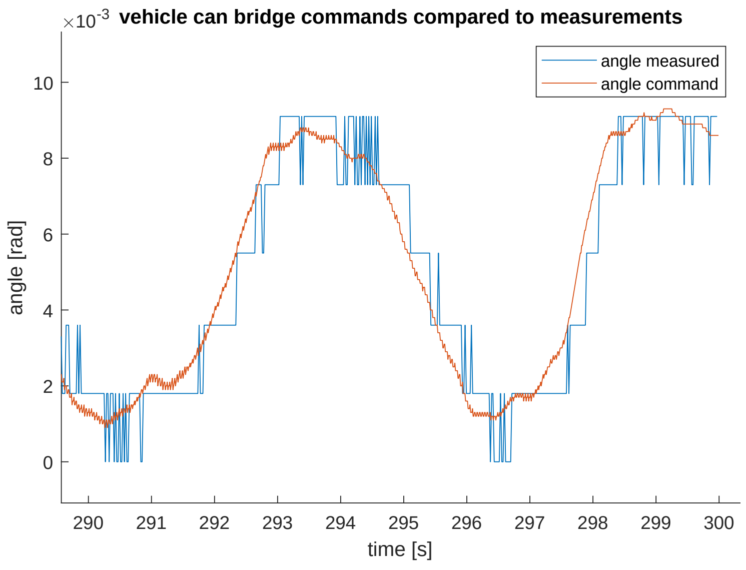


Figure F.2: Comparison between vehicle can bridge commands and measurements end of sequence.

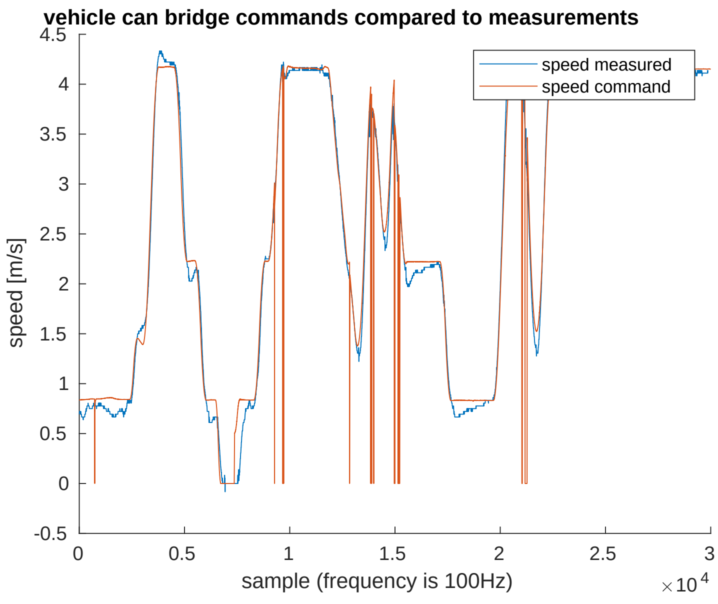


Figure F.3: Comparison between vehicle can bridge commands and measurements of the speeds.

Several other things became clear from these plots and manual inspection of the acquired .mats:

- Each log consists of data recorded during a timespan of 5 minutes;
30,000 samples at 100Hz: 30,000*1/100 = 300 seconds => 300/60 = 5 minutes.
- The measurements have an offset in Unix starting time compared to the commands;
- The can bridge speed measurements indicated some problem in the time vector by showing a more disrete type of behavior than expected;
- The speed commands drop to 0 sometimes (according to the team due to object handling/perception);
- The measured speed shows an overshoot (as a result of the control of the vehicle).

The input data is subject to a couple of cleaning steps (see also Appendix G, containing the *prepare.m* file where most of the cleaning is performed), that will be explained here successively in the next seven subsections:

Compensating for offset

The offset between the first value of the commands and the first value of the measurements is calculated by subtraction and this difference is added to the entire measurement data sequence, from which the first data entry is subtracted in the *prepare.m* file:

```
if offset_meas < 0
    v_meas = interp1(measurements(:,1)-measurements(1,1)+offset_meas,v_meas,tv)';
    delta_meas = interp1(measurements(:,1)-measurements(1,1)+offset_meas,delta_meas,
        tv)';
else
    e = 'error offset_meas'
end
```

Because the measurements will always start later than the commands, the offset will be negative always and the offset can be added. Just to make sure this is correct for all the measurements, the ‘safety if-statement’ is included.

Solving problem of ‘vertical’ values in data

At first, I did not take the messages providing the Nsecs (nanoseconds) into account. This turned out to be a shortcoming when comparing the data, so the Nsecs are retrieved and included, leading to less choppy data.

