Fault-Tolerant Environment Perception Systems for Autonomous Vehicles

Keja, S.C.

Award date:
2022

Department of Mechanical Engineering
Dynamics and Control group

Fault-Tolerant Environment Perception Systems for Autonomous Vehicles

Master Systems and Control
DC report number: DC 2022.046

Supervisors:
Prof. dr. ir. N. van de Wouw (TU/e)
Dr. ir. C.G. Murguia Rendon (TU/e)
Ir. E. van Nunen (Flanders Make)
Ir. P. De Clercq (Flanders Make)
Ir. S. Venkita (Flanders Make)

Committee:
Prof. dr. ir. N. van de Wouw (TU/e)
Dr. ir. C.G. Murguia Rendon (TU/e)
Ir. E. van Nunen (Flanders Make)
Dr. Ir. J. Elfring (TU/e)

Author:
S.C. Keja 0886654

June 26, 2022

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

24-05-2022

Name

Siebrand Keja

ID-number

0886654

Signature

Submit the signed declaration to the student administration of your department.

[i] See: https://www.tue.nl/en/our-university/about-the-university/organization/integrity/scientific-integrity/
The Netherlands Code of Conduct for Scientific Integrity, endorsed by 6 umbrella organizations, including the VSNU, can be found here also. More information about scientific integrity is published on the websites of TU/e and VSNU

Abstract

The topic of this graduation project is the fault detection for an environment perception system applied to a class of autonomous systems. The literature survey on this topic showed that in current research some topics regarding robust perception are not fully covered yet, for both general multiple target tracking systems as well as in the application domain of autonomous vehicles. Therefore, this project investigates how to develop a fault detection and isolation framework for the perception system of an autonomous tractor, by exploiting sensor redundancy in the perception system and making use of model-based approaches. A model for the perception system and environment is developed. This model is used to describe the nominal and degraded system behavior and assess performance. Then, fault detection approaches exploiting hardware-redundancy and models are developed and challenges are identified. Finally, a number of use cases are developed in order to asses the performance of the fault detection approaches.

Contents

List of symbols

General symbols

Symbol	Unit	Description
χ^2_d	-	Chi-squared distribution with d degrees of freedom
$c_{l_{j_i}}$	-	Weight value between measurement $_{j_i}$ from sensor i and track l
$\mathbb{E}\{X\}$	-	Expected value of the random variable X
c_a	-	Variable counting amount of associated measurements to a track in last N_a time steps
c_{conf}	-	Track confirmation indicator
c_m	-	Variable counting amount of lacking measurements for a track in last N_m time steps
d_{cd}	-	Threshold for confirmed to deleted track
d_{td}	-	Threshold for tentative to deleted track
d_{tc}	-	Threshold for tentative to confirmed track
d_{gate}	-	Gating threshold
$f_{d,i}$	-	Fault detection indicator, for sensor i
H_k	-	Number of tracks at time step k
j	-	Index over number of ground truth objects N_k
k	-	Time step index
K	-	Final time step
l	-	Index over the number of tracks H_k
$m_{k,i}$	-	Number of sensor readings for sensor i, time step k
M_k	-	Total number of sensor readings at time step k
N_a	-	Track management tuning parameter for track confirmation
N_k	-	Total number of ground truth objects in the world at time step k
N_m	-	Track management tuning parameter for track deletion
n	-	Number of dimensions in considered mathematical space
n_s	-	Number of sensors
O_l	-	Track l
$\mathcal{O}$	-	Set of all tracks
q	-	Dimension of the observed space
T_s	s	Sample period, i.e. time difference between consecutive samples
x_k^j	-	Coordinates of ground truth object j at time step k
X_k	-	Set of ground truth object states at time step k
v_k	-	Measurement noise, models measurement uncertainty at time step k
w_k	-	Process noise, models modelling uncertainty at time step k

Sensor model symbols

Symbol	Unit	Description
$\jmath$	-	Index over all sensor readings M_k at time step k
$\jmath_i$	-	Index over all sensor readings $m_{k,i}$ of sensor i
θ_i	rad	Orientation angle of sensor i
ϕ_i	rad	Opening angle of sensor i
c_{fov}	-	Field of view indicator
f_i^{loc}	-	Fault injection vector for sensor i
i	-	Index over number of sensors n_s
l_i	m	Range of sensor i
r_i	m	Mounting position of sensor i
R_i	m^2	Measurement covariance of sensor i
$\mathcal{R}$	-	Set of measurement covariances
Q	-	Process noise covariance
$z_{k,i}^{\jmath_i}$	-	Sensor reading $\jmath_i$ of sensor i, at time step k
$Z_{k,i}$	-	Set of sensor readings from sensor i, at time step k
$\mathcal{Z}_k$	-	Set of all sensor readings, at time step k

Kalman Filter symbols

Symbol	Unit	Description
ν_k	-	Measurement pre-fit residual at time step k
η_k	-	Measurement post-fit residual at time step k
K_k	-	Kalman gain at time step k
$P_{k\|k-1}^l$	-	Predicted estimate covariance
$P_{k\|k}^l$	-	Updated estimate covariance
$\mathcal{P}_{k\|k}$	-	Set of all state estimate covariances
$\hat{x}_{k\|k}^l$	-	State estimate for time step k
$\hat{X}_{k\|k}$	-	Set of all state estimates, for time step k

Hardware redundancy-based fault detection symbols

Symbol	Unit	Description
C_{k,i_c}	-	Cluster i_c at time step k
$\mathcal{C}_k$	-	Set of clusters at time step k
$d_i^{i'}$	-	Distance between measurement of sensor i and sensor i' according to some distance metric
D_i	-	Set of distance metrics for sensor i with sensor readings from other sensors
i_c	-	Index over number clusters n_c
n_c	-	Number of clusters, i.e. $\|\mathcal{C}\|$

Model-based fault detection symbols

Symbol	Unit	Description
$J(r_k)$	-	Residual evaluation function at time step k
r_k	-	Residual of a measured and reference signal at time step k
T_k	-	Fault detection threshold at time step k

Metrics

Symbol	Unit	Description
$d(.)$	-	Not specified metric
$d_{MD}(.)$	-	Mahalanobis distance
$d_E(.)$	-	Euclidean norm

List of abbreviations

General abbreviations

Abbreviation	Description
AV	Autonomous Vehicle
FD(IR)	Fault Detection (and Isolation and Reconfiguration)
FoV	Field of View
GNN	Global Nearest Neighbor
HRB	Hardware Redundancy-Based
KF	Kalman Filter
KPI	Key Performance Indicator
MB	Model-based
MD	Mahalanobis Distance
MTT	Multiple Target Tracking
MV	Majority Voter

Binary classifier abbreviations

Abbreviation	Description
FN	False Negative
FP	False Positive
PN	Predicted Negative
PP	Predicted Positive
TN	True Negative
TP	True Positive

Chapter 1

Introduction

The topic of this graduation project is the fault detection for an environment perception system applied to a class of autonomous systems. In order to solve the problem of detecting faults in the perception system, a model for the perception system and environment is developed. This model is used to describe the nominal and degraded system behavior and assess performance. This chapter is structured as follows. Section 1.1 presents the background and motivation for this research project. It gives a high-level overview on the research topics for autonomous vehicles, outlines the current state of practice, and concludes with a high-level problem statement. Section 1.2 states functionality assumptions and constraints imposed to carry out the research in this thesis. A number of definitions are given to make the remaining of the thesis more transparent. Next, Section 1.3 discusses relevant literature with respect to the high-level problem statement in the scope of this project. It concludes identifying the gap in current knowledge, relevant to the considered application. Section 1.4 discusses the research questions and project goals, based on the motivation and the literature survey. Finally, Section 1.5 gives the outline of this thesis.

1.1 Background

Autonomous driving is a promising technology, which had led to significant research on autonomous vehicles (AVs) over the past decades. Due to advances in Global Positioning Systems (GPS), LIght Detection And Ranging (LIDAR), RAdio Detection And Ranging (RADAR), computing power and other technologies, the level of driving automation evolved towards the current state; self-driving vehicles are used operationally in warehouses and self-driving functionalities are integrated with consumer vehicles [1], [2]. General benefits of AVs include increasing road traffic safety, travel time reduction and fuel efficiency. A thorough study about the advantages of AVs is discussed in [3].

A particular type of AVs are autonomous vehicles with a working function, also referred to as work-drive vehicles. These type of AVs are common in agriculture, logistics and the manufacturing industry. An example in the agricultural environment is a tractor with an implement, e.g., a front loader, mowing arm, sprayer, etc. Furthermore, automated guided vehicles with a robot arm used for drilling, picking or visual inspection can be seen in logistics and manufacturing industry. Figure 1.1 shows the AV that serves as a motivation for this project. Advantages of AVs in agriculture are reduction in expenditure on employment, less exposure towards health hazards caused by the spraying chemicals and a higher precision of the work performed by the autonomous functionalities.

Figure 1.1: Considered platform in typical application environment.

Compared with general AVs, an agricultural AV faces the same challenges as it operates in an environment where humans and vehicles are present. An additional challenge compared with regular AVs is that the autonomous tractor needs to interact with the environment. One of the core functionalities needed in autonomous vehicles is the capability of perceiving the environment. Based on the environment perception, it is possible to navigate and perform tasks. Perceiving the environment accurately is thus important for safe operation, in the context of collision avoidance. This is important with regard to humans (to prevent casualties), as well as for the platform itself (to prevent damage). Furthermore, accurate perception of the environment is needed to perform tasks and interact with objects in the surroundings.

The perception system is a part of the functional system architecture for an AV. The perception system performs the task of collecting information and extracting relevant knowledge of the environment. It is needed to detect the environment and keep track of earlier detections. Figure 1.2 shows a functional system architecture of a general AV [4], which enables the vehicle to operate safely and efficiently through the environment. Based on the position of the host platform and perceived objects, stored in the environmental model, missions can be planned. Mission planning concerns the high-level task of an AV, such as a certain operation in the field (e.g., spraying the farmland) or remaining idle. The task and motion planner involve low-level operation of an AV based on the current mission, such as navigating between waypoints, or picking up pallets. The vehicle control block concerns the control signals to the actuators.

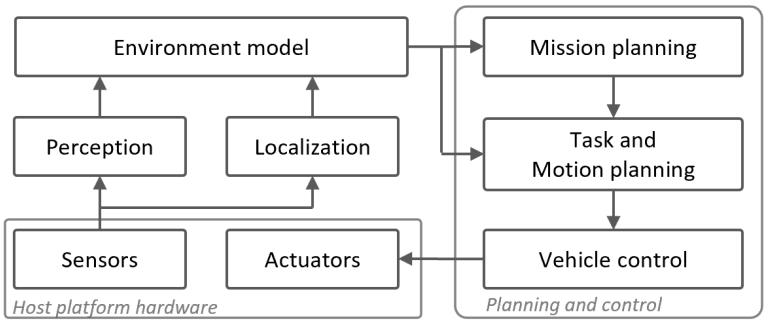

Figure 1.2: General functional system architecture for an AV, based on [4].

Another important part of an AV are the sensors, providing information about the surroundings (perception) and the position of the AV with regard to the environment (localization). The platform needs to keep track of several objects that exist in the world. Each object can be defined with a set of variables, such as position and object class, that can be measured with help of the perception sensors (for the considered cameras and lidars) mounted to the platform. With help of cameras the object class can be identified, but cameras cannot estimate the depth coordinates accurately. Lidars are able to measure depth, but can not detect object classes, making the two sensor types supplementary. Besides the measurements from the current time step, it is possible to use the sensor readings from previous time steps to keep track of the objects in the environment. These values are stored in the environment (or world) model. A detailed descrip-

tion of the host platform perception system part from the functional architecture can be found in Appendix A.2.

The perception system is critical for safe and efficient operation, therefore it is important that perception system is monitored for showing nominal or faulty behavior. The property indicating that a system is able to continue operation despite being in a degraded state is referred to as 'robustness'. Fault detection and fault-tolerance [5] are methods that can be used to create a robust system. In this project a robust perception system is developed. In the following, some basic terminology used in the field of fault detection and fault tolerance is discussed.

A fault is an unexpected deviation of at least one characteristic property (feature) of the system from the acceptable, usual or standard condition [5]. The propagation of a fault can result in a failure or malfunction of the system. A malfunction is an intermittent irregularity in the fulfillment of a system's desired function. A failure is a permanent interruption of a system's ability to perform a required function under specified operating conditions [6]. Fault tolerance is a system property indicating that the system can continue operation, although it can be in a degraded state due to faults.

One could think of the following typical cases, indicating the relevance of robust environment perception. False object detection could lead to erroneously belief in the presence of a human or other object, causing the system to go to the safe state, or to start replanning unnecessarily. On the other hand, missed detections could lead to dangerous situations (such as collisions) or inefficient operation, when the target object that needs to be acted upon is not detected. Both of these situations can be classified as failures.

Now that the most important aspects regarding the problem have been introduced, the background regarding this project is presented. This graduation project is conducted in cooperation with Flanders Make and Eindhoven University of Technology (TU/e). Flanders Make is a strategic research centre for the manufacturing industry, their aim is to enable the technological development of autonomous vehicles, machines and factories of the future. Autonomous vehicles with a work-drive functionality for agricultural and logistic applications is one of the topics currently being researched by the Belgium research institute Flanders Make. This graduation project is part of the work-drive research at Flanders Make. Challenges that are studied in this research concern modularity with respect to the perception architectures in automated vehicles with a work function [7].

By having the robust perception capabilities on the processed perception sensor output (more details on this are given in the next section), instead of the raw sensor data, the robust perception algorithms are independent of specific used sensors and therefore could contribute to the modular perception architectures. Therefore, the high-level problem statement is as follows: *to develop a framework that is able to distinguish between nominal and faulty object detections on an object fusion level for an autonomous vehicle that operates in the presence of faults in software and hardware of sensors.*

1.2 Preliminaries

In this section, preliminaries are discussed to clarify specific terms used throughout the document. Also, constraints and assumptions are given.

Figure 1.3 shows a smart sensor schematically. The sensor together with the object detection algorithm, can be observed as a 'black box object detector', or a so-called 'smart-sensor'.

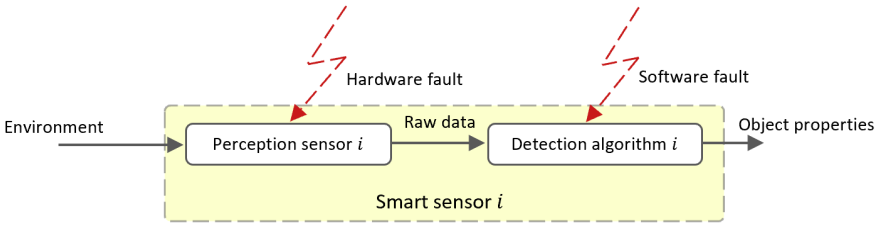

Figure 1.3: A schematic depiction of a smart sensor.

As the figure shows, faults can originate from the hardware or software. For this project, the origin of the fault is not relevant. The robust perception framework receives one or multiple sensor readings at time step k (output from smart sensor(s)), and needs to detect whether this sensor reading is in a degraded state. A set of smart sensors is attached to the considered work-drive AV. When referring to this considered AV, the term *host platform* is utilized, in order to prevent confusion when referring to other or general AVs. This expression is chosen to make explicit that the considered platform hosts the perception system. A complete overview of assumptions is documented in Appendix A.1.

Two main fault categories are considered in this project: localization and existence faults. Localization faults are modified observations of a given target. During nominal operation, sensor readings will adhere to a sensor noise according to an assumed distribution. When the noise increases or a bias is added, the sensor will output faulty measurements. Existence faults are divided into two subcategories, false object detections and missed detections. False object detections include clutter and decoys. A decoy can be introduced by an adversary as an attack to the system. Clutter is defined as the set of detections by a sensor not originating from true objects, where the number of detections and the coordinates of the detection can be sampled from a probability distribution. A missed detection fault type means that an object is not detected by a sensor, despite it is expected to be detected, for example caused by target suppression by an adversary. Another cause of missed detections could be due to sensing limitations, simply because the object is not identifiable yet (the considered object might be too far away, or unidentifiable caused by bad weather conditions) from the raw sensor data. Further possible causes for failing to detect an object are a limited Field of View (FoV) or an occlusion. Figure 1.4 shows the considered fault types and some examples in a diagram.

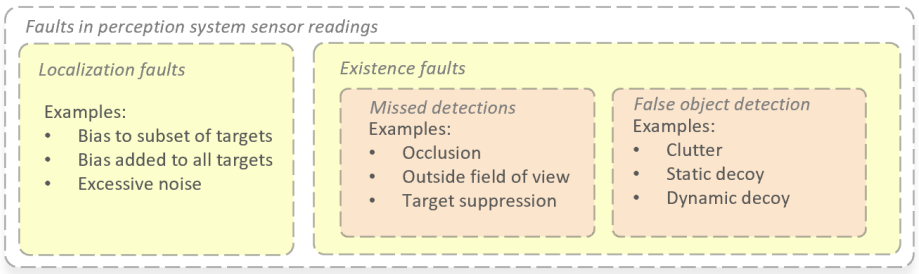

Figure 1.4: Fault types considered in this project.

In the literature on the topic of fault detection and fault tolerance, many terms can be encountered due to terminology not being unique, since these topics are researched in many application domains. This document maintains the following definitions, mostly based on definitions used in [5] and [8].

Analytical redundancy: making use of multiple ways to determine a variable, where at least one way uses a mathematical process model in analytical form.

Fault detection: fault detection is the capability to determine if there is a fault in the system.

Fault diagnosis: consists in determining the type, size and location of the fault, as well as its time of

detection. In this project, fault diagnosis is used in the context of determining if a detected fault is an existence or localisation fault.

Fault isolation: the ability to isolate the specific faulty measurement from the set of measurements.

Fault reconfiguration: the ability to take action after a fault has been detected and isolated in order to return the system to a stable state.

Nominal: expected value from the process

Residual: a fault indicator, based on deviation between measurements and model predictions.

The following definitions are related to multiple object tracking. See [9] for an extensive literature survey on multiple object tracking.

Object detection: detecting an object of interest in a single environment scan at a time step.

State: a set of variables at a point in time describing the dynamic behavior of interest for a system.

Object tracking: associating object detections to the object of interest across several environment scans over time.

1.3 Literature review

Based on the high-level problem statement, motivation and scope given in the previous sections, a literature study is performed. The goal of the literature survey is to asses general fault detection methods, discuss object tracking methods and in particular evaluate how object tracking and perception systems (applied but not limited to autonomous vehicles) cope with faults.

From a high-level view, there are roughly three building blocks to make a fault-tolerant system: fault detection, fault isolation and fault reconfiguration. There are basically two mechanisms to detect faults: one is to compare sensor readings mutually from the same time step, making use of hardware redundancy. The second approach is to compare model predictions with sensor readings (model-based fault detection). The main purpose of fault detection and isolation is to effectively detect faults and accurately isolate them from a failed component in the shortest time possible. This capability leads to reduction in diagnostic time or downtime in general and, therefore, increased system availability. In order for a system to reconfigure (automatically) from a failure, there are in general three ways to realize this capability [10], defined by the following:

- discard the faulty modules,
- switch to redundant modules,
- continue operation with less performance.