Eliminating fake 0’s from the commands

The commands return 0’s at unexpected instances. This is clearly not what happened in reality, otherwise a drop would have shown up in the measurements too. These outcomes are caused by ghost object detections. When an object is detected as an obstacle, the command is 0 for the speed, in order to prevent collisions. It requests a stop for 1 or 2 timestamps because the algorithm first has to notice that the obstacle is actually not present before it can update its command again. Therefore, it is important to eliminate these fake 0’s. This is done in *prepare.m* by finding entries of 0 that did not last longer than 3 seconds (this timespan is set as a minimum for an actual drop in the speed command, for example when it is desired that the vehicle would come to or remain in a standstill position). These ‘short-term 0’s’ are replaced by the last commanded non-zero value for the speed.

Sampling up data

Because the data is not gathered with the same sampling frequency, it is important to interpolate it in order to retrieve the same number of data, to be able to feed it into the model and compare everything well. This is done by making use of the *interp1* function in MATLAB that makes use of a linear interpolation in order to retrieve a vector with the same length as the desired time vector (which is constructed by taking the number of samples of the commands, which is 30,000 for each log). In *prepare.m* it is visible how this has been implemented exactly.

Eliminating NaNs

Of course, after interpolating, NaNs will occur when there has been no neighbouring data to perform the interpolation with at the beginning and/or end of the sequence. In order to solve this problem, a script was written that copies the last neighbouring value as many times as required (at the end a message is generated of how many NaNs needed to be replaced):

```
p = [];
for i = 1:length(v_meas)
    if isnan(v_meas(i))
        p = [p;i];
        v_meas(i,:)=v_meas(p(1)-1,:);
        delta_meas(i,:)=delta_meas(p(1)-1,:);
    end
```

```
end
mess = strcat(num2str(length(p)),' NaNs replaced in v_meas and delta_meas')
```

Instead of performing this extrapolation manually, there is an option to the *interp1*-function in MATLAB as well that takes care of this automatically. It is advised to use that functionality. It is left in because this way it works as well.

Plots after data cleaning

After cleaning the data, again a visualization is made of the commands versus the measurements. The 'prepared steering angle' is shown in Figure F.4 (zoomed in on).

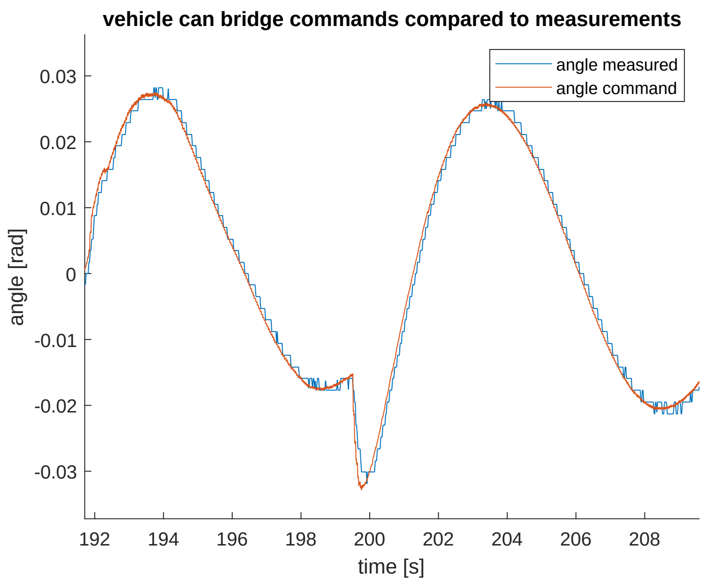


Figure F.4: Comparison between vehicle can bridge commands and measurements of the steering angle after *prepare.m*.

The difference between the measured speeds saved at seconds versus nanoseconds is shown in Figure F.5. And finally, the commanded speeds after all the preparation steps are shown in Figure F.6: the 0s have been eliminated effectively (the prepared speed is plotted on top of the commanded speed, so the 0s are gone) for the short time instances (the erroneous ones). The real speed commands for 0 m/s remained.

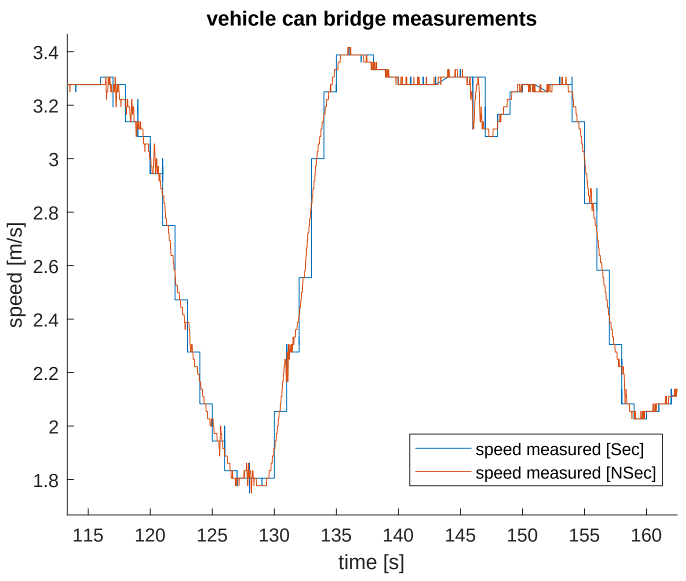


Figure F.5: Comparison between speed measurements saved at seconds versus nanoseconds.

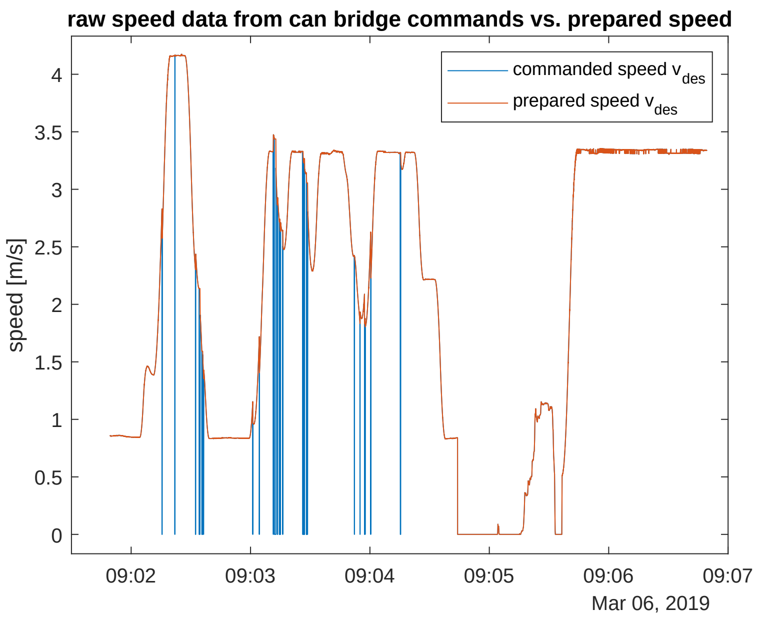


Figure F.6: Comparison between vehicle can bridge commands and measurements of the speed after *prepare.m*.

Compensating for offset, sampling up data and eliminating NaNs

Also the measurement data have an offset in the Unix time stamp compared to the commands. They have been shifted in the same way as the input data, see also Appendix G.

Localization data

The next two subsections belong to the cleaning steps for the localization data.

Sample up, eliminate NaNs, compensating for offset

All localization data turned out to be measured at a lower sampling frequency than the commands. Because the model will return as many values as the input, it is important to scale up the localization data as well in order to be able to compare well. This is done using a similar approach as for the input data. Again, the NaNs are eliminated using the same approach as before.

Wrap orientation angle

When inspecting the orientation angle coming from the localization data, it turned out that it is wrapped: kept between bounds from 0 to 2π. Sudden jumps arose each time the orientation angle crossed this 2π bound. When using the model at a later stage, this wrapping will not be automatically included. Therefore, a solution is to unwrap the data first, as is done by using the MATLAB funcion *unwrap* in the *prepare.m* file.

Sensor (fusion) noise data

The next two subsections belong to the cleaning steps for the sensor (fusion) noise data.

Sample up

The noise was acquired at a frequency four times as small as the commands were given. Because it would not make sense to interpolate the uncertainty in data, the entries in the arrays are simply replicated four times by using the MATLAB functions *repmat* and *reshape*.

Prepare covariance matrix

Furthermore, retrieving the covariance matrix turned out not to be as straightforward as expected because complex results were returned by calculation of the Mahalanobis distance. When the result is complex, this indicates negative definiteness of the covariance matrix. This error was discovered by the team of RRC before, but was never solved properly. It was manually fixed by tweaking the matrix in such a way that it would always be positive definite. An approach that was implemented first in order to prepare the covariance matrix in this application, was by replacing the xy covariance entry (as these had too high negative values) in the symmetric matrix by a 0 when the overall covariance matrix turned out to be negative definite. In the meantime, in collaboration with the team, the origin of the covariance data was inspected and how it is saved exactly in the rosbags. Not much documentation was provided by the external software piece, only that the ibeo localization provided getXPositionSigma, getYPositionSigma and getXyCorrelation. These values were saved as messages XXCovLocal, YYCovLocal and XYCovLocal respectively. This is where the error became clear: although the XyCorrelation was returned, it was stored as if it would equal the covariance in xy, which, by definition, is not the same [28]. The solution to this problem exists of calculating the correlation matrix from the xx and yy covariance matrix, which simply is a 2x2 identity matrix. Then, the correlation for xy is entered on the (1,2) and (2,1) entries and the *corr2cov* function from MATLAB has been used to find the correct covariance matrix, based on the variances in x and y and the correlation matrix. The actual script, including the preparation, is included in the *Mahalanobis.m* file, Appendix J.

Before proceeding, the covariance has been checked on validity by comparing the residual in the x-position with the covariance in the x-direction. This comparison is shown in Figure F.7.