Further, attention is given to multiple target tracking in the literature review, to asses how several approaches compare. The structure of the literature review is as follows. Section 1.3.1 discusses multiple target tracking approaches. Section 1.3.2 documents common hardware redundancy based approaches for fault detection. Section 1.3.3 gives a summary of literature in the field of fault detection and fault tolerance for perception systems. General achievements, advantages and disadvantages are listed per research. Finally, in Section 1.3.4 the gap in current research relevant for this project is identified.

1.3.1 Multiple target tracking

Multiple Target Tracking (MTT), also referred to as Multiple Object Tracking (MOT), concerns estimating the number of objects in the world (in the proximity of the application at hand) and the attributes of those objects. Attributes can be object type, orientation or position. In this project, the attribute is limited to position in a two-dimensional plane. Several challenges exist concerning MTT, such as the probability of detecting an object and the challenge of assigning sensor readings to (new) tracks, known as data association. An object might not be present in the field of view of the sensor. Also, a problem arises when target are densely distributed, this makes the assignment of detections to tracks (data association) ambiguous. Further,

false object detections increase complexity of the data association. To cope with those challenges, several approaches towards solving the problem exist, these will be discussed in a later part of this section, first a general multiple target tracking framework is introduced.

Figure 1.5 shows a diagram with functional elements as a general framework for most MTT approaches. The MTT scheme is now referred to as the 'system'. The system considers whether it needs to update tracks based on new sensor readings. It is assumed that the tracks already have been initialized. In the filtering step, states are predicted from the current time step state estimations to the next time step with help of a filter (such as a particle or Kalman filter). Based on the predicted tracks, validation gates per track are computed. Sensor readings satisfying the gate condition are considered as sensor reading candidates to be assigned to the track. Defining a gate area can be improved with help of knowledge on the measurement noise and state prediction covariance [11]. During data association, sensor readings are assigned to tracks, based on the used tracking algorithm. During the track management step, tracks can be initialized, confirmed or deleted. Unassigned sensor readings can be used to initialize a new track. Tentative tracks are tracks that have been initialized recently, and can be confirmed if enough evidence is provided that this concerns a real track. Furthermore, tracks that remain without or with a few detections for set period, can be deleted.

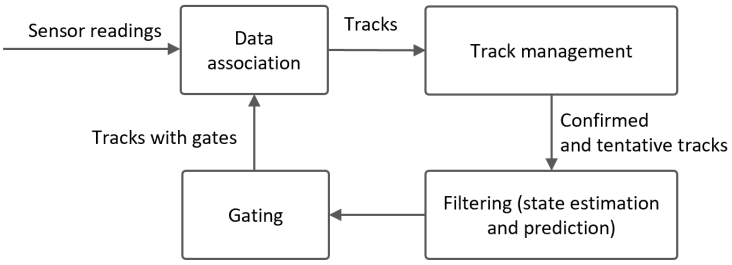

Figure 1.5: General target tracking scheme.

The three most commonly used MTT algorithms are the Multiple Hypothesis Tracking [12], Global Nearest Neighbour (GNN) [13] and Joint Probabilistic Data Association (JPDA) [14] apporaches. The GNN approach assigns the observation to a track that has the smallest mutual distance. The JPDA algorithm incorporates all detections found inside the validation gate. All the detections inside the gate contribute as a weighed average. The MHT is a target tracking algorithm that keeps at each time step multiple hypotheses about the past and current data association uncertainties. At every time step, all possible hypotheses are generated and over time part of them will be pruned, else, the number of hypotheses to maintain will become too large to keep track off. Hypotheses are pruned by combining similar hypotheses, or by pruning low probability hypotheses. GNN and JPDA approaches tend to perform very bad compared to MHT in environments with high clutter to low signal value. A clear disadvantage of MHT is the high computational load. An in-depth study of several MTT algorithms is presented in [15]. Extensions and improvements of the previously discussed tracking methods are also discussed in this work.

1.3.2 Hardware redundancy-based fault detection

The considered host platform possesses multiple perception sensors. Some sensors have a overlapping field of view. The sensors with overlapping field of views will detect on their intersecting FoVs the same objects during nominal operation. If one sensor gets corrupted by a sufficiently large ('detectable') fault, it is possible to identify the faulty sensor by comparing the sensor readings mutually. The common mechanism to implement this, works as follows. First, the sensor readings are mutually compared with help of a distance metric. Then, sensor readings are declared faulty or nominal based on the distance metric values. To declare sensor readings faulty or nominal, there should be a sufficient amount of healthy sensors. The amount needed depends on the chosen method. Finally, with help of the nominal sensor readings a final output is computed.

The four most commonly used voter techniques in fault tolerant systems [16], are the majority voter,

weighted averaging technique, median voter and the plurality voter. The majority, median and plurality voters compare the received values and will output either the majority median or plurality as the final value. The weighed average voter combines all measurements and produces a new distinct output. The choice of weights can be defined based on the prior knowledge of the sensor reliability [16].

A clear disadvantage of the described voter techniques is that the nominal sensor readings are deterministic. In most systems, sensor readings are subject to noise, and therefore cannot be compared directly. Comparing readings mutually would lead to declaring nominal measurements as faults, since it is not possible to measure the variable of interest without uncertainty in a real world setting. This is also applicable to the perception sensors of the host platform. To cope with this problem, sensor measurements are compared mutually with help of a distance metric [17], if the distance is smaller than a threshold, the measurements are considered equivalent. With help of this attribute, it is possible to use the preceding voter techniques. The former discussed voters were applied to object coordinate measurements. Additionally, it is possible to use a voter method to distinguish between true and false object detections. In [18] a voter algorithm for an automated vehicle was implemented to distinguish between false and true object detections. This is a k out of n voter for confirming detections, with n the number of sensors and k a tunable algorithm parameter. When setting k, this forms a special instance of the plurality or majority voter.

1.3.3 Model-based fault detection

Grübmuller et al. developed a fault-tolerant environment perception architecture that is able to detect software and hardware faults [19]. To this end, the state of the hardware and software are monitored separately. 'Observers' are defined to estimate states (e.g., position and velocity) of detected objects over time. This means that every detected object has a separate observer. Hardware faults are detected with help of a χ^2-fault detector by comparing observer values with sensor readings. To detect software faults, redundant observers are initialized, and their outputs are compared mutually. Brumback et al. [20] were among the first to use fault detection with help of a χ^2 distribution. Faults considered in the research are clutter and faults in state estimation due to modelling errors. Disadvantages are that only fault detection is performed on a single object detection per sensor. Further, no explicit use is made of sensing redundancy, but only observer (i.e., software) redundancy. Furthermore, combined clutter and localization fault, which gives rise to problems with gating for data association, is not addressed in this research.

In a series of publications, [21], [22], [23], Realpte et al. published research about a fault-tolerant perception system. In order to detect the faults, a Support Vector Machine (SVM) algorithm is trained. Drawbacks are that the considered fault-tolerant framework uses raw sensor data (low-level sensor fusion). Using raw sensor data is sensor specific and obstructs the possibility of modular perception architectures. Another drawback is the assumption made that a reference sensor can be used. This reference sensor is assumed to always be in a nominal (i.e., not faulty) state.

The research in [24] provides a fault detection approach that makes use of a world model, for an AV perception system. Faults are detected by comparing the bounding box smart sensor outputs with a reference (predicted by the model). Also object classes from sensor readings are compared with the reference with help of a look-up table listing the class correlations. When the correlation between the estimated (detection) class and the predicted (reference) class is too low, a fault indicator can be triggered. Disadvantages of the approaches in this research is that no use is made of sensor noise knowledge or sensing redundancy. Both could have great potential to improve fault detection.

In [25], Fault detection in (distributed) multiple target tracking system is studied. With help of the OPtimal Sub-pattern Assignment (OSPA) metric [26], various fault types can be detected, such as dynamic decoys, clutter and suppressed targets (existence faults) as well as bias and excessive noise (localization faults). However, it is not possible to distinguish between the fault types (Fault Diagnosis).

1.3.4 Discussion

The literature survey shows that in current research some topics regarding robust perception are not fully covered yet, for both general multiple target tracking systems as in [25] as well as in the application domain

of autonomous vehicles [19]. The topics that remain open are as follows:

1. the possibility to exploit hardware redundancy to detect faults in an environment perception system,
2. FDI in an environment with both false object detections and bias at the same time instant,
3. distinguish between fault types, clutter and localization fault.

1.4 Problem formulation

Based on the research gap from the previous section and the high-level problem statement introduced earlier in the introduction, the following problem statement is defined:

How to develop a fault detection and isolation framework for the perception system of an autonomous tractor, by exploiting sensor redundancy in the perception system?

The problem statement can be subdivided into the following challenges.

1 How to detect faults in the perception system?
2 How to distinguish between localization and existence faults?
3 How to perform the late data fusion from multiple sensors such that objects can be tracked accurately?

From the defined challenges, the following research goals can be formulated.
Develop a fault detection framework for a work-drive system of Flanders Make that is able to:
1 detect and isolate faulty data in the environment perception system,
2 distinguish between localization and existence faults,
3 fuse data from multiple sensors in order to track and localize objects.
4 Implement the framework in a simulation environment and evaluate its performance.

The final chapter, Section 6.1 discusses how each research goal of this project is fulfilled. In order to develop a framework that allows for robust perception, a model of the system will be developed. Then, by defining degraded system models, appropriate solutions are provided to detect faulty measurements. The following section explains the thesis outline in more detail.

1.5 Thesis outline

The remainder of this thesis is structured as follows. Chapter 2 presents how the system and environment are modelled. The chapter describes how the perception system is modelled in order to asses the fault detection performance.

Chapters 3 and 4 explain the development and the motivation of the fault detection algorithms for the perception system. Hardware-redundancy and model-based based fault detection methods are considered, respectively.

Chapter 5 presents the design of the use cases and the used metrics to evaluate the performance. Thereafter, the performance of the fault detection schemes in terms of the discussed metrics are given, and assessed with help of simulations.

Finally, Chapter 6 concludes with a discussion on the obtained results and a conclusion. Furthermore, possible research paths for useful and promising future work are given.

Chapter 2

System modelling

The goal of this project is to develop a method that monitors the state of the perception system and detects if this system is in a degraded state, i.e., faults are present. In the considered application, the perception system is part of an autonomous vehicle (host platform), that navigates in an agricultural environment. In order to analyze the performance of the developed methods, models are made of the perception system and objects in the environment. Figure 2.1 shows an adapted version of the multiple target tracking architecture from Figure 1.5, now including a fault detection module. The parts with a yellow background need to be modeled to design the fault detection approaches. The fault detection block itself is discussed in Chapters 3 and 4. This chapter describes how the perception system and objects in the environment are modelled in order to asses the fault detection performance.

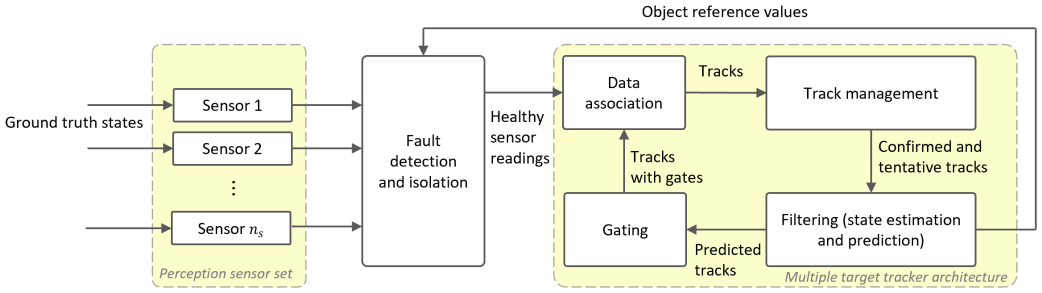

Figure 2.1: Multiple target tracking architecture modified for fault detection.

Figure 2.2 shows the model of the multiple target tracker and perception sensors in a high-level overview. Compared to Figure 2.1, the 'Filtering' block is now split into an update step (also referred to as state estimation) and prediction step block. Furthermore, the fault detection and isolation block is discarded, since the diagram only shows the modelled components.

The model made of the perception system and objects in the environment is simplified with regard to the real system; a detailed description of the application perception system can be found in Appendix A.2. The data processed by the multiple target tracker function blocks, is the set of all tracks $\mathcal{O} = \{O_1, O_2, \ldots, O_{H_k}\}$. A track O_l is defined as a list containing several variables relevant for an object trajectory. It contains the associated sensor readings and the estimated states up till the current time step. With this information available, it is possible to update state estimates by taking into account the associated measurement, stored in the considered object list O_l. In Section 2.1.1 an explicit definition of the track variable is given. Section 2.3 explains how the Global Nearest Neighbour (GNN) algorithm works.

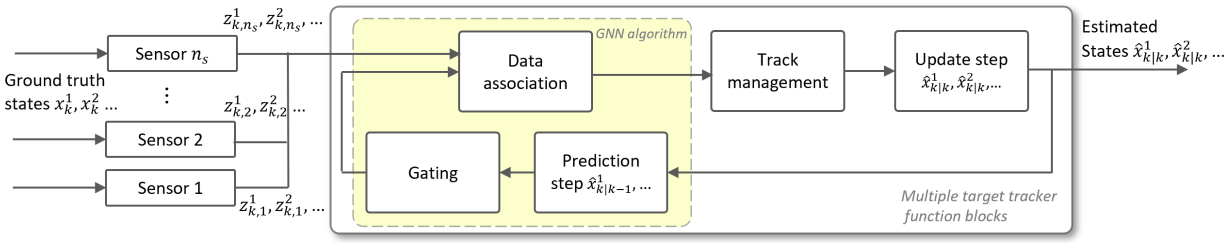

Figure 2.2: Overview of involved components for multiple target tracking.

Based on the multiple target tracking architecture, the chapter is structured as follows. Section 2.1 concerns the state space representation used to model object states, measurements and state estimates x_k^j, $z_{k,i}^{j_i}$ and $\hat{x}_{k|k}^l$ respectively. The section introduces notation and explains the tracking structure. Section 2.2 discusses Kalman filters and their capability to filter the sensor readings. In the multiple target tracking overview figure, the prediction and update step blocks are part of the Kalman filter. Section 2.3 explains the methodology of the data association module. Section 2.4 explains the track management. Finally, Section 2.5 explains how faults are introduced in the nominal sensor model.

2.1 State space representation

This section starts with defining state space models, which are commonly used to describe dynamic systems. Based on this equations, models are derived for the perception sensors and objects in the environment.

2.1.1 State space model

A state space model is a mathematical model of a physical system described by input, state and output variables. The state space representation is used to model objects in the environment and the host platform. The external inputs could be the input velocity, if a set of state space equations models a vehicle. A continuous-time model with external input u can be described as follows:

$$\dot{x}(t) = f(x, u), \tag{2.1}$$
$$z(t) = g(x, u), \tag{2.2}$$

with x the system state, z the observed state (output) and $\dot{x}$ the time derivative of the state. The non-linear system equations described in discrete time, are:

$$x_{k+1} = h(x_k, u), \tag{2.3}$$
$$z_k = l(x_k, u). \tag{2.4}$$

If the functions $f(x_k, u)$ and $g(x_k, u)$ are linear, the system equations can be written in state space form:

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{2.5}$$
$$z(t) = Cx(t) + Du(t), \tag{2.6}$$

with A, B, C and D the state matrix, the input (or control) matrix, the output (or observation) matrix and the feedthrough (or feedforward) matrix, respectively. If Equations (2.1) or (2.2) are non-linear, they can be linearized around an operating point and written in the linear state space form [27], given by Equations (2.5), (2.6). The continuous time state space equations can be described as discrete time equations:

$$x_{k+1} = A_d x_k + B_d u_k, \tag{2.7}$$
$$z_k = C_d x_k + D_d u_k, \tag{2.8}$$

with $k \in \mathbb{N}$ the sample time step. The discretized state space equations can be derived using the analytical solution of Equation (2.5) assuming zero-order hold for the input $u(t)$. With some rewriting this results

in:

$$x((k+1)T_s) = e^{AT_s}x(kT_s) + \int_{kT_s}^{(k+1)T_s} e^{A(k+1)T-\tau}d\tau Bu(kT_s),$$

with $\tau \in [kT_s, (k+1)T_s)$ and T_s the sample period. By defining $\lambda = (k+1)T_s - \tau$ we obtain:

$$x((k+1)T_s) = e^{AT_s}x(kT_s) + \int_{0}^{T_s} e^{A\lambda}d\lambda Bu(kT_s),$$

with $\lambda \in [0, T_s)$. Note that the equation is now written in the form of Equations (2.7) and (2.8). Therefore discretized state space matrices can be described in terms of the continuous time matrices and the sample period:

$$A_d = e^{AT_s}, \tag{2.9}$$

$$B_d = \int_{0}^{T_s} e^{A\lambda}d\lambda B. \tag{2.10}$$

A more elaborate derivation of these equations can be found in [28]. In the rest of this document, the subscripts for the discrete time state space matrices A_d, B_d, C_d, D_d will be omitted from notation for sake of simplicity. If the real system dynamics differ too much from the state space description, there is a modelling mismatch. To take this mismatch into consideration, a process noise term w_k is introduced in the equations. Furthermore, to allow for the measurement noise, the term v_k is introduced in the observation equations. The set of adapted discrete-time state space equations is now given by:

$$x_{k+1} = Ax_k + Bu_{k+1} + w_{k+1}, \tag{2.11}$$

$$z_k = Cx_k + Du_k + v_k. \tag{2.12}$$

with $A \in \mathbb{R}^{n\times n}$ state transition matrix, $B \in \mathbb{R}^{n\times p}$ control-input matrix and $w_k \in \mathbb{R}^n$ the process noise vector, which is described by a zero mean multivariate normal distribution with covariance $Q \in \mathbb{R}^{n\times n}$: $w_k \sim \mathcal{N}(0, Q)$. Further, $C \in \mathbb{R}^{q\times n}$ is the observation matrix mapping the ground truth space to the observed space and $v_k \in \mathbb{R}^q$ the measurement noise vector, described by a zero mean multivariate normal distribution with covariance matrix $R \in \mathbb{R}^{q\times q}$: $v_k \sim \mathcal{N}(0, R)$. This is a probabilistic state space model [29]. Furthermore, the following notation is introduced. In order to consider for multiple objects in the environment, Equation (2.11) is adapted as follows:

$$x^j_{k+1} = A_j x^j_k + B_j u^j_{k+1} + w^j_{k+1}, \tag{2.13}$$

with $j \in \{1, 2, \ldots, N_k\}$ the ground truth object index and N_k the amount of objects in the environment. The set of ground truth object states at time step k is defined as:

$$X_k = \{x^1_k, x^2_k, \ldots, x^{N_k}_k\}.$$

For a static object in two-dimensional space, Equation (2.13) is modified as follows:

$$\begin{bmatrix} x^j_{k+1,1} \\ x^j_{k+1,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x^j_{k,1} \\ x^j_{k,2} \end{bmatrix} + \begin{bmatrix} w^j_{k+1,1} \\ w^j_{k+1,2} \end{bmatrix},$$

describing a point object at a constant position over time. The control input term is dropped, since it is assumed the object is static. The state matrix A is now the identity matrix, because the two-dimensional coordinates remain constant over time. Assumed there is only one static object in the considered setting, Equation (2.12) extends to:

$$\begin{cases} z_{k,1} = C_1 x_k + v_{k,1}, \\ z_{k,2} = C_2 x_k + v_{k,2}, \\ \quad \vdots \\ z_{k,n_s} = C_{n_s} x_k + v_{k,n_s}. \end{cases} \tag{2.14}$$

Now, each sensor i can detect multiple objects, Equation (2.14) is adapted as follows for measurements $z_{i,k}^{\jmath_i}$, with each sensor detecting N_k objects:

$$z_{i,k}^{\jmath_i} = C_i x_k^j + v_{k,i}, \tag{2.15}$$

where $i \in \{1, 2, \ldots, n_s\}$ is the sensor index and $\jmath_i \in \{1, 2, \ldots, m_{k,i}\}$ is the detection index for sensor i at time step k, where $m_{k,i}$ is the number of detections by sensor i at time step k. If a sensor i has no limit in its field of view, then the amount of objects modeled in the environment is equal to the amount of detected objects by sensor i, and thus $m_{k,i} = N_k$, in nominal behavior. A set of sensor readings at time step k, from sensor i, is defined as:

$$Z_{k,i} = \{z_{k,i}^1, z_{k,i}^2, \ldots, z_{k,i}^{m_{k,i}}\}.$$

The set of all sensor readings at time step k, is defined as:

$$\mathcal{Z}_k = \{Z_{k,1}, Z_{k,2}, \ldots, Z_{k,n_s}\}.$$

The set of measurement noise covariances is defined as:

$$\mathcal{R} = \{R_1, R_2, \ldots, R_{n_s}\},$$

which are assumed to be constant over time, per sensor. Since now multiple objects in the environment are considered, multiple tracks can be initialized. The set of state, covariance and measurement predictions are defined as:

$$\hat{X}_{k|k-1} = \{\hat{x}_{k|k-1}^1, \hat{x}_{k|k-1}^2, \ldots, \hat{x}_{k|k-1}^{H_k}\},$$

$$\mathcal{P}_{k|k-1} = \{\hat{P}_{k|k-1}^1, \hat{P}_{k|k-1}^2, \ldots, \hat{P}_{k|k-1}^{H_k}\},$$

$$\hat{Z}_{k|k-1} = \{\hat{z}_{k|k-1}^1, \hat{z}_{k|k-1}^2, \ldots, \hat{z}_{k|k-1}^{H_k}\},$$

with H_k the number of tracks. Further, let $l \in \{0, 1, \ldots H_k\}$ be the index over the number of tracks.

In the introduction, 1.2 object tracking was defined as: *associating object detections to the object of interest across several environment scans over time.* In order to perform this task, a structure is defined that supports the object tracking task. This structure is called a 'track'. A track is defined as a structured list:

$$O_{l,k} = \{\hat{x}_{k|k}, P_{k|k}, \mathbf{z}, \mathbf{tr}_{\text{status}}\},$$

with $\hat{x}_{k|k}, P_{k|k}$ the state and covariance estimates, $\mathbf{z}$ a vector containing all the associated measurements. Further, $\mathbf{tr}_{\text{status}} = (c_{\text{conf}}, c_a, c_m)$ is a list containing information about the track management, Section 2.4 explains these variables in more detail. The data association and track management approaches are explained in Sections 2.3 and 2.4, respectively. The state and covariance estimation is explained in the following section. If a track is initialized, it is assumed that the underlying state space model is known. The set of all tracks is defined as $\mathcal{O} = \{O_1, O_2, \ldots, O_{H_k}\}$.

2.1.2 Sensor model

The perception sensors of the host platform are limited in their field of view. To take this effect into account, the field of view of a sensor is parametrized with an opening angle ϕ_i and a detection range l_i. The mounting position on the platform is parametrized with the position r_i and the orientation θ_i. This is schematically depicted for one perception sensor in Figure 2.3. Table 2.1 gives an overview of the sensor field of view parameters.

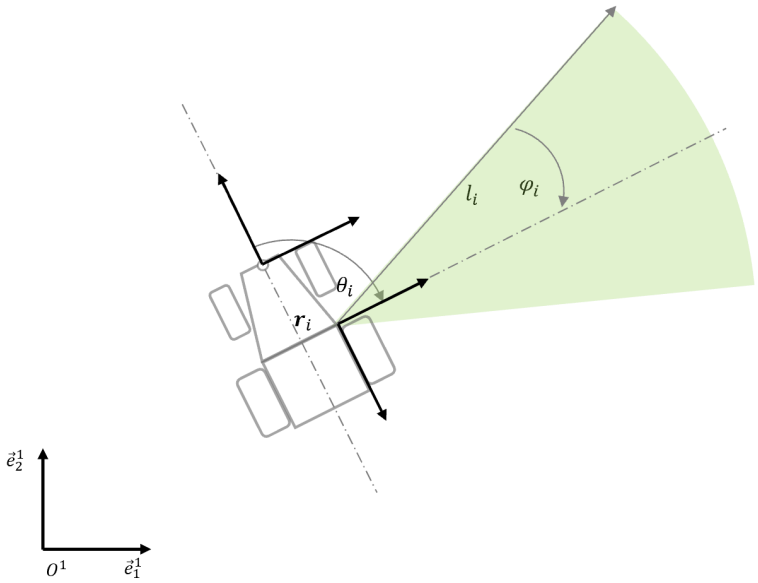

Figure 2.3: Field of view example for one sensor.

Table 2.1: Sensor field of view parameters.

Parameter	Unit	Description
θ_i	rad	Orientation angle of sensor i
ϕ_i	rad	Opening angle of sensor i
l_i	m	Range of sensor i
r_i	m	Mounting position of sensor i

In order to verify that an object is in the field of view of a sensor, the coordinates x_k^j need to be transformed to the local coordinate frame of the sensor, $\tilde{x}_k^j$. Then, with the following expression it can be verified if the object is in the field of view:

$$\begin{cases} c_{fov} = 1 & \text{if } ||\tilde{x}_k^j|| \leq l_i \ \ \& \ \ \angle|\tilde{x}_k^j| \leq \phi_i, \\ c_{fov} = 0 & \text{else.} \end{cases} \tag{2.16}$$

The scalar c_{fov} is used to determine if an object is detected by a sensor.

2.2 State Estimation

This section discusses state estimation for objects that are being tracked. An autonomous vehicle navigates through the environment and makes decisions based on the perceived surroundings. To this end, it is important that objects are tracked accurately. Outputs from the sensor can be used to track objects, but measurements are often subject to noise, resulting in inaccurate measurements. Also, it is often not possible to obtain all internal states by sensor measurements alone. As state estimator, the Kalman filter is used in this project. The Kalman filter combines measurements with model-based predictions to obtain a state estimation that is more accurate than state estimations obtained by only using measurements or model-based predictions. The filtering is optimal under condition of Gaussian noise and observable system equations. Appendix C.1 gives more details on the observability condition.

The Kalman filter has been used in widespread practical applications in many fields [30], such as orbit calculation, target tracking and navigation [31], in combination with GPS and environmental perception sensors. It also plays a role, in sensor data fusion, microeconomics [32], and in the field of digital image

processing and the current research fields like pattern recognition, image segmentation and image edge detection [33].

Section 2.2.1 introduces the general Kalman filter. Section 2.2.2 discusses how measurements are fused.

2.2.1 Linear Kalman Filter

The Kalman filter [34] is a recursive algorithm that combines a measurement with a prediction to compute an estimate of the current state. The Kalman filter procedure consists of two steps. It estimates the next state of a system ('prediction'), then it corrects the estimation with a measurement ('update'). The sequence of ground truth states $x_k, x_{k+1}, x_{k+2}, \ldots$ and the measurements $z_k, z_{k+1}, z_{k+2}, \ldots$ are modeled by a probabilistic state space model, as described by Equation (2.11) and (2.12).

The Kalman filter is an optimal linear estimator in the Minimum Mean-Square-Error (MMSE) sense [35] given a set of conditions, where the MMSE is defined as $\mathbb{E}\{||\hat{x}_k - x_k||^2\}$, with $\mathbb{E}$ the expectation, x_k the ground truth state and $\hat{x}_k$ the estimated state. The first assumption concerns the system model is linear, time-invariant and in state space form. The system equations after linearization described in the previous section adhere to these assumptions, see Equations (2.13) and (2.15). Further, the distributions of the state and measurements are Gaussian since the state space models are linear and the disturbances are assumed to be Gaussian:

$$z_k \sim \mathcal{N}(Cx_k, R). \tag{2.17}$$

This can also be interpreted as the process and measurement noise being zero mean Gaussian and mutually uncorrelated:

$$w_k \sim \mathcal{N}(0_{n\times n}, Q), \tag{2.18}$$
$$v_k \sim \mathcal{N}(0_{q\times q}, R). \tag{2.19}$$

In the following, $\hat{x}_{k|k-1}$ is an estimate of the state x_k at time step k based on the posterior state estimate at time step $k-1$. The Kalman filter models the system with a linear state space model, see previous section, discretized in the time domain:

$$x_k = Ax_{k-1} + Bu_k + w_k, \tag{2.20}$$
$$z_k = Cx_k + v_k, \tag{2.21}$$

The optimal state estimator for the previous described system, is the Kalman filter. The filter gives an estimate of the state $\hat{x}_{k|k}$ and its covariance matrix $P_{k|k}$. This result is obtained by a prediction and an update step. Prediction step equations are given by:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_k, \tag{2.22}$$
$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q. \tag{2.23}$$

Update step equations are as follows:

$$\nu_k = z_k - C\hat{x}_{k|k-1}, \tag{2.24}$$
$$K_k = P_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1}, \tag{2.25}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\nu_k, \tag{2.26}$$
$$P_{k|k} = (I - K_kC)P_{k|k-1}, \tag{2.27}$$
$$\eta_k = z_k - C\hat{x}_{k|k}, \tag{2.28}$$

with ν_k the measurement pre-fit residual, K_k the Kalman gain, $\hat{x}_{k|k}$ and $P_{k|k}$ updated state estimate and covariance and finally η_k the measurement post-fit residual. Figure 2.4 shows the discrete time Kalman filter visualized in a block diagram. In the prediction step, the predicted state and covariance estimations are computed with help of Equation (2.22) and (2.23). In the update step, the state and covariance estimate are computed, see Equations (2.24)-(2.28).

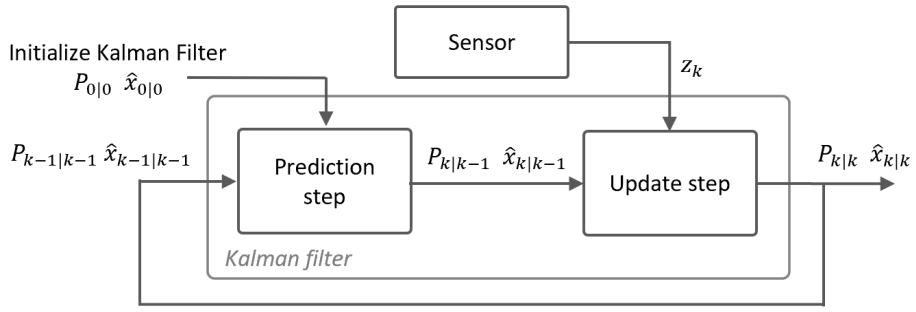

Figure 2.4: Block diagram of discrete time Kalman filter.

2.2.2 Sensor fusion

As an extension to the preceding, this section describes a single system that can be estimated by multiple sensors n_s. There are two alternative approaches for handling measurements from multiple sensors: the group-sensor method and sequential fusion methods [36]. The group-sensor method treats all separate sensor readings as if they originate from one sensor, and uses this augmented measurement to update a Kalman filter. The sequential approach updates the state estimate consecutively, with measurements from the same time step. In this section, the group-sensor method is considered. To this end, the system model equations are now adapted to accommodate for multiple sensors detecting a single object:

$$\begin{cases} x_k = Ax_{k-1} + Bu_k + w_k, \\ z_{k,1} = C_1 x_k + v_{k,1}, \\ z_{k,2} = C_2 x_k + v_{k,2}, \\ \quad \vdots \\ z_{k,n_s} = C_{n_s} x_k + v_{k,n_s}. \end{cases} \tag{2.29}$$

The set of sensor readings at time step k is defined as:

$$\mathcal{Z}_k = \{z_{k,1}, z_{k,2}, \ldots, z_{k,n_s}\}.$$

The prediction equations stay the same as in Equations (2.22)-(2.23) and Equations (2.24)-(2.28), respectively. As can be observed, the conventional Kalman filter algorithm equations can be used, but the following variables need to be changed to incorporate multiple sensor readings in the prediction and update equations:

$$z_k := \begin{bmatrix} z_{k,1} \\ z_{k,2} \\ \vdots \\ z_{k,n_s} \end{bmatrix}, \tag{2.30}$$

$$C := \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_{n_s} \end{bmatrix}, \tag{2.31}$$

$$v_k := \begin{bmatrix} v_{k,1} \\ v_{k,2} \\ \vdots \\ v_{k,n_s} \end{bmatrix}, \tag{2.32}$$

$$\begin{aligned} R &:= \mathbb{E}\{v_k v_k^T\} \\ &:= \mathbb{E}\{\left[v_1^T, v_2^T, \ldots, v_{n_s}^T\right]^T \left[v_1^T, v_2^T, \ldots, v_{n_s}\right]\} \\ &:= \text{blockdiag}\{R_1, R_2, \ldots, R_{n_s}\}. \end{aligned} \tag{2.33}$$

The above derivation holds when the sensor measurement noises are not cross-correlated at time step k, [36], i.e.,

$$\mathbb{E}\{v_{k,i}v_{k',i'}^T\} = R_i\delta_{ii'}\delta_{kk'},$$

where $\delta_{ii'}$, $\delta_{kk'}$ are the Kronecker delta functions.

2.3 Data association

This section presents the data association method. The purpose of data association is to assign measurements to the corresponding tracks. In an environment with multiple objects, multiple sensor readings are generated per scan (by an individual sensor). When tracks are far apart and only the true objects are detected, this task is not too difficult, but it is often the case that objects are in close proximity and due to the uncertainty in the sensor readings it might result into a conflicting situation. Additionally, some of those sensor readings may not originate from the real target, but are clutter or false alarms. Figure 2.5 shows an ambiguous situation, illustrating the relevance of a data association method. It is not clear beforehand what sensor readings belongs to what target, since there are multiple sensor readings and object state predictions. Furthermore, object tracks 1 and 2 share a sensor reading that seems to be belonging to both the tracks. To cope with these problems, a dedicated approach is needed. To this end, the Global Nearest Neighbour (GNN) [13] algorithm is considered. This approach consists of three consecutive steps: the state prediction, gating computation and finally the data association itself. Figure 2.2 gives an overview of the discussed data association steps in the overall object tracking architecture.

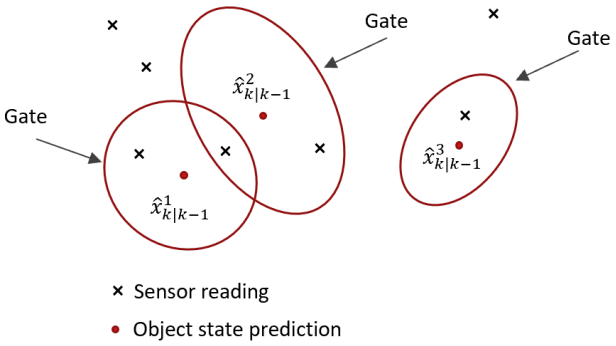

Figure 2.5: Example of a conflicting data association situation.

Now, the mathematics involved in the steps is presented. First, the set of states $\hat{X}_{k|k-1}$ and covariance estimates $\mathcal{P}_{k|k-1}$ of the tracks are predicted for the time step of sensor readings $\mathcal{Z}_k$. Prediction of the state and covariance estimates is treated in detail in Section 2.2. Then, based on a distance metric in combination with covariances of the measurement noise $\mathcal{R}$ and state estimate $\mathcal{P}_{k|k-1}$, the gates can be defined. The gate is defined by setting an appropriate threshold to use in combination with the distance metric. The gates serve the purpose of excluding sensor readings from data association that are highly unlikely.

On the basis of the used distance metric to compare predictions and measurements, a gating threshold can be chosen. Since the residual covariance matrix can be computed on the basis of the measurement covariance R_i and the predicted estimate covariance $\mathcal{P}_{k|k-1}$, the Mahalanobis distance metric is used [37]. The advantage of this metric is the chi-squared distribution property, and therefore the gating threshold can conveniently be defined from a table of the χ^2_d distribution, with d degrees of freedom. The gate threshold d_{gate} is chosen according to an allowable probability of a valid observation falling outside the gate.

The Mahalanobis distance [37] is defined as:

$$d_{MD}(z^{j_i}_{k,i}, \hat{z}^l_{k|k-1})^2 = (z^{j_i}_{k,i} - C_i\hat{x}^l_{k|k-1})\big(\Sigma_{il}\big)^{-1}(z^{j_i}_{k,i} - C_i\hat{x}^l_{k|k-1})^T,$$

with $\Sigma_{il} = C_iP^l_{k|k-1}C_i^T + R_i$ being the residual covariance matrix. If the distance metric exceeds the gate

threshold, the weight value is set to infinity, $c_{lj_i} := \infty$.

$$c_{lj_i} := \begin{cases} \infty & \text{if } d_{\text{MD}} \geq d_{\text{gate}}, \\ d_{MD}(z_{k,i}^{j_i}, \hat{z}_{k|k-1}^{l})^2 & \text{if } d_{\text{MD}} < d_{\text{gate}}. \end{cases} \tag{2.34}$$

Finally, the data association itself is performed. This algorithm tries to find for every tracked object, the closest sensor reading. First the weights computed with the distance metric are stored in a cost matrix. Each row in the cost matrix corresponds to a different track and each column to a different sensor reading. By setting weight values to infinity during the gating, it is assured that these measurement and track combinations will not be assigned. The cost matrix is defined as follows:

$$C_i^{\text{cost}} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1m_{k,i}} \\ c_{21} & c_{22} & \dots & c_{2m_{k,i}} \\ \vdots & \dots & \ddots & \vdots \\ c_{H_k1} & c_{H_k2} & \dots & c_{H_km_{k,i}} \end{bmatrix}, \tag{2.35}$$

the superscript 'cost' is introduced to avoid confusion with the output matrix C_i from the state equations. The GNN algorithm aims to find the combination of tracks and measurements that minimizes the following cost function:

$$J = \sum_{l=1}^{H_k} \sum_{j_i=1}^{m_{k,i}} e_{lj_i} c_{lj_i},$$

subject to the following constraints:

$$\begin{aligned} \sum_{j_i=1}^{m_{k,i}} e_{lj_i} &= 1, \forall l, \\ \sum_{l=1}^{H_k} e_{lj_i} &= 1, \forall j_i, \end{aligned} \tag{2.36}$$

with $e_{lj_i} \in \{0, 1\}$ and where $e_{lj_i} = 1$ when a measurement is associated to a track. These constraints assure that only one track is assigned to one measurement and vice versa. Unassigned measurements can be used to initialize new tracks, as discussed in the next section.