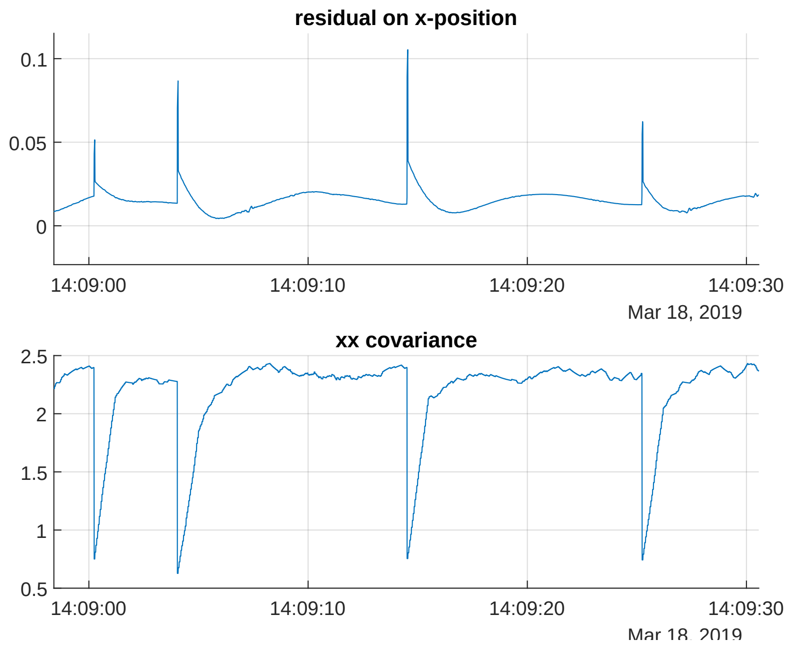


Figure F.7: Residual in x-position and its according covariance in xx-direction.

Again, something unexpected is visible from Figure F.7. At the time of a fault, a high covariance would be expected, but each time a fault occured (as indicated by a peak in the residual), a drop in the covariance is visible. This could indicate that what is defined as a fault, the jump in the residual, actually represents the end of a fault: namely a correction. When the vehicle is driving, the algorithm makes a prediction, until it gets closer and when it is finally sure of its prediction, then it jumps to the correct measurement. The covariance is low again when the algorithm had corrected itself. In order to validate this assumption, inspection again is needed of the faults that have been visualized in Figures 2.18 and 2.19. There, it becomes clear from Figure 2.18 that this is not necessarily the case, as it jumps two times in a row to a location that lies further from the actual path. So the hypothesis of the algorithm correcting itself, is rejected. There could be other reasons that the covariance suddenly 'jumps': [15] [16] [30] [37] [29].

What is of higher interest than the actual reason, is how to handle it, while it is already present (following the fault tolerant control vision). There are multiple possible approaches: one is to compute the median of all the covariances and use this fixed value for computing the Mahalanobis distance (as included in *Mahalanobis.m*) by using the *median* function in MATLAB. A more reliable approach, however, is to take all the local minima of the covariances. The assumption is made that the 'reset' values (which are small, but not equal to 0) represent the real covariance best. A MATLAB-function is written that is able to do the job by finding local minima lower than a certain pre-set value (in order to retrieve the actual minima) using the *find* and *islocalmin* functionalities:

```
function cov_min = covmin(xx,yy,xy);
% This function is written in order to retrieve the correct covariance
% matrix for the noise in the measurements, by taking the minimum values to
% which the covariances are resetted at the time of a jump.

xx_min(1) = xx(1,1);
idx = find(islocalmin(xx)&xx<2);     % find local minima in xx_cov, below 2
for i = 2:length(xx)                 % start at index 2, for taking i-1
    xx_min(i)=xx_min(i-1);
    if find(i==idx>0)
        xx_min(i)=xx(i);
    end
end
```

```

yy_min(1) = yy(1,1);
idx = find(islocalmin(yy)&yy<2);     % find local minima in yy_cov, below 2
for i = 2:length(yy)                 % start at index 2, for taking i-1
    yy_min(i)=yy_min(i-1);
    if find(i==idx>0)
        yy_min(i)=yy(i);
    end
end

xy_min(1) = xy(1,1);
idx = find(islocalmin(yy));          % take local minima from xy at same instance as
    yy
for i = 2:length(xy)                 % start at index 2, for taking i-1
    xy_min(i)=xy_min(i-1);
    if find(i==idx>0)
        xy_min(i)=xy(i);
    end
end

cov_min = [];                                    % initialize covariance matrix
for i = 1:length(xx)
cov_min = [cov_min; {[xx_min(i), xy_min(i);      % fill covariance matrix
                      xy_min(i), yy_min(i)]}];
end
end
```

A major drawback of both approaches is the inability of implementing on-line. Both computations require previous and future values of the covariance.

G
M-file: *prepare.m*

```
function [Ts,Tstop,tv,tr,posInit,v_des,delta_des,v_des_no_zero,v_meas,delta_meas] =
    prepare(data_corr,data_norm,commands,measurements)
% Prepare input [v_des; delta_des]
v_des       = commands(:,2);
delta_des   = commands(:,3);
v_meas      = measurements(:,2);
delta_meas  = measurements(:,3);

% Convert data to time series for simin blocks
nsamples    = length(v_des);              % number of samples [#]
Ts          = 0.01;                       % step time [s]
Tstop       = (nsamples-1)*Ts;            % stopping time of the simulation [s]
tv          = (0:(nsamples-1))*Ts;        % time vector [s]

% Get the data to the correct length by interpolating
if size(data_corr,2)<4
else
   data_corr(:,4)=unwrap(data_corr(:,4));
   data_norm(:,4)=unwrap(data_norm(:,4));
end
offset      = commands(1,1)-data_corr(1,1)              % compute offset with commands
    to shift data
if offset < 0
    data_corr    = interp1(data_corr(:,1)-data_corr(1)+offset,data_corr(:,2:end),tv);
             % exclude the time vector
    data_norm    = interp1(data_norm(:,1)+offset,data_norm(:,2:end),tv);
                              % exclude the time vector
else
    e = 'error offset'
end
offset_meas = (commands(1,1)-measurements(1,1)) % compute offset with commands to
    shift data
if offset_meas < 0
    v_meas       = interp1(measurements(:,1)-measurements(1,1)+offset_meas,v_meas,tv)
        ';
    delta_meas   = interp1(measurements(:,1)-measurements(1,1)+offset_meas,delta_meas
        ,tv)';
else
    e = 'error offset_meas'
end
```

```
% Get rid of NaNs; extrapolate functionality of interp1 can be used instead
p = [];
for i = 1:length(data_corr)
    if isnan(data_corr(i))
        p = [p;i];
        pext = find(p>length(data_corr)/2,1);
        psmall = find(p<length(data_corr)/2);
        if p(end) > length(data_corr)/2
            data_corr(i,:)=data_corr(p(pext)-1,:);
            data_norm(i,:)=data_norm(p(pext)-1,:)';
        end
    end
end

if isempty(psmall)
else
    for j = 1:length(p)
        data_corr(j,:)=data_corr((psmall(end)+1),:);
        data_norm(j,:)=data_norm((psmall(end)+1),:);
    end
end
if isempty(pext)
else
    if pext>1
        if p(pext-1)<length(data_corr)/2
            for i = 1:length(p)
            data_corr(i,:)=data_corr(p(pext-1)+1,:);
            data_norm(i,:)=data_norm(p(pext-1)+1,:);
            end
        end
    end
end
mess = strcat(num2str(length(p)),' NaNs replaced in data_corr and data_norm')

p = [];
for i = 1:length(v_meas)
    if isnan(v_meas(i))
        p = [p;i];
        v_meas(i,:)=v_meas(p(1)-1,:);
        delta_meas(i,:)=delta_meas(p(1)-1,:);
    end
end
mess = strcat(num2str(length(p)),' NaNs replaced in v_meas and delta_meas')

% prerequisite for 'real 0' is at least 3s 0: 3/0.01 = 300 timesteps
equ = eq(v_des,0);
arr = find(diff(equ));
d_arr = diff(arr);
for i = 1:length(d_arr)
    if d_arr(i)<300                 % 0 holds for shorter than 3 seconds
        if commands(arr(i),2)==0
        else
            ents = arr(i)+1;
            ente = arr(i+1);
            v_des(ents:ente)=v_des(arr(i));
        end
    end
end
v_des_no_zero = v_des;

tr = [];
for i = 1:size(data_corr,2)
    if i == 3
        tr = [tr,data_corr(:,i)];   % get the orientation for theta correct
    else
        tr = [tr,data_norm(:,i)];   % get normalized data for x and y
    end
end

% Set initial conditions and update tr - if the sensor doesn't have entries for
% theta, speed and steering angle, input the measurements from the can bus
posInit = [];
for i = 1:size(data_corr,2)
    posInit = [posInit,tr(1,i)];    % take initial positions from data
end
% Fill up the initial positions with values retrieved elsewhere if length =! 5
if size(posInit,2)<3        % only x and y information available (f.e.: wgs84)
    posInit = [posInit,0, v_meas(1), delta_des(1)]; % set initial orientation to 0
    tr = [tr,zeros(length(data_corr),1),v_meas,delta_meas];
elseif size(posInit,2)<4    % only x, y and theta information available
    posInit = [posInit, v_meas(1), delta_meas(1)];
    tr = [tr,v_meas,delta_meas];
elseif size(posInit,2)<5    % only x, y, theta and speed information available
    posInit = [posInit, delta_meas(1)];
    tr = [tr,delta_meas];
end

end
```