An algorithm to solve the linear assignment problem is the Hungarian method [38]. In this work, the function `Matchpairs` from Matlab is implemented to perform this task [39] [40]. Since multiple sensors are used, an object can be detected multiple times. In order to make sure every sensor readings is correctly assigned, the gating and data association steps need to be performed separately for every set of sensor readings $Z_{k,i}$. Algorithm 2.1 shows the data association mechanism in pseudo code, corresponding to the prediction, gating and data association steps of Figure 2.2. The explanation of the preceding is applied to a single sensor, but generalization to multiple sensors is trivial since the data association is performed per sensor individually.

Input: sensor readings $\mathcal{Z}_k$, Predicted state and covariance estimates $\hat{X}_{k|k-1}$, $\mathcal{P}_{k|k-1}$
Output: Set of tracks $\mathcal{O}$ (updated with measurements)
Algorithm parameter: $\mathcal{R}$

1. **for** all tracks
2. perform prediction step
3. **end**
4. **for** all sets of sensor readings
5. **for** all tracks
6. Construct cost matrix C_i
7. **end**
8. assign readings to track with linear assignment solver
9. store associated detections in corresponding tracks
10. **end**

Algorithm 2.1: Pseudo code of GNN algorithm.

2.4 Track management

Due to the limited field of view of the perception sensors, it is possible that objects enter and leave the perceived area. In order to manage this, a track management function is needed. The track management functionality serves three goals: 1) initializing new tracks, 2) verify whether initialized tracks can be confirmed and 3) delete confirmed tracks that remain undetected [41]. To this end, the following variables are defined per track: $c_{\text{conf}} \in \{0, 1\}$ to indicate whether a track is confirmed, with $c_{\text{conf}} = 0$ for tentative tracks and $c_{\text{conf}} = 1$ for a confirmed track. Distinction is made between tentative and confirmed tracks, since it is possible a track is initialized by false object detections. Therefore, the tentative track is subject to a more strict requirement to exist than the confirmed tracks, more details about this are given later in this section.

Further, $c_{\text{a}} \in \mathbb{N}$ and $c_{\text{m}} \in \mathbb{N}$ are defined as counter variables. The scalar $c_{\text{a}} \in \mathbb{N}$ counts how many times in the last N_a time steps at least one measurement is associated to the track and $c_{\text{m}} \in \mathbb{N}$ counts how many times in the last N_m time steps the track remained without associated measurements. Further, for the counter variables the thresholds d_{tc}, d_{td} and d_{cd} are defined for the transition from tentative to confirmed, from tentative to deletion and from confirmed to deletion, respectively. A confirmed track should be harder to delete than a tentative track, therefore $d_{td} < d_{cd}$. The transitions are schematically depicted in Figure 2.6. The figure shows the transitions and conditions for a track in a flowchart, with conditions for transiting to a new state indicated along the edges. Every vertex depicts one of the possible operations on a track, where `tent` means that a track is tentative, `conf` indicates a track O_l is confirmed and finally `delet` means a track is to be deleted from the set of tracks $\mathcal{O}$.

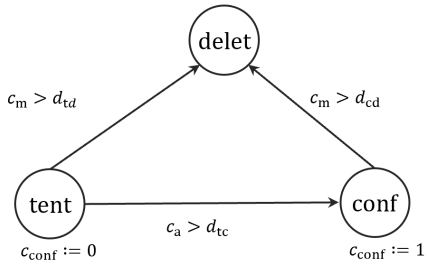

Figure 2.6: The transitions for a track depicted with a flowchart.

2.5 Fault modelling

Equation (2.21) introduced the nominal discrete-time state space model, for modelling perception sensor measurements. In order to model degraded sensor behavior, the nominal sensor models need to be adjusted to model faults. In literature, mainly two approaches exist towards fault modelling in (discrete-time) state space systems, these are additive fault and multiplicative fault representation. Therefore, to introduce faults in the system, the nominal sensor model introduced in Equation (2.15), is adapted. For a sensor with index i, the observation equations are now:

$$z_{k,i}^{\jmath_i} = C_i x_k^j + v_{k,i} + H_i f_{k,i}^{loc}, \tag{2.37}$$

with $z_k^{\jmath_i}$ is defined as: $z_{k,i}^{\jmath_i} \in \{\text{null}, \mathbb{R}^{q_i}\}$, where null is part of the domain to model the possibility that an object is not detected. Setting the value to the real value zero would not satisfy, since this would imply that an object in the origin of the coordinate frame is detected. The localization faults are modelled with $H_i \in \mathbb{R}^{q_i \times g_i}$ as the fault entry matrix and $f_{k,i}^{loc} \in \mathbb{R}^{g_i}$ the localization fault vector. By describing the fault injection with help of a fault vector and fault entry matrix, it is possible to map faults from the fault space $\mathbb{R}^{g_i}$, to the measurement space $\mathbb{R}^{q_i}$. By setting the elements appropriately in the fault vector and entry matrix, the fault is transformed to Cartesian coordinates and can be injected in the system equations. In a two-dimensional plane this can be parametrized with a direction ω and a magnitude F. The fault entry matrix and vector are then:

$$H_i = \begin{bmatrix} \cos\omega & 0 \\ 0 & \sin\omega \end{bmatrix} \quad f_{k,i}^{loc} = \begin{bmatrix} F \\ F \end{bmatrix}.$$

To describe the effect of clutter, $c_{k,i}$ is introduced, this means that Equation (2.37) can be augmented with clutter terms, per sensor i, modelled as:

$$z_{k,i}^{j_i} = c_{k,i}.$$

The false positive object detections $c_{k,i}$ are defined on the domain $c_{k,i} \in \{\text{null}, \mathbb{R}^{n_{fp} \cdot n}\}$, where n_{fp} is equal to the number of false positive object detections. In order to model the absence of false positive object detections, the domain of real numbers $\mathbb{R}^{n_{fp} \cdot n}$ should be augmented with the concept of detecting 'nothing', this is implemented by adding the term null to the domain.

The probability of detecting a false object at time step k can be modelled with a Bernoulli distribution. The Bernoulli distribution is the discrete probability distribution of a random variable taking value 1 with probability p and taking value 0 by probability $q = 1 - p$. The probability mass function of this distribution is:

$$\begin{cases} p & \text{if } \beta = 1, \\ q = 1 - p & \text{if } \beta = 0, \end{cases} \tag{2.38}$$

where $\beta = 0$ and $\beta = 1$ are associated with $c_{k,i} \in \{\text{null}\}$ and $c_{k,i} \in \{\mathbb{R}^{n_{fp} \cdot n}\}$, respectively.

The clutter behavior is captured by sampling from a multivariate normal distribution, therefore each seperate false positive object inside $c_{k,i}$ follows the distribution $\mathcal{N}(\mu_i, \Sigma_i)$ with parameters $\mu_i \in \mathbb{R}^{q_i}$, $\Sigma_i \in \mathbb{R}^{q_i \times q_i}$, which are sensor specific. The assumption is that false positive object detections are more likely to happen close to the sensor, therefore μ_i is equal to the sensor mounting position on the host platform.Vector $z_{k,i}^{j_i}$ is defined as: $z_{k,i}^{j_i} \in \{\text{null}, \mathbb{R}^n\}$, where null is part of the domain to model the possibility that an object is not detected, caused by a limited field of view.

Table 2.2 gives an overview of the parameters and functions involved in modelling faults in the system equations.

Table 2.2: Simulation parameters

Parameter	Fault class	Description
H_i	Localization fault	Fault entry matrix
$f_{k,i}^{loc}$	Localization fault	Fault vector
p	Existence fault	Probability of detecting clutter
$\mathcal{N}(\mu_i, \Sigma_i)$	Existence fault	Coordinate probability distribution clutter
c_{fov}	Existence fault	Parameter describing if object is detected

Chapter 3

Hardware redundancy-based fault detection

This chapter is structured as follows. First, Section 3.1 discusses a general framework for detecting faults with help of hardware-redundancy. Then, in Sections 3.2 and 3.3 methods to detect faults are derived for single and multiple objects, respectively.

3.1 Fault detection schemes

This section gives a general introduction to Hardware Redundancy-Based (HRB) fault detection. Since only measurements at one time step are compared mutually in this method, time indices are not relevant and thus will be dropped from notation, unless needed and explicitly stated otherwise. Further, it is assumed that every sensor only outputs one detection per time step, i.e., only one object is in the field of view of the perception sensors.
Fault detection based on hardware redundancy makes use of multiple sensors measuring the same entity x^j at a point in time. The measurements from the set of measurements $\mathcal{Z}$ are compared mutually and finding a healthy output (nominal measurements, i.e., not corrupted by faults) z_h, in a so-called voter. The most commonly used and applicable voters for the considered application are the majority, plurality and weighted average voters [16]. The working mechanism of these voting schemes will be briefly discussed in this section. Figure 3.1 shows a general scheme of such a system. Hardware redundancy can be exploited to detect faults in the measurements from perception sensors if their fields of view are overlapping.

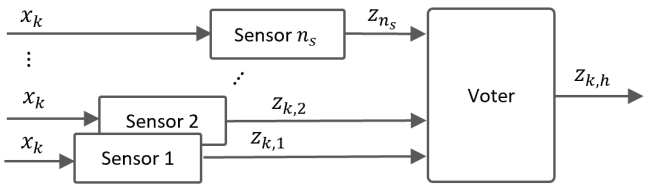

Figure 3.1: System with redundant modules and a voter.

An important advantage of redundant systems with $n_s \geq 3$ sensors is their fault-tolerance property. This means that if a fault occurs in a minority of the sensors, the fault can be detected, isolated and a output z_h can be found with help of the healthy measurements. Multiple fault-tolerant algorithms incorporating hardware redundancy exist, but the general framework of a voter consists of the following steps.

1. Compare measurements from $\mathcal{Z} = \{z_1, z_2, \ldots, z_{n_s}\} = \{z_i\}_{i=1:n_s}$ mutually and compute a degree of overlap, in terms of a distance metric.

2. Use the metrics computed in step 1 to distinguish between healthy and faulty measurements.
3. Based on the healthy measurements, compute final output value z_h.

In the following, a more detailed explanation supported by equations and expressions.

Step 1 distance metric on measurements
Measurements of object positions are neither exact nor Boolean values, therefore it is necessary to have a distance measure quantifying the order of equivalence between measurements. Therefore, the first step compares the measurements $\{z_i\}_{i=1:n_s}$ mutually, with a metric:

$$d_i^{i'} = d(z_i, z_{i'}). \tag{3.1}$$

There are many possible distance metrics to use. More details about the distance metric are given in the next section. The distances $d_i^{i'}$ for the measurement of one particular sensor i to any other sensor i' measurement, for which it thus holds that $i' \neq i$, are stored in the set D_i. For an example case with $n_s = 3$ sensors, D_1 would be as follows:

$$D_1 = \{d(z_1, z_2), d(z_1, z_3)\} := \{d_1^2, d_1^3\}.$$

The superset $\mathcal{D}$ contains the subsets D_i, i.e. $\mathcal{D} := \{D_1, D_2, \ldots, D_{n_s}\}$.

Step 2 determine healthy and faulty measurements
In the second step, the sensor reading from sensor i is declared faulty or healthy based on the distance metrics in D_i, computed in the previous step. The preceding is performed for all measurements, finally resulting in a set of fault indicators $f_{d,i} \in \{0, 1\}$:

$$\mathcal{F}_{loc} = \{f_{d,1}, f_{d,2}, \ldots, f_{d,n_s}\}.$$

The set of healthy sensor reading indices is described as $\mathcal{K} \subseteq \{1, 2, \ldots, n_s\}$ and $\mathcal{Z}_\mathcal{K} \subseteq \mathcal{Z} = \{z_1, z_2, \ldots, z_{n_s}\}$ describes the subset of healthy measurements. Another possibility is to assign a weight w_i to the sensor reading, depending on how well the sensor reading i matches with the other measurements $i' \neq i$. Voting supported with weights is known as weighted average voters. Possible methods to compute these weight are fuzzy [42] or soft voters [17].

Step 3 determine single output
Finally, based on the subset of healthy measurements $\mathcal{Z}_\mathcal{K}$, the merged output z_h can be computed. A simple average can be computed over the healthy measurements, or if weights w_i are computed in the previous step, a weighted average can be computed as an output. A possible approach is to use weights that are the inverse sensor noise covariance R_i matrices [43], in order to give more weight to more reliable sensors. The weighted average output is computed as follows:

$$z_{k,h} = \big(\sum_{i=1}^{n_h} \Lambda_i z_i\big)\big(\sum_{i=1}^{n_h} \Lambda_i\big)^{-1}, \tag{3.2}$$

with n_h the cardinality of $\mathcal{Z}_\mathcal{K}$, i.e., $n_h = |\mathcal{Z}_\mathcal{K}|$ and $\Lambda_i = R_i^{-1}$.

3.2 Single object: majority voter

In this section, first the problem setting will be specified, then relationships with regard to the nominal and the faulty state will be derived. Finally, the majority voter algorithm will be explained.

It is assumed only one object is detected by all sensors per time step and no existence faults occur. The state equations for modelling the ground truth state of a single object reduces to :

$$x_{k+1} = Ax_k + Bu_k + w_k, \tag{3.3}$$

the index j over the number of objects is dropped, since only one object is considered. The measurement equations are adapted as follows:

$$\begin{cases} z_{k,1} = C_1 x_k + v_{k,1} + H_1 f^{loc}_{k,1}, \\ z_{k,2} = C_2 x_k + v_{k,2} + H_2 f^{loc}_{k,2}, \\ \quad \vdots \\ z_{k,n_s} = C_{n_s} x_k + v_{k,n_s} + H_{n_s} f^{loc}_{k,n_s}. \end{cases} \tag{3.4}$$

The set of measurements is defined as $\mathcal{Z}_k = \{z_{k,i}\}_{i=1:n_s}$. The goal of the majority voter is to compute a single output value based on the healthy measurements, by detecting and isolating faulty measurements. By comparing the measurements by a distance metric it can be determined how similar those measurements are. By comparing a distance metric with a threshold, the measurements can be declared (dis)similar. When two measurements are dissimilar, one or both of the measurements might be in a degraded state, i.e. faulty. The order of similarity is determined by computing the difference between the pairs of measurements. When comparing measurements from sensors i and i' by computing the difference, we obtain the following:

$$z_{k,i} - z_{k,i'} = C_i x_k - C_{i'} x_k + v_i - v_{i'} + H_i f^{loc}_{k,i} - H_{i'} f^{loc}_{k,i'},$$

where $C_i = C_{i'}$. If, it is assumed there are no faults, (i.e., we consider the nominal state), in either sensor i' or i, the difference reduces to the following:

$$z_{k,i} - z_{k,i'} = v_i - v_{i'}. \tag{3.5}$$

The scalar $\gamma_{ii'}$ is defined as the random variable describing the difference between the measurement noise from v_i and $v_{i'}$, hence, $\gamma_{ii'} := v_i - v_{i'}$. The difference of the two multivariate normal random variables $v_i \sim \mathcal{N}(0, R_i)$ and $v_{i'} \sim \mathcal{N}(0, R_{i'})$ is a normal distribution described by:

$$\gamma_{ii'} = v_i - v_{i'} \sim \mathcal{N}(0, R_i + R_{i'}). \tag{3.6}$$

This can be proven with help of the characteristic function [44], a full proof is given in Appendix C.2. Based on this derivation, we can thus find that the difference in the nominal case is: $\gamma_{ii'} = v_i + (-v_{i'}) \sim \mathcal{N}(0, R_i + R_{i'})$. If a fault is injected in sensor i, Equation (3.5) changes to:

$$z_{k,i} - z_{k,i'} = v_i + H_i f^{\text{loc}}_{k,i} - v_{i'}, \tag{3.7}$$

we define $\tilde{\gamma}_{ii'} = v_i + H_i f^{\text{loc}}_{k,i} - v_{i'}$ to describe the difference of measurements for the faulty case. Assuming that $H_i f^{\text{loc}}_{k,i}$ is deterministic over time, the difference is again described by a normal distribution, with a shifted mean: $\tilde{\gamma}_{ii'} \sim \mathcal{N}(H_i f^{loc}_i, R_i + R_{i'})$. This knowledge can be used to derive a useful distance metric function $d(.,.)$ and accompanying similarity threshold ϵ later in this section.

In Section 3.1, a general approach to hardware redundancy fault detection was given. One simple algorithm to solve this problem is the majority voter [16]. Figure 3.2 shows the majority voter fault detection scheme [17].

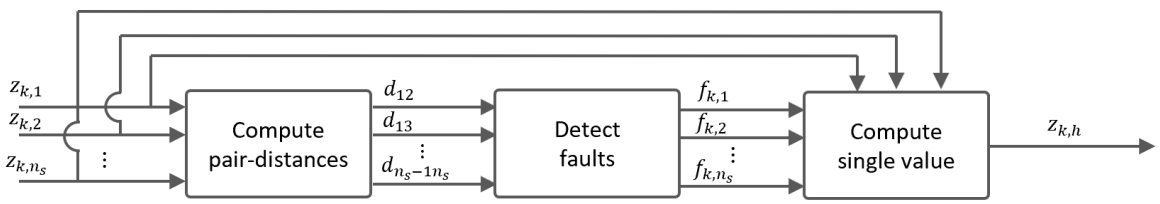

Figure 3.2: Majority voter.

As explained in the previous section, the algorithms work with three streps. First distances on any pair of measurements are computed. Then the distances are compared with a threshold to determine if a sensor

reading is faulty or not. Finally a single output value is computed. Algorithm 3.1 shows the pseudo code of computing the pair-wise distances for the majority voter algorithm.

1. **Input:** $\mathcal{Z}_k$ **Output:** $\mathcal{F}_{d,k}$
2. **Constants:** n_s, h
3. **for** $i = 1 : n_s$
4. Compute $D_i = \{d(z_i, z_{i'})|\forall i' \neq i\}$
5. **end**

Algorithm 3.1: Compute pair-wise distances.