H
Subsystems observer model Simulink

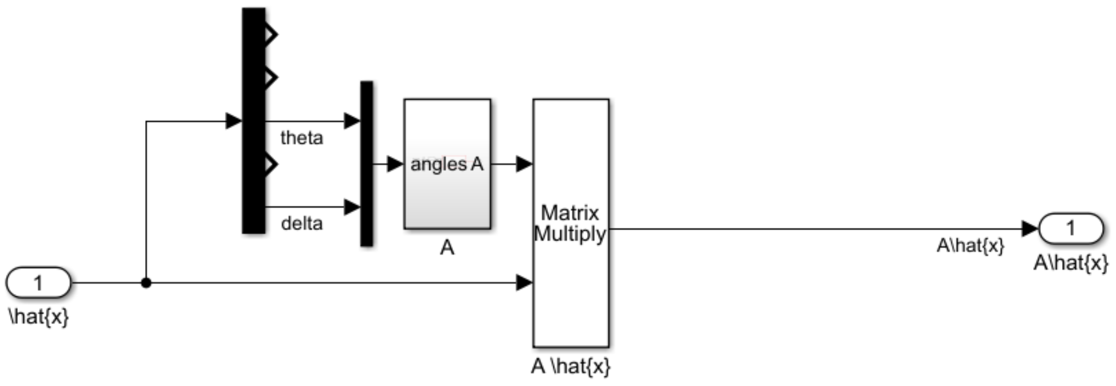


Figure H.1: Subsystem Ax of the observer model

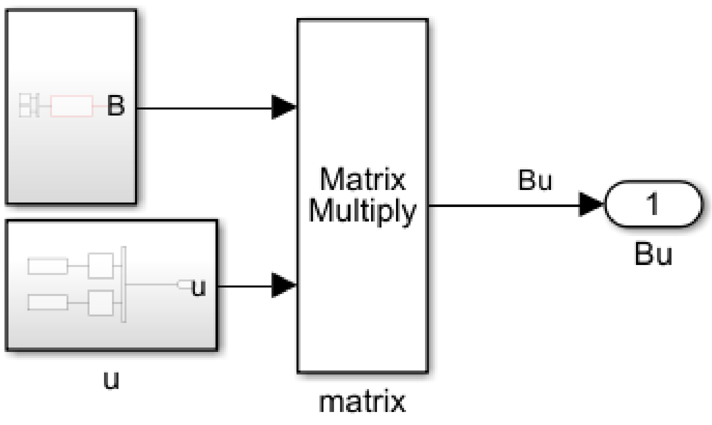


Figure H.2: Subsystem Bu of the observer model

```
function A = fcn(angles,par)
```

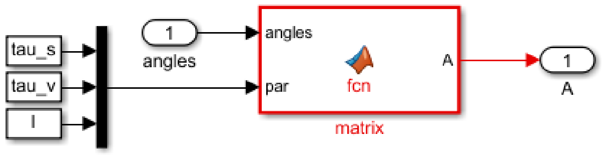


Figure H.3: Subsubsystem A of the observer model

```
tau_s = par(1);     % time delay actuators [s]
tau_v = par(2);     % time delay actuators [s]
l = par(3);         % length of half the wheelbase

A = [0  0  0  cos(angles(2))*cos(angles(1))  0;
      0  0  0  cos(angles(2))*sin(angles(1))  0;
      0  0  0  sin(angles(2))/l         0;
      0  0  0  −1/tau_v            0;
      0  0  0  0             −1/tau_s];
```

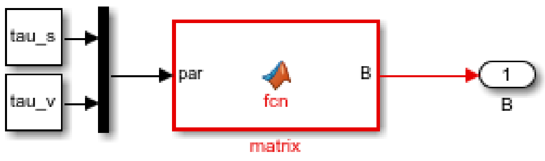


Figure H.4: Subsubsystem B of the observer model

```
function B = fcn(par)
tau_s = par(1);     % time delay actuators [s]
tau_v = par(2);     % time delay actuators [s]

B = [0  0;
      0  0;
      0  0;
      1/tau_v              0;
      0             1/tau_s];
```

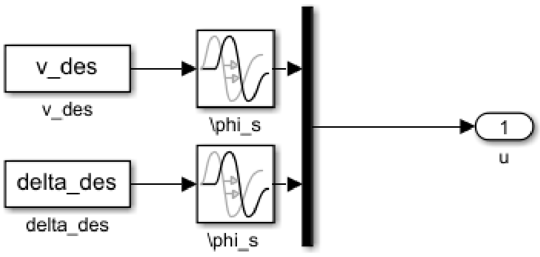


Figure H.5: Subsubsystem u of the observer model

Subsystems two observer model Simulink

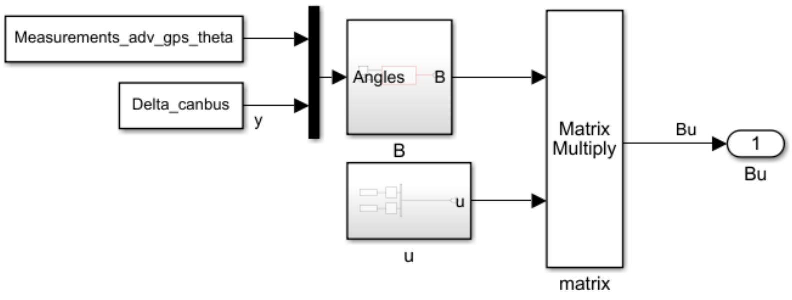


Figure I.1: Subsystem Bu of the second observer model

```
Measurements_adv_gps_theta = timeseries(tr_adv_gps(:,3),tv);
Delta_canbus               = timeseries(delta_meas,tv);
```

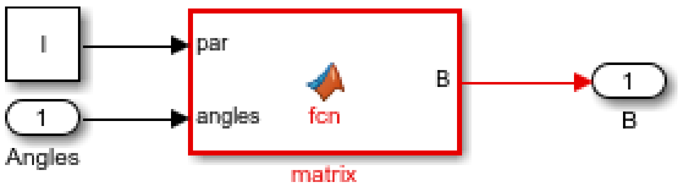


Figure I.2: Subsubsystem B of the second observer model

```
function B = fcn(par,angles)
l = par;    % length of half the wheelbase [m]

B = [cos(angles(2))*cos(angles(1)) 0;
     cos(angles(2))*sin(angles(1)) 0;
     -sin(angles(2))/l 0];
```

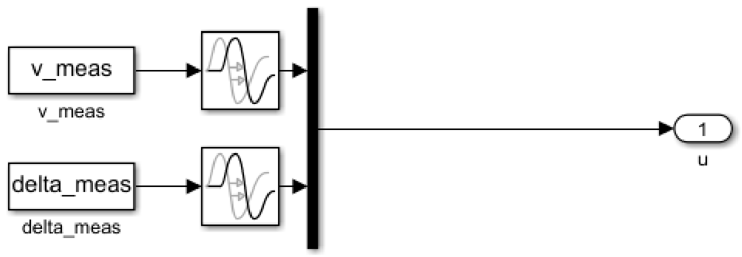


Figure I.3: Subsubsystem u of the second observer model

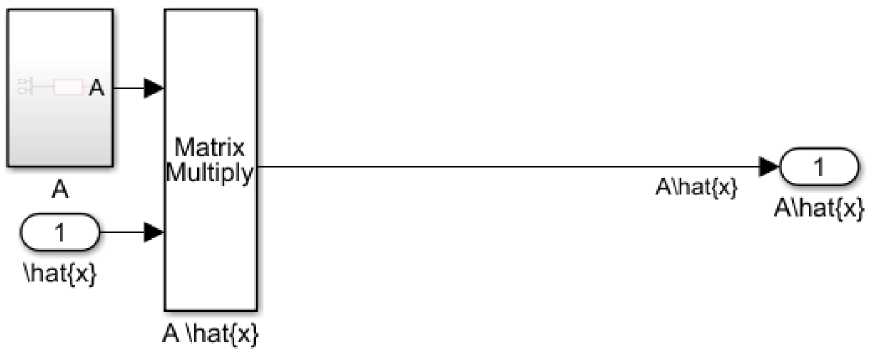


Figure I.4: Subsystem Ax of the first observer model

```
function A = fcn(par)
tau_s = par(1);     % time delay actuators [s]
tau_v = par(2);     % time delay actuators [s]

A = [-1/tau_v              0;
       0              -1/tau_s];
```

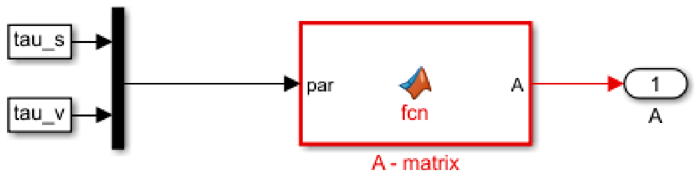


Figure I.5: Subsubsystem A of the first observer model

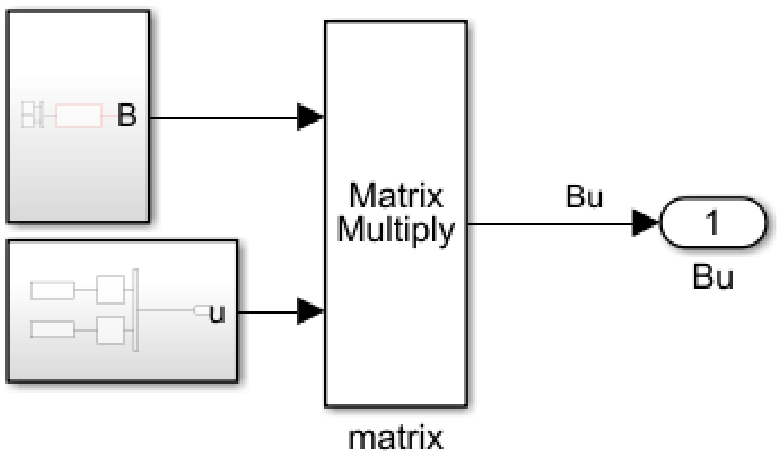


Figure I.6: Subsystem Bu of the first observer model

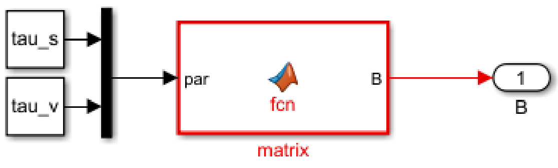


Figure I.7: Subsubsystem B of the first observer model

```
function B = fcn(par)
tau_s = par(1);     % time delay actuators [s]
tau_v = par(2);     % time delay actuators [s]

B = [1/tau_v              0;
     0               1/tau_s];
```