A commonly used distance metric is the Euclidean distance

$$d_E(z_i, z_{i'}) = \sqrt{(z_i - z_{i'})^2}.$$

The Euclidean distance expresses the length of a line segment in Euclidean space between the two measurements. This is a suitable first crude solution, but it is not possible to express how likely the distance is in terms of a probability distribution. If knowledge is available about the probability distributions from the measurements, a suitable metric is the Mahalanobis distance:

$$D_{MD}(z_i, z_{i'}) = (z_i - z_{i'})^T \Sigma^{-1} (z_i - z_{i'}),$$

with $\Sigma = R_i + R_{i'}$ during nominal condition, prove given in Equation (3.6). This metric follows a χ^2_g probability distribution, with degrees of freedom g equal to the dimension of the observation space [45]. Therefore, it is possible to compute the probability of a measurement being in nominal or faulty state with help of the distance metric. To apply this metric, the variance of $(z_i - z_{i'})$ should be known.

In the second step, a sensor reading from sensor i is declared faulty or healthy based on the distance metrics in D_i, computed in the previous step. In order to declare a sensor reading z_i faulty or not, the distance with respect to all the other measurements, expressed with $d_i^{i'}$, are compared with a threshold:

$$\begin{cases} d_i^{i'} \leq \epsilon & \text{then } z_i \equiv z_{i'}, \\ d_i^{i'} > \epsilon & \text{then } z_i \not\equiv z_{i'}. \end{cases} \tag{3.8}$$

If the distance metric is smaller than the threshold, the measurements are equivalent, indicated with the symbol '$\equiv$'. If the distance metric exceeds the threshold ϵ, one of the measurements can be faulty and there is no consensus between measurements from sensor i and i', hence $z_i \not\equiv z_{i'}$. The number of equivalent measurements in set i is called the consensus number, $n_{c,i}$. Depending on the consensus number $n_{c,i}$, compared with a threshold h, a sensor reading z_i is declared faulty or healthy, setting $f_{d,i} \in \{0, 1\}$. The threshold variable h can vary according to the used method. A majority voter demands that the consensus number $n_{c,i}$ is equal or larger than the majority of measurements. The majority threshold $h \in \mathbb{N}_{>0}$, with n_s measurements, is defined as:

$$h = \begin{cases} \frac{n_s}{2} + 1, & \text{if } n_s \text{ even}, \\ \frac{n_s+1}{2}, & \text{if } n_s \text{ odd}. \end{cases} \tag{3.9}$$

The preceding is done for all measurements, finally resulting in a set of fault indicators $f_{d,i}$:

$$\mathcal{F}_{loc} = \{f_{d,1}, f_{d,2}, \ldots, f_{d,n_s}\}.$$

The set of healthy sensor reading indices is described as $\mathcal{K} \subseteq \{1, 2, \ldots, n_s\}$ and $\mathcal{Z}_\mathcal{K} \subseteq \mathcal{Z} = \{z_1, z_2, \ldots, z_{n_s}\}$ describes the subset of healthy measurements. The final coordinates are computed with help of Equation (3.2) and the set of healthy measurements. If the cardinality of the set of healthy measurements is smaller than h, it is not possible to determine what measurements are faulty and nominal based on the majority voter test. The final coordinates are now computed with help of all measurements and Equation (3.2). Figure 3.3 shows probability distributions $P(.)$ for nominal $\gamma_{ii'} \sim \mathcal{N}(0, 1)$ and faulty situations $\tilde{\gamma}_{ii'} \sim \mathcal{N}(f_1^{\text{loc}}, 1)$, with $f_1^{\text{loc}} = 1.5$. For illustrative purposes, a one-dimensional, $n = 1$, situation is considered. The plot shows the effect of varying the equivalence threshold ϵ influencing the probabilities of detecting faults (in)correctly

(true positive and false positive) and detecting (in)correctly the nominal state (true negative and false negative).

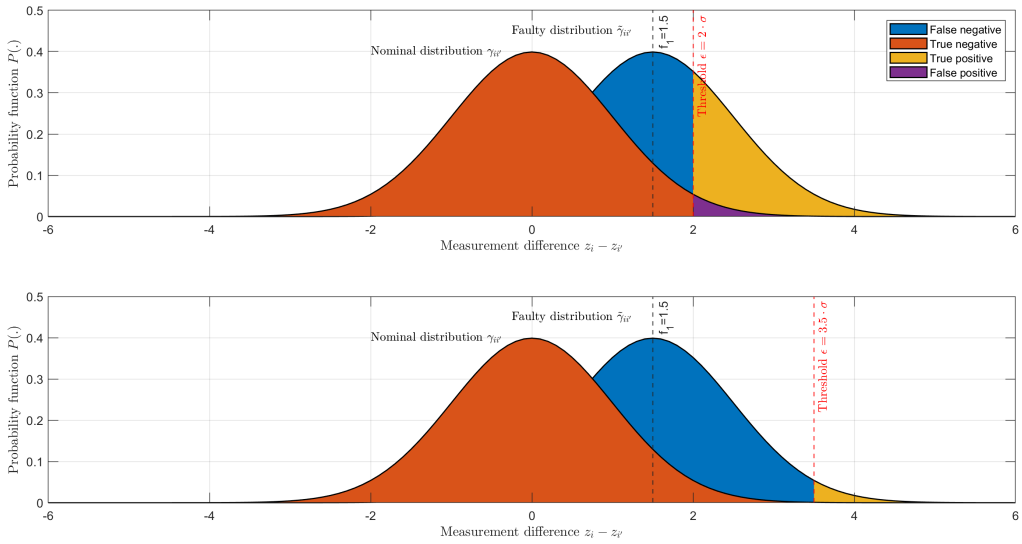

Figure 3.3: Influence of the equivalence threshold ϵ on the classification assessment.

To model existence faults for the case that a single object is considered, the equations given in Section 2.5 are now considered. The principle of the h out of n_s voter is used [18], as discussed in the literature survey in the introduction. This means that at least h out of n_s sensors need to confirm the detection of an object, to declare the object detection as a true positive object detection. Algorithm 3.2 shows the pseudo code of the majority voter algorithm for detecting existence faults as well.

1. **Input:** $\mathcal{Z}_k$ **Output:** $\mathcal{F}_{d,k}$
2. **Algorithm parameter:** $\mathcal{D}_{dmv}$
3. **Constants:** n_s, h
4. **if** $|\mathcal{Z}_k| \leq h$
5. Declare detections as existence faults, false positive object detections.
6. **elseif** $h < |\mathcal{Z}_k| < n_s$
7. Declare missing detections as existence faults, missed detections.
8. Apply regular majority voter algorithm to remaining detections.
9. **else**
10. Apply regular majority voter algorithm
11. **end**

Algorithm 3.2: Static majority voter algorithm, incorporating existence faults.

3.3 Multiple objects: clustering algorithm

In the previous sections, fault detection schemes are discussed for single object positions. Detecting faults in data measuring states for several objects is more complicated than for a single object, since it is not known what measurements belong to what object. Therefore this complexity is treated as a separate extension.

If the set of measurements $\mathcal{Z}_k$ contain detections of multiple objects, it is not possible to use a HRB fault detector directly. To this end, first the data that originates from the same object needs to be grouped together. Therefore, a hierarchical clustering algorithm [46] is implemented.

In the following, the clustering mechanism is explained. The clustering algorithm is initialized by setting every separate sensor reading as a singleton cluster. Singleton clusters are clusters containing only one sensor reading, one cluster $i_c = \{1, 2, \ldots, n_c\}$ containing one sensor reading is defined as:

$$C_{i_c} = \{z_{k,i}^{j_i}\}$$

with i_c the index over the number of clusters. In the clustering process, these clusters will be merged. By comparing the clusters mutually, according to a linkage criteria and a distance metric, the two closest clusters of detections can be merged. The linkage criteria determines what elements from clusters should be compared. The following linkage criteria are common for hierarchical clustering:

$$\begin{aligned}
&\max\{d(a,b) : a \in A, b \in B\},\\
&\min\{d(a,b) : a \in A, b \in B\},\\
&||c_s - c_t||,\\
&\frac{1}{|A| \cdot |B|} \sum_{a \in A} \sum_{b \in B} d(a,b),
\end{aligned}$$

which are the complete (or maximum), single (or minimum), centroid and average linkage clustering criteria, respectively. Clusters are defined as A and B with elements a and b respectively, for notation simplicity. Further, $d()$ is an unspecified metric, usually the Euclidean distance is used.

Merging clusters with the linkage criteria continues until a cluster contains as many measurements as sensors are present. Then, the cluster is finished and not considered in the clustering algorithm anymore. This process continues until there are no more incomplete clusters.

To improve the clustering, additional conditions are used to allow clusters to be merged. The first condition is that the clusters should not be too distant. The distance between the clusters can be compared with a threshold and it can be defined with assumptions on the sensor noise covariance magnitude. Furthermore, clusters containing measurements from the same sensor are not allowed to be merged, since it is assumed every sensor detects an object at most once. Algorithm 3.3 shows the clustering algorithm.

Input: set of measurements $\mathcal{Z}_k$ **Output:** set of clusters $\mathcal{C}_k$
Algorithm parameter: distance threshold T_{cl}

1. Start clustering: declare each detection as a singleton cluster.
2. Identify the two clusters that are closest together, with help of Euclidean distance and centroid linkage criterion.
3. Merge closest clusters into a new cluster, provided distance is smaller than set threshold T_{cl}.
4. If new cluster contains n_s singleton clusters, cluster is complete and prune cluster from process clustering algorithm.
5. Steps 2 and 3 are repeated untill:
   - No more clusters remain
     OR
   - Either the remaining clusters can not be merged, since intersection on the sensor indices between clusters are non-empty or the remaining clusters are too distant

Algorithm 3.3: Clustering algorithm

The clustering algorithm returns a set of clusters $\mathcal{C}_k$ containing n_c clusters C_{i_c}, with i_c the index over the number of clusters n_c. A cluster contains a subset of the set of measurements, i.e., $C_{i_c} \subseteq \mathcal{Z}_k$.

Figure 3.5 shows the clustering algorithm performance for several linkage criteria, with Euclidean distance as distance metric. Four objects are positioned equidistant, and are moved further away from each other, expressed by scaling factor c. The scaling factor can be interpreted as a normalized distance between the objects. Initially, the objects are positioned in the origin ($c = 0$). In the final position ($c = 1$), the distance between two objects is 5 m in x- and y-direction. Figure 3.4 shows the situations for $c = 1$ on the left and $c = 0.1$ on the right. The green diamonds depict the ground truth object states, the circles represent the measurements. Furthermore we have $n_s = 3$ sensors.

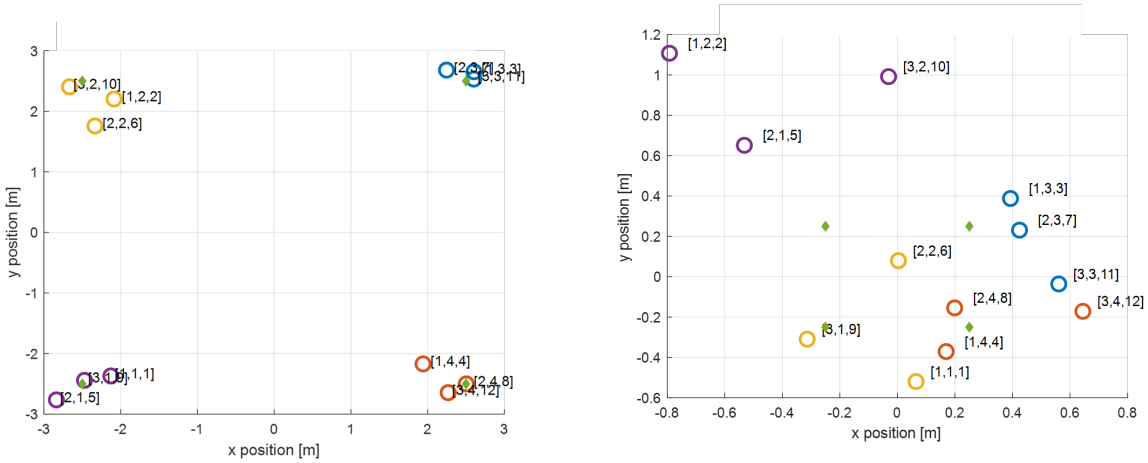

Figure 3.4: Problem setting considered for performance analysis clustering algorithm, left and right figures with scaling factors $c = 1$ and $c = 0$, respectively.

In a simulation, the distance is varied from $c = 0$ to $c = 1$. For the varying scale factor in the simulation, 500 samples were taken, and the fraction of correct clustered detections P is computed. Then the average of the fraction $\bar{P}$ of correctly clustered detections is computed and plotted against the scale factor, see Figure 3.5. This graph is plotted for the complete (or maximum), single (or minimum), centroid and average linkage clustering criteria, respectively. For a small and large scale factor, $c \to 0$ and $c \to 1$, the results are not relevant. For small scale values, the objects are very close to each other and the clustering performance is random. For larger scale factors the objects are very distant and the clustering will always be correct. Therefore the graphs are plotted on the domain $0.135 \leq c \leq 0.235$. In the first part and at the tail of the graphs it can indeed be observed the performance converges for the used methods. From Table 3.1 and Figure 3.5 we can observe that the performance of the criteria do not differ significantly.

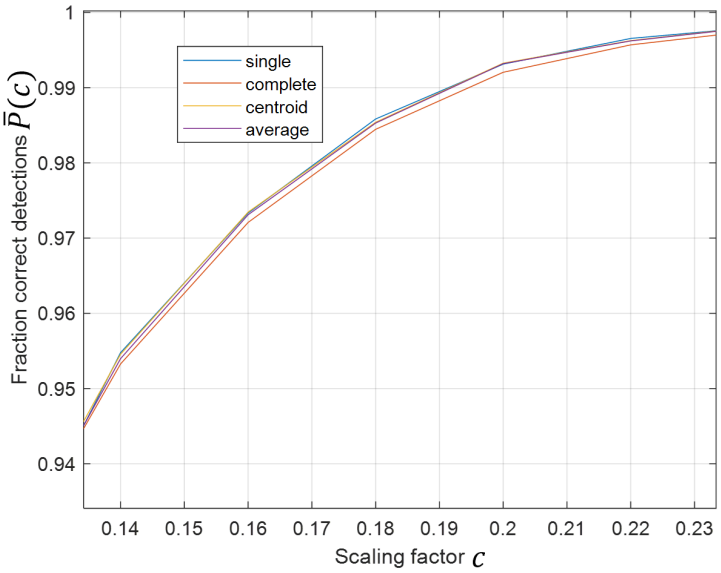

Figure 3.5: Performance of several linkage criteria, as function of mutual object distance.

Table 3.1

Linkage method	Single	Complete	Centroid	Average
Average $\bar{P}$	0.9747	0.9741	0.9750	0.9747

This chapter discussed an approach to exploit redundancy of the perception sensors in order to detect faulty measurements. The regular majority voter structure is modified to cope with noisy (nominal) measurements and to be effective when multiple objects are present in the field of view.

Chapter 4

Model-based fault detection

This chapter discusses model-based approaches towards fault detection for a perception system. Sections 4.1, 4.2 and 4.3 consider a single sensor and single object case, multiple sensors and a single object case and multiple sensors and multple objects situation, respectively. First, a short introduction on model-based fault detection is given. The core element of model based fault detection is the generation of residuals exploiting the concept of analytical redundancy. In order to provide useful information for FDI, the residual r_k should be as follows:

$$r_k \neq 0 \iff f_k \neq 0, \tag{4.1}$$

with f_k any type of fault, for now localization faults f_k^{loc} are considered. Examples of residuals are the measurement pre-fit and post-fit residuals, ν_k and η_k defined by Equations (2.24) and (2.28). A fault can be detected by comparing a residual evaluation function $J(r_k)$ with a threshold value T_k, according to the test:

$$\begin{cases} J(r_k) \leq T_k & \text{for } f_k^{loc} = 0, \\ J(r_k) > T_k & \text{for } f_k^{loc} \neq 0. \end{cases} \tag{4.2}$$

If the threshold is exceeded, the presence of a fault is likely [47]. Its likelihood is dependent on the used metrics. Possible residual evaluation functions are metrics such as the 2-norm or the Mahalanobis distance, with a threshold a positive constant. In [48], more details about residual evaluation functions are given. To indicate whether a fault is detected, we define the Boolean variable $f_d \in \{0, 1\}$ with the following functions:

$$f_d = \begin{cases} 1 & \text{if } J(r_k) > T_k, \\ 0 & \text{if } J(r_k) \leq T_k. \end{cases} \tag{4.3}$$

4.1 Single sensor and object

In a first step we consider $N_k = 1$ object in the environment and $n_s = 1$ sensor. The observation and prediction equations are now adapted to the environment and system setting:

$$\begin{cases} x_k = Ax_{k-1}, \\ z_k = Cx_k + v_k + Hf_{loc,k}. \end{cases} \tag{4.4}$$

$$\begin{cases} \hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1}, \\ \hat{z}_{k|k-1} = C\hat{x}_{k|k-1} \\ P_{k|k-1} = AP_{k-1|k-1}A^T \end{cases} \tag{4.5}$$

The process noise w_k and the control input Bu_{k-1} are omitted, since only static objects are considered in this chapter. Therefore, we assume there is no uncertainty in the system modelling equations nor input that will cause movement of the object, thus we define the covariance of the process noise as $Q = 0_{n \times n}$ and

$Bu_{k-1} = 0$. The objects are considered in a $n = 2$ dimensional plane and are defined as point objects. Both state variables are observable. Therefore the state and observation matrix are defined as follows:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The covariance $R \in \mathbb{R}^{2\times 2}$ of the measurement noise $v_k \sim \mathcal{N}(0, R)$ is assumed to be known:

$$R := \begin{bmatrix} \sigma^2_{x_1x_1} & \sigma^2_{x_1x_2} \\ \sigma^2_{x_2x_1} & \sigma^2_{x_2x_2} \end{bmatrix}.$$

The Kalman filter update equations do not need to be modified. In order to detect and isolate faults, the pre-fit residual $r_k := \nu_k$ is monitored as the relevant residual value. If a fault is detected, the measurement is isolated and not used to update the Kalman filter. Expanding the residual by writing the terms explicitly results into the following expression:

$$r_k := \nu_k = z_k - C\hat{x}_{k|k-1} = C(x_{k-1} - \hat{x}_{k|k-1}) + v_{k-1} + Hf_{loc,k-1}.$$

Figure 4.1 shows a a block diagram of the fault detection scheme integrated with the Kalman filter, for the single object and single sensor setting. In the following, it will be explained how the fault detection and isolation mechanisms work.