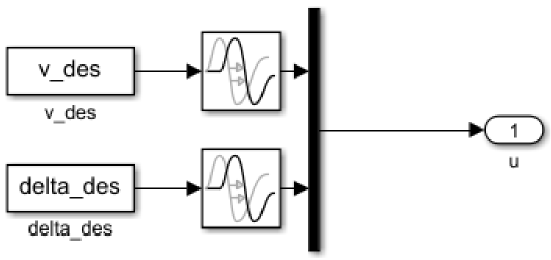


Figure I.8: Subsubsystem u of the first observer model

J

M-file: *Mahalanobis.m*

```
function [d,d_med] = Mahalanobis(noise_ibeo_loc,tr,z)
%% Mahalanobis distance
%figure
%for i = 1:length(xcov_ibeo_loc)
%    ellipse(xcov_ibeo_loc(i),ycov_ibeo_loc(i))
%end

%meancovx = mean(xcov_ibeo_loc);
%meancovy = mean(ycov_ibeo_loc);
%meancov = [meancovx; meancovy];

% Load in covariance data
xcovl = noise_ibeo_loc.xx_lcov;
ycovl = noise_ibeo_loc.yy_lcov;
xycovl = noise_ibeo_loc.xy_lcov;        % this is actually the correlation!
xcovl_x4 = vectorfourtimes(xcovl);
ycovl_x4 = vectorfourtimes(ycovl);
xycovl_x4 = vectorfourtimes(xycovl);    % this is actually the correlation!

theta = tr(:,3);         % retrieve orientation of the vehicle for rotation

cov_xx_yy = [];
cor = [];
for i = 1:length(xcovl_x4)
    [cov_xx_yy] = [cov_xx_yy, {([xcovl_x4(i), 0;
                    0, ycovl_x4(i)])}];
    cor = [cor, {corrcov(cov_xx_yy{i})}];   % correlation based on cov_xx,yy
end              % turns out to be diagonal matrix with ones on the diagonal

for i = 1:length(cor)
    cor{i}(2,1) = xycovl_x4(i); % fill correlation matrix with xy correlation
    cor{i}(1,2) = xycovl_x4(i); % fill correlation matrix with xy correlation
end

ExpCovariance = [];              % initialize covariance matrix
for i = 1:length(cor)
    ExpSigma = [xcovl_x4(i)  ycovl_x4(i)];
    ExpCorrC = cor{i};
    ExpCovariance = [ExpCovariance, {corr2cov(ExpSigma, ExpCorrC)}];
    cova = corr2cov(ExpSigma, ExpCorrC);
    xx(i) = cova(1,1);
    xy(i) = cova(1,2);
    yy(i) = cova(2,2);
end

% Compute Mahalanobis distance d by: sqrt((R*z)'*C^(-1)*R*z) R: rotation
% matrix, z: sample (r'-lambda*(r'(-))), C: covariance matrix
for i = 1:length(z)
    d(i) = sqrt(([cos(theta(i)), -sin(theta(i));
                  sin(theta(i)), cos(theta(i))]*[z(:,i)])'*2*inv(ExpCovariance{i})*
                 [cos(theta(i)), -sin(theta(i));
                  sin(theta(i)), cos(theta(i))]*[z(:,i)]);
end

xx_med = median(xx);             % prepare for calculating median later
xy_med = median(xy);             % prepare for calculating median later
yy_med = median(yy);             % prepare for calculating median later
cov_med = [xx_med, xy_med;       % constant covariance matrix based on median values
           xy_med, yy_med];

for i = 1:length(z)
    d_med(i) = sqrt(([cos(theta(i)), -sin(theta(i));
                      sin(theta(i)), cos(theta(i))]*[z(:,i)])'*2*inv(cov_med)*...
                     [cos(theta(i)), -sin(theta(i));
                      sin(theta(i)), cos(theta(i))]*[z(:,i)]);
end

cov_min = covmin(xx,yy,xy); % calculate covariance matrix based on local minima cov

for i = 1:length(z)
    d_min(i) = sqrt(([cos(theta(i)), -sin(theta(i));
                      sin(theta(i)), cos(theta(i))]*[z(:,i)])'*2*inv(cov_min{i})*...
                     [cos(theta(i)), -sin(theta(i));
                      sin(theta(i)), cos(theta(i))]*[z(:,i)]);
end
end
```