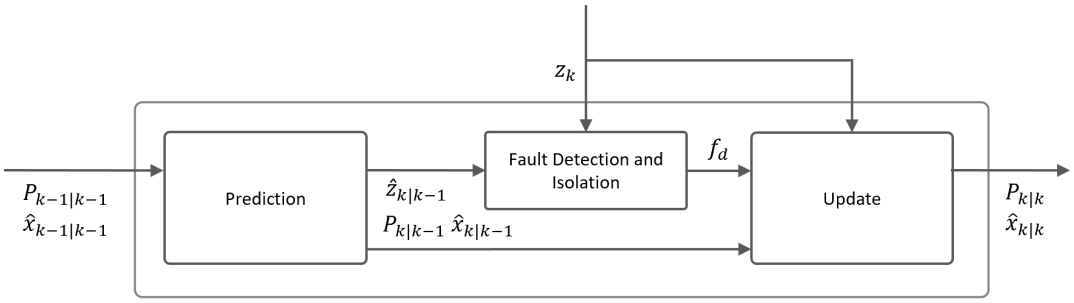

Figure 4.1: Block diagram of the fault detection scheme.

Approach 1
As a first simple fault detection scheme, the Euclidean norm and the a static threshold are used, as follows.

$$J(r_k) = ||r_k||_2 = ||z_k - \hat{z}_{k|k-1}||_2 \tag{4.6}$$
$$T_k = c_1\sigma_1 + c_2\sigma_2, \tag{4.7}$$

where c_1 and c_2 are tunable scalars and σ_1 and σ_2 are the standard deviation elements from the measurement noise covariance matrix. Algorithm 4.1 shows the peudo-code of the algorithm, as implemented in the 'Fault Detection and Isolation' block from Figure 4.1.

1. **Input:** z_k, $\hat{z}_{k|k-1}$, R **Output:** f_d
2. **Algorithm parameter:** c_1, c_2
3. Compute $J(r_k) = ||r_k||_2 = ||z_k - \hat{z}_{k|k-1}||_2$
4. If $J(r_k) > T_k$, set $f_d := 1$
5. If $J(r_k) \leq T_k$, set $f_d := 0$

Algorithm 4.1: Static threshold fault detector algorithm.

Approach 2
To make use of the sensor noise knowledge in a more meaningful way, a better approach is to use the Mahalanobis distance, inspired by [45]. The Mahalanobis distance for the residual is defined as:

$$D_{MD}(r_k)^2 = (z_k - \hat{z}_{k|k-1})\big(\Sigma\big)^{-1}(z_k - \hat{z}_{k|k-1})^T = r_k\Sigma^{-1}r_k^T. \tag{4.8}$$

To apply this method, the variance Σ of the residual $r_k = z_k - \hat{z}_{k|k-1}$ should be computed. To this end, first the prediction and observation model equations are substituted in the residual function:

$$r_k = \nu_k = z_k - \hat{z}_{k|k-1} = C(x_k - \hat{x}_{k|k-1}) + v_k.$$

Now, we can compute the variance of $r_k = z_k - \hat{z}_{k|k-1}$:

$$\mathbb{E}\{r_k r_k^T\} = C\mathbb{E}\{(x_k - \hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1})^T\}C^T + \mathbb{E}\{v_k v_k^T\},$$

with $P_{k|k-1} = \mathbb{E}\{(x_k - \hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1})^T\}$ and $R = \mathbb{E}\{v_k v_k^T\}$ and thus:

$$\mathbb{E}\{r_k r_k^T\} = CP_{k|k-1}C + R.$$

Now we can use $\Sigma := CP_{k|k-1}C + R$ for computing the Mahalanobis distance. By definition, under the assumption the previous modelling equations hold, the Mahalanobis distance follows a χ^2_d distribution with degrees of freedom d equal to the dimension of the residual, $r_k \in \mathbb{R}^2$. This means that if anomalies (faults) occur in a measurement, these can be found with a statistical test. Defined as follows, with residual evaluation $J(r_k) = p$ and threshold function $T_k = \alpha$, the p-value and the significance level α of the significance test, respectively:

1. H_0: $D_{MD} \sim \chi^2_d = \mathcal{N}(0,1)$
2. H_a: $D_{MD} \nsim \chi^2_d = \mathcal{N}(0,1)$
3. Test statistic: $p := 1 - \int_0^{D_{MD}} \chi^2_d(u)\mathrm{d}u$
4. Reject H_0 if $p < \alpha$, faulty measurement detected, set $f_d := 1$
5. Accept H_0 if $p \geq \alpha$, no fault detected, set $f_d := 0$

The threshold $\alpha \in (0,1)$ can be tuned based on the distribution of the residual during nominal operation, i.e. when there are no faults, and a desired false alarm rate. If a measurement is classified as faulty, while the system is in nominal operation, the detected fault is a false alarm. Since the distribution of the residual is known, α can be chosen to correspond to a set false alarm rate.
The statistical test can be written as an algorithm, see pseudo code in Algorithm 4.2. This algorithm is implemented in the 'Fault Detection and Isolation' block, Figure 4.1.

1. **Input:** z_k, $\hat{z}_{k|k-1}$, R, $P_{k|k-1}$ **Output:** f_d
2. **Algorithm parameter:** $T_k = \alpha$
3. Compute: $D_{MD}(r_k)^2$.
4. Compute p-value: $J(r_k) = p := 1 - \int_0^{D^2_{MD}} \chi^2_d(u)\mathrm{d}u$
5. If $p < \alpha$, faulty measurement detected, set $f_d := 1$
6. If $p \geq \alpha$, no fault detected, set $f_d := 0$

Algorithm 4.2: χ^2-fault detector algorithm.

4.2 Multiple sensors, single object

As an extension to the preceding, we now consider any number of sensors, $n_s \in \mathbb{N}$, and a single object, and thus single Kalman filter $N = 1$. Since there are now multiple sensor readings per Kalman filter, the sensor readings need to be fused, possible methods to do this are the group-sensor and sequential fusion methods [36]. In this section, the group-sensor method is considered. First it is described how the Kalman filter works for the fault free case, then how a fault detector is integrated in the fusion process. To this end, the system model equations are now:

$$\begin{cases} x_k = Ax_{k-1}, \\ z_{k,1} = C_1 x_k + v_{k,1}, \\ z_{k,2} = C_2 x_k + v_{k,2}, \\ \quad \vdots \\ z_{k,n_s} = C_{n_s} x_k + v_{k,n_s}. \end{cases} \tag{4.9}$$

The set of sensor readings at time step k is defined as:

$$\mathcal{Z}_k = \{z_{k,1}, z_{k,2}, \ldots, z_{k,n_s}\}.$$

With prediction equations:

$$\begin{cases} \hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1}, \\ \hat{z}_{k|k-1} = C\hat{x}_{k|k-1}, \\ P_{k|k-1} = AP_{k-1|k-1}A^T. \end{cases} \tag{4.10}$$

With update equations:

$$\begin{cases} \nu_k = z_k - C\hat{x}_{k|k-1}, \\ K_k = (P_{k|k-1}C^T)(CP_{k|k-1}C^T + R)^{-1}, \\ \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\nu_k, \\ P_{k|k} = (I - K_kC)P_{k|k-1}. \end{cases} \tag{4.11}$$

As can be observed, the conventional Kalman filter algorithm equations can be used, but the following variables need to be changed to incorporate multiple sensor readings in the prediction and update equations:

$$z_k := \left[z_{k,1}^T, z_{k,2}^T, \ldots, z_{k,n_s}^T\right]^T, \tag{4.12}$$

$$C := \left[C_1^T, C_2^T, \ldots, C_{n_s}^T\right]^T, \tag{4.13}$$

$$v_k := \left[v_{k,1}^T, v_{k,2}^T, \ldots, v_{k,n_s}^T\right]^T \tag{4.14}$$

where

$$\begin{aligned} R &:= \mathbb{E}\{v_k v_k^T\} \\ &:= \mathbb{E}\{\left[v_1^T, v_2^T, \ldots, v_{n_s}^T\right]^T \left[v_1^T, v_2^T, \ldots, v_{n_s}\right]\} \\ &:= \text{blockdiag}\{R_1, R_2, \ldots, R_{n_s}\}. \end{aligned} \tag{4.15}$$

The above derivation holds when the sensor measurement noises are not cross-correlated at time step k,[36], i.e.,

$$\mathbb{E}\{v_{k,i}v_{k',i'}^T\} = R_i\delta_{ii'}\delta_{kk'},$$

where $\delta_{ii'}$, $\delta_{kk'}$ are the Kronecker delta functions.
In the following it is explained how the sensor fusion works in combination with a fault detection framework. First, the system model equations need to be adapted, to model faults:

$$\begin{cases} x_k = Ax_{k-1}, \\ z_{k,1} = C_1x_k + v_{k,1} + H_1f_{loc,k}, \\ z_{k,2} = C_2x_k + v_{k,2} + H_2f_{loc,k}, \\ \quad \vdots \\ z_{k,n_s} = C_{n_s}x_k + v_{k,n_s} + H_{n_s}f_{loc,k}. \end{cases} \tag{4.16}$$

Figure 4.2 shows the placement of the fault detection algorithm in the overall sensor fusion process.

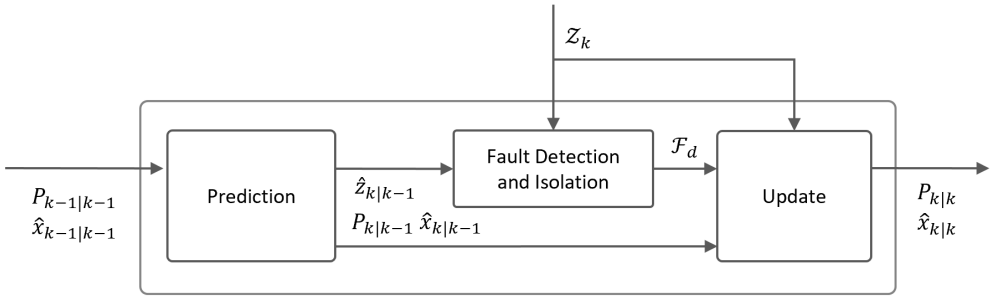

Figure 4.2: Block diagram of the sensor fusion scheme.

The fault detection algorithm for a single sensor case as described in the pseudo code of Algorithm 4.2, can be used for multiple sensor case as well. The residual function $J(r_{k,i})$ is now evaluated per sensor reading i. To this end, the Mahalanobis distance is modified as follows:

$$D_{MD}(r_{k,i})^2 = (z_{k,i} - \hat{z}_{k|k-1})\big(\Sigma_i\big)^{-1}(z_{k,i} - \hat{z}_{k|k-1})^T = r_{k,i}\Sigma_i^{-1}r_{k,i}^T, \tag{4.17}$$

with $\Sigma_i = CP_{k|k-1}C + R_i$. A detection threshold $T_{k,i} := \alpha_i$ per sensor can be set. The algorithm is presented in Algorithm 4.3.

1. **Input:** $\mathcal{Z}_k$, $\hat{z}_{k|k-1}$, R_i, $P_{k|k-1}$ **Output:** $\mathcal{F}_{d,k}$
2. **Algorithm parameter:** $T_{k,i} = \alpha_i$
3. **for** $i = 1 : n_s$
4. Compute: $D_{MD}(r_{k,i})^2$.
5. Compute p-value: $J(r_{k,i}) = p := 1 - \int_0^{D^2_{MD}} \chi^2_d(u)\mathrm{d}u$
6. If $p < \alpha_i$, faulty measurement detected, set $f_{d,i} := 1$
7. If $p \geq \alpha_i$, no fault detected, set $f_{d,i} := 0$
8. Store $f_{d,i}$ in $\mathcal{F}_{d,k}$.
9. **end**

Algorithm 4.3: χ^2-fault detector algorithm for multiple sensors.

Figure 4.3 shows the mechanism of the fault detection algorithm in a block diagram.

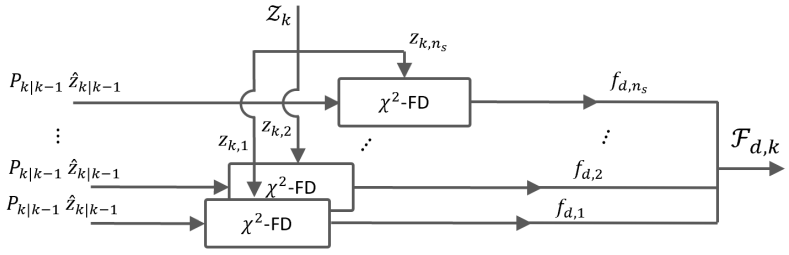

Figure 4.3: Block diagram of the fault detection scheme.

With help of $\mathcal{F}_{d,k}$ as output from the fault detection block, it is known what sensor readings are declared faulty. In the consecutive update step, the faulty measurements should not be used to compute the state and covariance estimates $\hat{x}_{k|k}$ and $P_{k|k}$. To this end, the variables in Equations (4.12) till (4.15) should contain only the values corresponding to the healthy sensors. The healthy sensor index subset $\mathcal{K}_k \subseteq \{1, 2, \ldots, n_s\}$, from the sensor index set $\mathcal{S} = \{1, 2, \ldots, n_s\}$, can be obtained with help of $\mathcal{F}_{d,k}$. Now the variables belonging to the faulty sensor indices are pruned from Equations (4.12) till (4.15), resulting into the following array of measurements, observation matrix, and measurement covariance matrix only containing elements belonging to healthy sensors: $z_{k,\mathcal{K}}$, $C_{k,\mathcal{K}}$, $R_{k,\mathcal{K}}$.

4.3 Multiple sensors and multiple objects

Figure 4.4 shows the block diagram of the filtering process for the problem setting considering multiple sensors and multiple objects in the environment. Before it is possible to apply fault detection, the received sensor readings from the set $\mathcal{Z}_k$ must be associated to the tracks, with help of predictions from the set $\hat{Z}_{k|k-1}$. To this end, an additional step is used, called 'Data Association'. Section 2.3 documents how the data association algorithm works. After the data association step, the same fault detection algorithms as discussed in Figure 4.3 can be utilized. The challenge is to find a gate that includes localization faults, but does detect existence faults by excluding them from the gate. The gate is defined by the confidence ellipsoid during the data association step, if a very conservative gate is chosen, the object detections will likely be included in the

data association step. But, if a fault is injected in one of the sensors with a sufficient magnitude, it is possible the fault falls outside the gate. This will cause the fault to remain undetected as a localization fault.

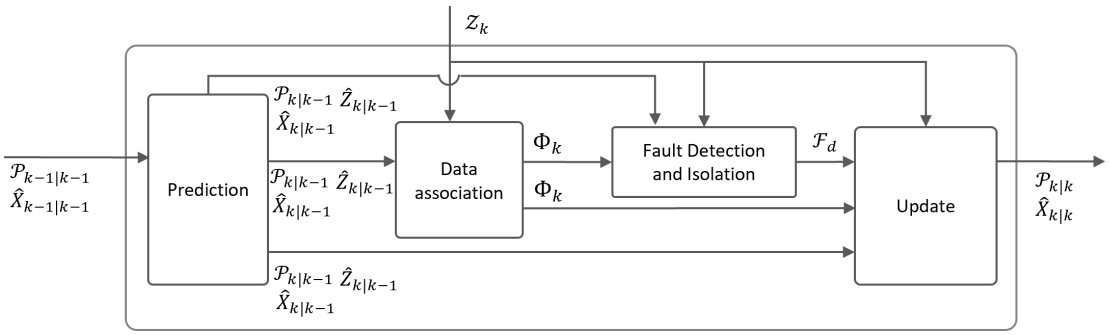

Figure 4.4: Block diagram of the fault detection scheme.

This chapter showed an approach towards detecting faults with help of a model-based approach, based on the knowledge of the sensor noise models.

Chapter 5

Simulation and performance analysis

This chapter is organized as follows. First, in Section 5.1, the used performance metrics for the simulations are discussed. Next, in Section 5.2 the simulation results are presented.

5.1 Performance metrics

According to the mentioned goals, presented in the introduction Section 1.4, faults should be detected and objects tracked with the system corrupted by faults. To asses how well the objects are tracked and faults are detected, a number of performance indicators are defined. To this end, a number of fault detection performance indicators are defined.

Detecting faults can be defined as a binary classifier problem. In the actual situation, there is either a fault in the sample data, indicated with P (Positive), or no fault in the sample data, indicated with N (Negative). The classifier, in this case the fault detection algorithm, predicts based on a test if there is a fault in the sample PP (Predicted Positive) or no fault in the sample data, PN (Predicted Negative). Table 5.1 shows how the classification outcomes relate to the predictions and actual values. This table is commonly referred to as the confusion matrix (or table).

Table 5.1: Confusion matrix.

		Prediction	
		PP	PN
Actual	P	True Positive (TP)	False Negative (FN)
	N	False Positive (FP)	True Negative (TN)

A very good fault detector has a very high true positive and true negative score, and a very low false positive and false negative score. The amount of true positives or false positive classifications on itself are meaningless. In order to obtain significant performance values, the amount of true positive detections, or any other classification, needs to be compared to the other classifications. To this end, the performance metrics are defined as a fraction of other classifications. Often used metrics are the detection rate and false alarm rate, given by:

$$\begin{aligned} \text{Detection rate} &= \frac{\text{TP}}{\text{TP}+\text{FN}} = \frac{\text{TP}}{\text{P}}, \\ \text{False alarm rate} &= \frac{\text{FP}}{\text{FP}+\text{TN}} = \frac{\text{FP}}{\text{N}}, \end{aligned} \tag{5.1}$$

Detection Rate (DR) can be interpreted as the fraction of actual detected faults, out of all faults. Detection rate is also known as recall, sensitivity or True Positive Rate (TPR). False Alarm Rate (FAR) is defined as the percentage of false alarms encountered as a fraction of all data without faults. False alarm rate is also

known in literature as Fall-out or False Positive Ratio (FPR). The Receiver Operating Characteristic curve (ROC-curve) is a graph plotting the true positive rate against the false alarm rate as a function of a classifier parameter (usually the detection threshold) [49]. With help of this curve, a user of the algorithm can choose a desired threshold based on the amount of false alarms that are tolerated. Equation (5.2) shows how the actual classifier performance equations are implemented in the time domain, with $k \in \{1, 2, \ldots, K\}$ the time index over all time steps.

$$\begin{aligned} \text{Detection rate} &= \frac{\sum_{k=1}^{K} \text{TP}(k)}{\sum_{k=1}^{K} (\text{TP}(k) + \text{FN}(k))} = \frac{\sum_{k=1}^{K} \text{TP}(k)}{\sum_{k=1}^{K} \text{P}(k)}, \\ \text{False alarm rate} &= \frac{\sum_{k=1}^{K} \text{FP}(k)}{\sum_{k=1}^{K} (\text{FP}(k) + \text{TN}(k))} = \frac{\sum_{k=1}^{K} \text{FP}(k)}{\sum_{k=1}^{K} \text{N}(k)}. \end{aligned} \tag{5.2}$$

5.2 Simulations

This section presents the results of the simulations, made with help of the models from Chapter 2 and the algorithms from Chapters 3 and 4. First, in Section 5.2.1, the performance of the fault detection algorithms developed in Chapters 3 and 4 are discussed with help of ROC-curves. In Section 5.2.2 the performance of the model-based fault detectors is assessed for multiple objects. First, an introductory example is discussed that serves as a motivation for further analysis, in which the influence of the gate size is investigated. Finally, in Section 5.2.3 the performance of the majority voter combined with the model-based approach is discussed. A decoy (false object detection) is introduced in one sensor and it is shown that no track is initialized.

5.2.1 Effect of localization fault magnitude on detection performance

In this section, the performance of the majority voter algorithms (as discussed in Section 3.2) with the static and chi-squared threshold, and the model-based algorithms with static (Algorithm 4.1) and chi-squared threshold (Algorithm 4.2) are assessed in terms of the detection and false alarm rate as a function of the detection threshold, with help of ROC-curves.

The results of the ROC-curves are obtained by looping over various fault detection threshold values for varying fault magnitudes. For every fault magnitude and detection threshold combination, the simulation was run for $K = 250$ samples, and the average true detection and false alarm rates are computed. The resulting values can be plotted in the ROC-curves.

Figure 5.1 shows the problem setting in a two-dimensional plane for varying fault size f_1^{loc}, with $n_s = 3$ sensors, with localization fault injected in sensor 1. The fault magnitude is constant for each diagram, but the direction is not, therefore the faulty measurements appear in a circular pattern around the nominal measurements. The model-based fault detection approach uses the same setting, but with the difference that $n_s = 2$ sensors are used.

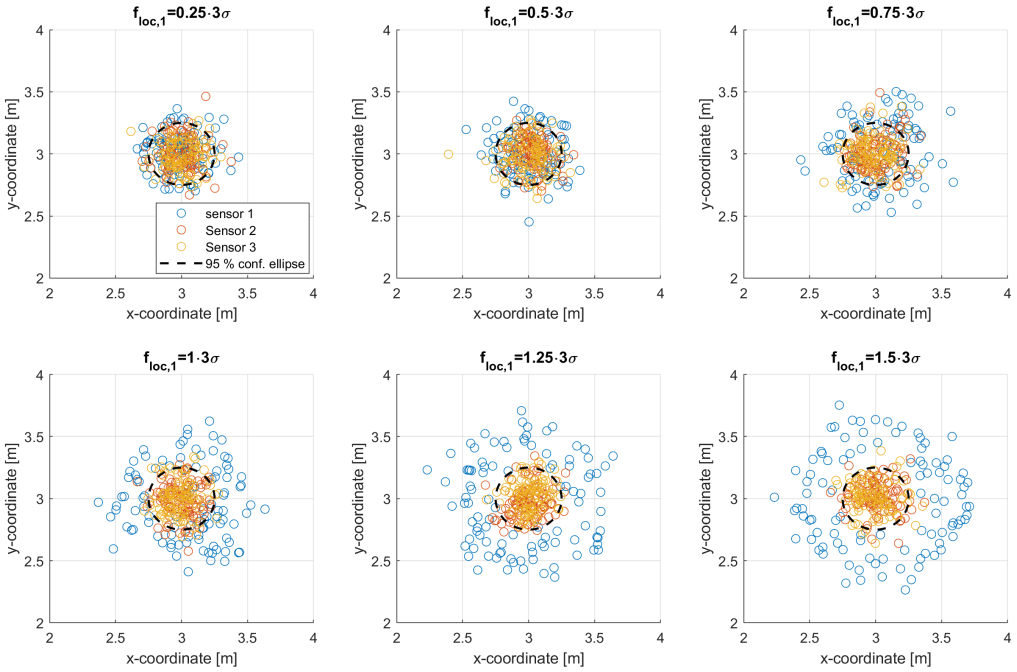

Figure 5.1: Several diagrams, with each diagram showing sensor readings for a set fault magnitude in sensor 1.

Figure 5.2 shows the ROC-curves for the model-based fault detection algorithms. The performances for both the deterministic and chi-squared algorithms are for the given simulation equal. The measurement noise covariance is chosen as a diagonal matrix: $R_i = \begin{bmatrix} \sigma_{11}^2 & 0 \\ 0 & \sigma_{22}^2 \end{bmatrix}$, with $\sigma_{11} = \sigma_{22} = \frac{1}{8}$ for all sensors, which makes static fault detector also suitable to detect faults. When this is not the case, the detection threshold is not able to perform as well, since not only the noise size, but the noise direction plays also a role. Another advantage of the chi-squared fault detector is that the detection threshold is linearly related to the false alarm rate. The general tendency is that when decreasing the detection threshold, for a set fault magnitude, the true detection rates increases. However, decreasing the detection threshold comes at the cost of a higher false alarm rate. The fault magnitude itself also influences the detection rate performance. For a set threshold, the detection rate increases for increasing fault magnitude. Choosing the threshold should be related to the smallest fault one desires to detect and the tolerated false alarm rate.

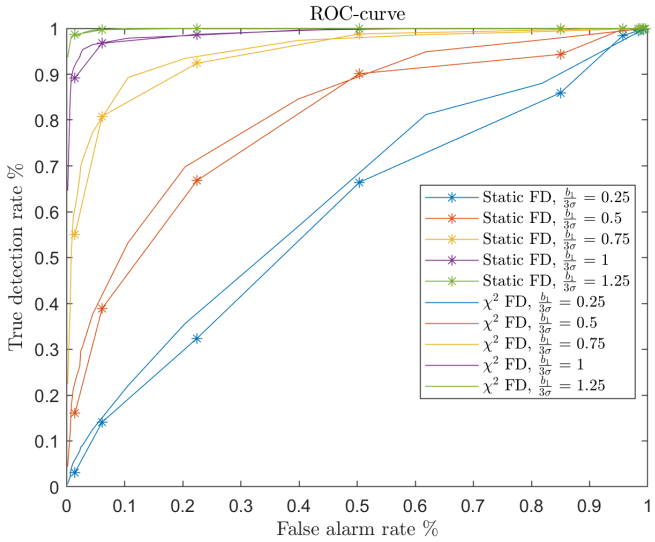

Figure 5.2: ROC-curves for several fault magnitudes, as a function of the thresholds α (significance level, for the chi-squared test) and c (constant threshold, for the static fault detector) and with $b_1 = f_1^{loc}$.

Figures 5.3 and 5.4 show the ROC-curves for several faults magnitudes as a function of a varying fault detection thresholds for the majority voter with chi-squared (CS) and static distance (SD) metrics. It shows again that for increasing fault magnitude the detection rate improves. The chi-squared fault detector shows improved performance over the static fault detector for the same false alarm rates, compared in terms of the detection rate. To this end, the CS majority voter is favored over the SD majority voter.

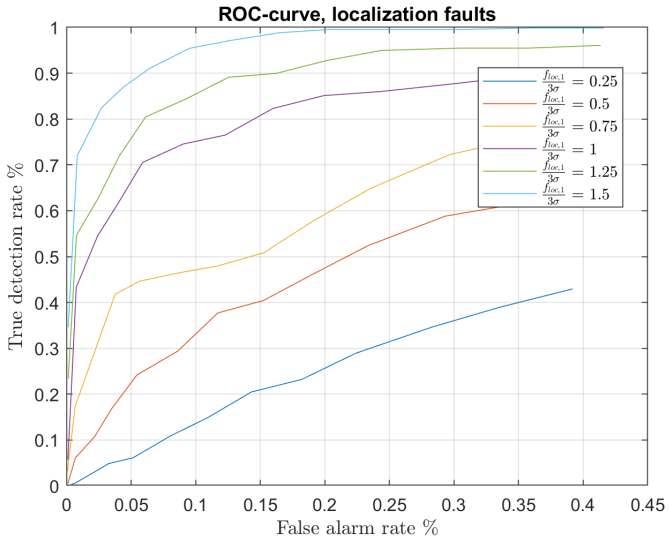

Figure 5.3: Majority voter ROC-curve with chi-squared distance metric.

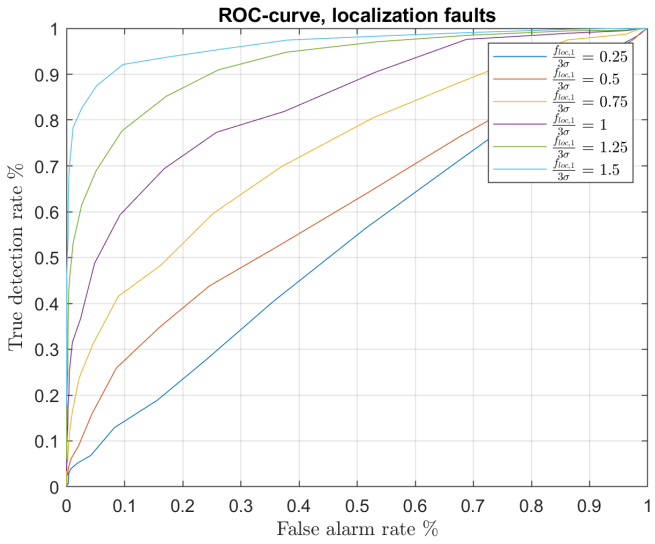

Figure 5.4: Majority voter ROC-curve with static distance metric.

5.2.2 Static environment

Introductory example

The use case discussed in this section, serves the purpose to show what the advantages and challenges of the chi-squared fault detection mechanism are, discussed in Section 4.3. Figure 5.5 shows the estimated and measured coordinates of two static objects plotted over time. In Appendix B.2, a figure can be found showing the sensor readings in a two-dimensional plane. Starting from time step $k = 22$ a bias fault is introduced in sensor 1, with random angle and magnitude $f_1 = 2\sigma$. In Figure 5.5 the dashed vertical line in the figure indicates the time the fault is injected, the blue line is the Kalman filter output and the other graphs are the noisy sensor measurements (with $n_s = 2$).

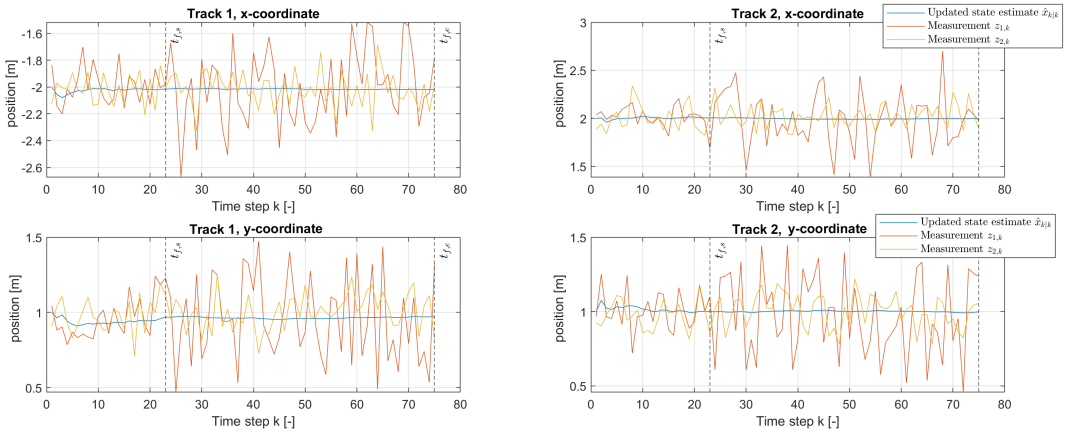

Figure 5.5: Measurements and state estimates over time.

Figures 5.6 shows detected localization faults per object over time. Figure 5.7 shows the detected false object detections over time. Since only a localization fault is injected in sensor 1, only the fault detections in the two upper subplots of figure 5.6 are true fault detections, the other detected faults are false alarms. The faults detected in any of the sensors before a fault is injected, are also all false alarms. The false alarms with regard to the false object detections are caused by sensor readings that fall outside the gate during data association. Unassociated measurements are used to initialize a new track if conditions are met, otherwise those are

declared false object detections. The next section studies the influence of gate size and fault magnitude on the fault detection performance.

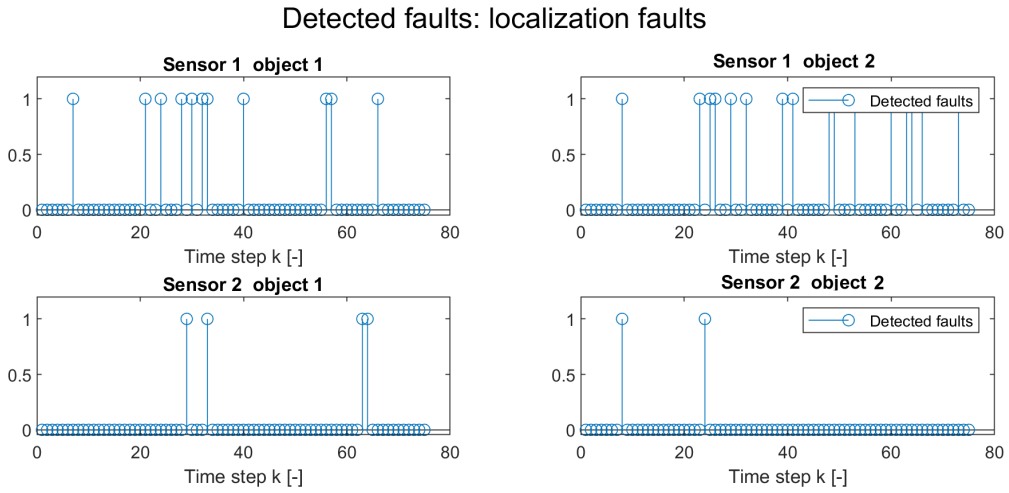

Figure 5.6: Detected localization faults over time.

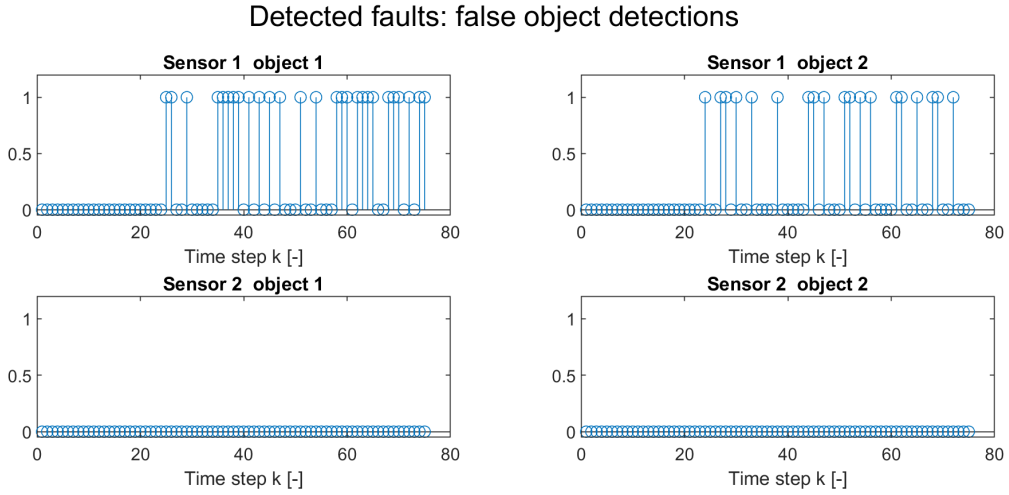

Figure 5.7: Detected false object detections over time.

Data association gate and fault magnitude

Figure 5.8 shows the detection rate of the localization faults and of the false object detections. The problem setting is the same as in the previous section, but this situation is now analyzed as a function of the gating threshold size and the fault magnitude. By increasing the gate size, the probability of including all detections in the gate increases. Very large faults still fall outside the gate and are thus not detected. Small faults that fall inside the gate, are not always detected since the chi-squared test declares them insignificant for the set threshold. This explains the yellow disc shaped area towards the left in the graph.

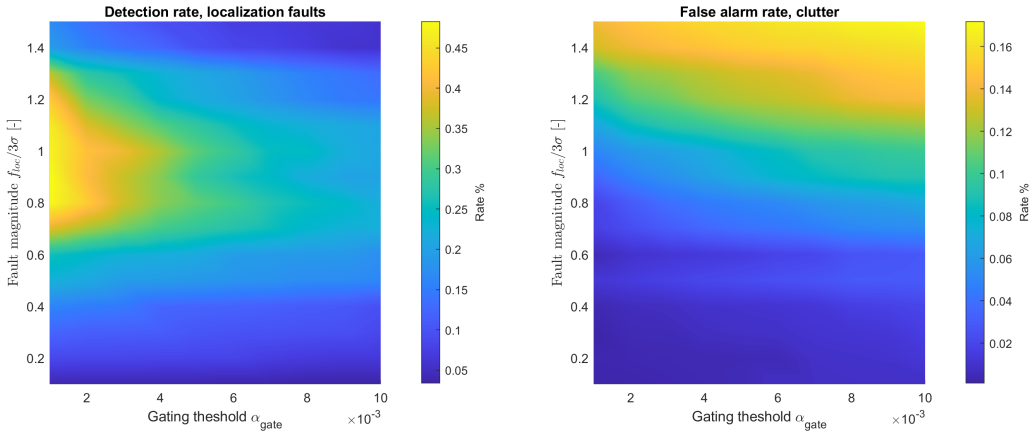

Figure 5.8: Colour maps for the localization detection rate and false alarm rate of localization and false object detections as a function of the fault magnitude and gating size. The gate is expressed in terms of the confidence region α_{gate}, related to the gate threshold d_{gate} by the chi-squared distribution.

5.2.3 Dynamic environment

This section considers a use case for which a decoy $c^1_{k,1} = (35, 0)$ is injected in sensor 1 at time step $k = 5$. The field of view of the $n_s = 3$ sensors is limited with ranges $l_1 = l_2 = l_3 = 20$ m and opening angles $\phi_1 = \frac{\pi}{6}$ Rad, $\phi_2 = \frac{\pi}{2}$ Rad and $\phi_3 = \frac{\pi}{3}$ Rad, see Figure 5.9. An actual static object is in the environment positioned at $x = (15, 12)$. The platform travels in a straight line with velocity $v = 2.5$ m/s. Object x^1_k is detected from the start of the simulation.

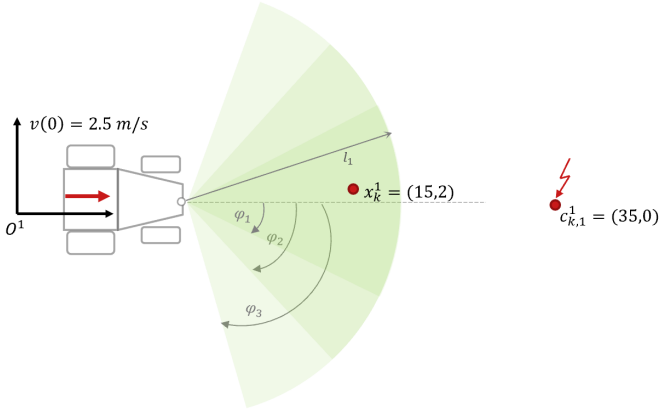

Figure 5.9: Overview of considered use case.

When the object enters the field of view, a track is initialized since a majority of the sensors detect the object. Figure 5.11 shows the state estimates computed by the Kalman filter and the associated sensor measurements. The track is pruned at $k = 22$ since there were no more detections for 15 (track deletion tuning parameter $N = 15$) consecutive time steps. Figure 5.12 shows that the decoy injected in sensor 1 is detected as a false object detection, since only 1 sensor receives this measurement, hence no track is initialized.

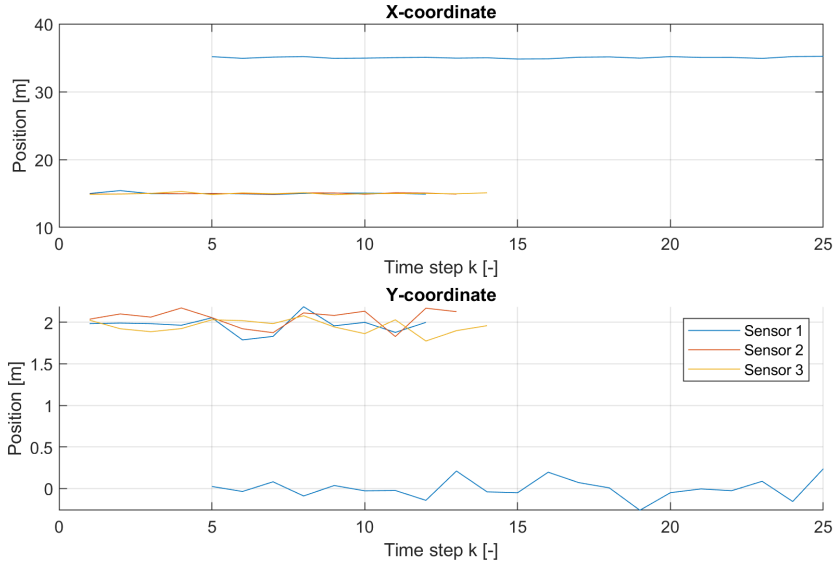

Figure 5.10: Sensor readings in x and y coordinates plotted for the three sensors over time.

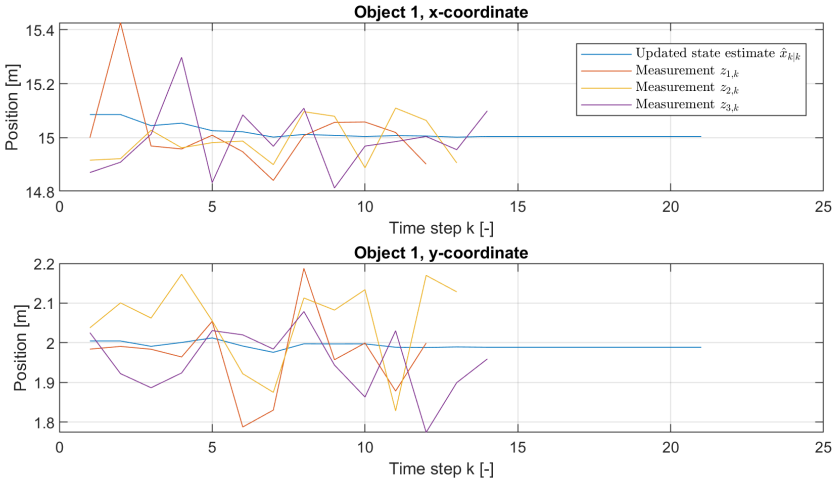

Figure 5.11: Kalman filter state estimates and associated sensor readings.

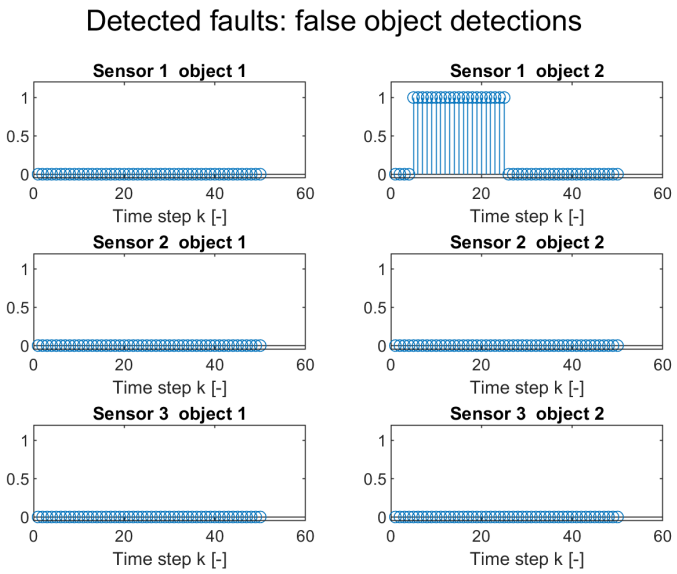

Figure 5.12: Detected false object sensor readings for all sensors and object tracks.

In conclusion, it is observed that object tracks are only initialized when those are confirmed by all available sensors in the field of view.

Chapter 6

Conclusion and discussion

First, Section 6.1 presents some conclusions with help of the project goals defined in the introduction. Then, Section 6.2 provides a discussion about how the set assumptions would influence the performance of fault detection framework when applied in a real world setting. Based on this and limitations of the provided solutions, the section gives suggestions for future work.

6.1 Conclusion

The goal for this graduation project was to develop a fault detection and isolation framework for the perception system of an autonomous tractor. To this end, a model of the perception system and environment to asses the performance was developed. Based on the sensor models in degraded state, fault detection schemes were derived in order to detect the anomalies in the observation models. Finally, the performance of these algorithms was assessed with help of simulated use cases. The remainder of the conclusion will be discussed with help of the sub-goals as defined in Section 1.4.

1) Develop a fault detection framework for a work-drive system of Flanders Make that is able to detect and isolate faulty data in the environment perception system.
Fault detection and isolation is in general applied to systems where the variable of interest is measured directly and the origin of this measurement is known. Examples are a global positioning system [45] or servo control systems in aviation applications [50]. Detecting faults by using the output from perception sensors is accompanied by the challenge that these sensors do measure the variables of interest (i.e., in this project object positions) directly, but for the consecutive time steps it is unknown which measurements originate from what object in the real world.

Therefore, first the received sensor readings need to be structured before a fault detection method can be applied. The structuring refers to a method of grouping sensor readings that belong to the same object. To structure the data, a clustering algorithm with appropriate constraints or data association method can be used. Data association is used when a track is available as a reference, otherwise clustering can be used. After the data is structured, it is possible to detect faults by comparing the measurements mutually or with the reference value from the track. Then the definitions of the nominal sensor models, faulty sensor models, and state estimates can help by defining appropriate fault detection and isolation tests. With help of these definitions, a static and hypothesis-based fault detector was derived for both the hardware-redundancy as the model-based fault detection algorithms.

2) Develop a fault detection framework for a work-drive system of Flanders Make that is able to distinguish between localization and existence faults.
The model-based fault detection scheme classifies object detections that remain unassociated during data association, and are not used to initialize a new track, as false object detections. If the magnitude of a localization fault increases towards a point that it falls outside a gate, this measurement will wrongly be classified as a existence fault. To make sure localization faults are correctly classified, the gate size should be

chosen such that the largest fault magnitude that is desired to be detected, falls within the gate.

3) Develop a fault detection framework for a work-drive system of Flanders Make that is able to fuse data from multiple sensors in order to track and localize objects.
Data from multiple sensors are fused by using the information from multiple sensors in an augmented measurement vector and using it in the Kalman filter. Object tracks are initialized with help of the majority voter.

4) Implement the framework in a simulation environment and evaluate the performance.
In the introduction the functional architecture of the perception system was presented in Figure 1.2. The relevant parts of this system and the environment were modeled in order to asses the performance of the fault detection schemes, as explained in Chapter 2. The environment objects and nominal sensor models are modelled by discrete time linear state space equations. The observation models were adapted with additional terms and functions to model additive faults and limited field of view, respectively.

6.2 Discussion and future work

This section discusses how limitations of the fault detection schemes and assumptions used to develop models and fault detection schemes influence the performance when it would be implemented in the real world system. Based on the limitations, suggestions for future work are given.

The host platform is equipped with cameras and lidars. Objects detected by those sensors are typically represented by bounding boxes in two dimensions (for cameras) and in three dimensions (for lidars), while the sensor model outputs measurements as point objects in a two-dimensional plane. In order to apply the fault detection schemes on the measurements from a lidar, the centers of geometry of the three-dimensional bounding boxes can be used. In order to apply fault detection on the camera detections, more modifications are needed, since it is not trivial to obtain depth information from these object detections. In [24] a fault detection method was developed for a perception system that outputs two-dimensional bounding boxes.

In this work, only the performance of the fault detectors was assessed in an environment with static objects. Tracking dynamic objects poses the additional challenge that predicting the objects states becomes more complex, when they are not moving with constant speed. If an object is moving with constant speed, this can be augmented in the state vector and the prediction equations need to be adapted accordingly. In reality, humans and other dynamic objects in the environment are not moving with a constant speed but (de)accelerate often during their trajectory. The change of speed can be modeled with help of the control input vector in the state space equations. However, when performing a prediction step, the tracking system does not know what the values of the control inputs are, since these take place in an external object. To cope with this problem, a first approach could be to use a larger process noise, indicating there is a high modelling uncertainty. A dedicated algorithms to cope with this problem is the interacting Multiple Model (IMM) algorithm [51].

Further research based on the work of this project, could be to further investigate the performance between the model-based and hardware-redundancy based approaches, also in a simulation setting. A majority voter type fault detector could work better in circumstances where the model is updated accidentally with faulty data, which would result in faulty reference predictions in subsequent time steps. When computing model-based residuals, the faulty measurements can be regarded as healthy. When a majority voter type fault detector is used, only the mutual sensor readings are compared, if the fault magnitude is sufficiently large, this fault can be detected, provided that the majority amount of sensors is healthy. The risk of this particular case will especially increase in the presence of drift faults, since a drift will cause the reference (model-based) values to gradually increase over time.

The chi-squared fault detection algorithm makes use of the measurement covariances, which are assumed to be known. In reality, the estimated covariance can differ significantly or vary over time. In order to cope with varying measurement noise, for robust fault detection, the measurement covariance can be updated real-time. In [45] an approach was given to solve this problem, by computing the residual over a window

of time steps. This would also be a useful feature in the work drive research, since one of the goals was to have modular perception architectures. By using the aforementioned capability, the measurement models can be adapted for a new (sub)set of sensors, resulting into a stable performance for different sensor sets and therefore increasing the level of modularity.

As discussed in Chapter 5, the implemented fault detection schemes can detect localization faults, but for decreasing magnitude, especially for faults $f_k^{loc} < \frac{3}{4}(3\sigma)$ the detection rate drops significantly, and comes at the cost of a higher false alarm rate. In [52] a model-based cumulative sum (also referred to as CUSUM) procedure is presented, that is able to detect small persistent faults.

Bibliography

[1] (), [Online]. Available: https://geodis.com/se/en/blog/technology-automation/autonomous-vehicles-and-collaborative-robots-solutions-warehouses-today.

[2] F. Eggers and F. Eggers, "Drivers of autonomous vehicles—analyzing consumer preferences for self-driving car brand extensions," *Marketing Letters*, vol. 33, Mar. 2022. DOI: 10.1007/s11002-021-09571-x.

[3] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[4] S. Ulbrich, A. Reschka, J. Rieken, *et al.*, "Towards a functional system architecture for automated vehicles," Mar. 2017.

[5] R. Isermann, *Fault-Diagnosis Systems From Fault Detection to Fault Tolerance*. Jan. 2006, vol. 28, ISBN: 978-3-540-24112-6. DOI: 10.1007/3-540-30368-5.

[6] D. Miljković, "Fault detection methods: A literature survey.," May 2011, pp. 750–755.

[7] Flanders Make, "Work-drive SBO project proposal, Modular, autonomous work-drive architectures," Flanders Make, Tech. Rep., 2020.

[8] E. Kýlýç, "Fault detection and diagnosis in nonlinear dynamical systems," Jan. 2005.

[9] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, "Multiple object tracking: A literature review," *Artificial Intelligence*, vol. 293, p. 103 448, 2021, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2020.103448. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370220301958.

[10] "Fault-detection, fault-isolation and recovery (fdir) techniques," NASA, Tech. Rep., 2006. [Online]. Available: https://extapps.ksc.nasa.gov/Reliability/Documents/Preferred_Practices/dfe7.pdf.

[11] Y. Kosuge and T. Matsuzaki, "The optimum gate shape and threshold for target tracking," in *SICE 2003 Annual Conference (IEEE Cat. No.03TH8734)*, vol. 2, 2003, 2152–2157 Vol.2.

[12] D. B. Reid, "An algorithm for tracking multiple targets," *IEEE Transactions on Automatic Control*, vol. 24, no. 6, pp. 843–854, 1979.

[13] P. Konstantinova, A. Udvarev, and T. Semerdjiev, "A study of a target tracking algorithm using global nearest neighbor approach," Jan. 2003. DOI: 10.1145/973620.973668.

[14] T. Fortmann, Y. Bar-Shalom, and M. Scheffe, "Sonar tracking of multiple targets using joint probabilistic data association," *IEEE Journal of Oceanic Engineering*, vol. 8, no. 3, pp. 173–184, 1983. DOI: 10.1109/JOE.1983.1145560.

[15] T. D. Laet, *Rigorously Bayesian Multitarget Tracking and Localization*. Leuven, België: Katholieke Universiteit Leuven – Faculty of Engineering, 2010, ISBN: 978-94-6018-209-9.

[16] P. Lorczak, A. Caglayan, and D. Eckhardt, "A theoretical investigation of generalized voters for redundant systems," in *[1989] The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*, (Jun. 23–21, 1989), Chicago, IL, USA: IEEE, 1989, pp. 444–451.

[17] G. Latif-Shabgahi, "A novel algorithm for weighted average voting used in fault tolerant computing systems," *Microprocessors and Microsystems*, vol. 28, no. 7, pp. 357–361, 2004, ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2004.02.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933104000134.

[18] M. Berk, O. Schubert, H.-M. Kroll, B. Buschardt, and D. Straub, "Assessing the safety of environment perception in automated driving vehicles," *SAE International Journal of Transportation Safety*, vol. 8, no. 1, pp. 49–74, 2020, ISSN: 23275626, 23275634. [Online]. Available: https://www.jstor.org/stable/27034112 (visited on 04/11/2022).

[19] S. Grubmüller, G. Stettinger, M. Á. Sotelo, and D. Watzenig, "Fault-tolerant environmental perception architecture for robust automated driving," in *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, Graz, Austria: IEEE, 2020, pp. 1–6.

[20] B. Brumback and M. Srinath, "A chi-square test for fault-detection in kalman filters," *IEEE Transactions on Automatic Control*, vol. 32, no. 6, pp. 552–554, 1987. DOI: 10.1109/TAC.1987.1104658.

[21] M. Realpe, B. Vintimilla, and L. Vlacic, "Towards fault tolerant perception for autonomous vehicles: Local fusion," in *2015 IEEE 7th International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, Jul. 2015, pp. 253–258. DOI: 10.1109/ICCIS.2015.7274630.

[22] ——, "Sensor fault detection and diagnosis for autonomous vehicles," in *2015 the 4th International Conference on Material Science and Engineering Technology (ICMSET 2015)*, (Oct. 28–26, 2015), Singapore: EDP Sciences, 2015, pp. 253–258. DOI: https://doi.org/10.1051/matecconf/20153004003.

[23] M. Realpe, B. X. Vintimilla, and L. Vlacic, "A fault tolerant perception system for autonomous vehicles," in *2016 35th Chinese Control Conference (CCC)*, IEEE, 2016, pp. 6531–6536. DOI: 10.1109/ChiCC.2016.7554385.

[24] Y. Hu, "Fault diagnosis for the perception software in highly automated vehicles and reference generation based on world model," *Annual Conference of the PHM Society*, vol. 12, p. 11, Nov. 2020. DOI: 10.36001/phmconf.2020.v12i1.1142.

[25] J.-P. Ramirez-Paredes, E. A. Doucette, J. W. Curtis, and V. Ayala-Ramirez, "Sensor compromise detection in multiple-target tracking systems," *Sensors*, vol. 18, no. 2, 2018, ISSN: 1424-8220. DOI: 10.3390/s18020638. [Online]. Available: https://www.mdpi.com/1424-8220/18/2/638.

[26] D. Schuhmacher, B.-T. Vo, and B.-N. Vo, "A consistent metric for performance evaluation of multi-object filters," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3447–3457, 2008. DOI: 10.1109/TSP.2008.920469.

[27] O. Bosgra, *4CM40 Physical Modelling chapter 1*. Eindhoven University of Technology, 2008.

[28] S. Koskie, *Discrete-Time Control Systems*, ser. ECE 595 Discrete-Time Control Systems. Indiana University — Purdue University Indianapolis, 2005. [Online]. Available: https://users.wpi.edu/~zli11/teaching/rbe595_2017/LectureSlide_PDF/discretization.pdf.

[29] S. Särkkä, *Bayesian Filtering and Smoothing*, ser. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013. DOI: 10.1017/CBO9781139344203.

[30] Q. Li, L. Ranyang, K. Ji, and W. Dai, "Kalman filter and its application," Nov. 2015, pp. 74–77. DOI: 10.1109/ICINIS.2015.35.

[31] D.-T. Tran, T. Huynh, N. Thang, T. Nguyen, and N. Chuc, "Designing kalman filters for integration of inertial navigation system and global positioning system," Jan. 2006, pp. 6–7.

[32] G. Pasricha, "Kalman filter and its economic applications," University Library of Munich, Germany, MPRA Paper, 2006. [Online]. Available: https://EconPapers.repec.org/RePEc:pra:mprapa:22734.

[33] K. Fronckova and A. Slaby, "Kalman filter employment in image processing," in *Computational Science and Its Applications – ICCSA 2020*, O. Gervasi, B. Murgante, S. Misra, *et al.*, Eds., Cham: Springer International Publishing, 2020, pp. 833–844, ISBN: 978-3-030-58799-4.

[34] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[35] J. Humpherys, P. Redd, and J. West, "A fresh look at the kalman filter," *SIAM Review*, vol. 54, no. 4, pp. 801–823, 2012. DOI: 10.1137/100799666. eprint: https://doi.org/10.1137/100799666. [Online]. Available: https://doi.org/10.1137/100799666.

[36] C. Wen, Y. Cai, C. Wen, and X. Xu, "Optimal sequential kalman filtering with cross-correlated measurement noises," *Aerospace Science and Technology*, vol. 26, no. 1, pp. 153–159, 2013, ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2012.02.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963812000557.

[37] P. Mahalanobis, "On the generalized distance in statistics," *Proceedings of the National Institute of Science of India*, vol. 2, pp. 49–55, 1936.

[38] H. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistic Quarterly*, vol. 2, May 2012.

[39] MathWorks. "Matchpairs, solve linear assignment problem." (), [Online]. Available: https://nl.mathworks.com/help/matlab/ref/matchpairs.html.

[40] I. S. Duff and J. Koster, "On algorithms for permuting large entries to the diagonal of a sparse matrix," *SIAM J. Matrix Anal. Appl.*, vol. 22, pp. 973–996, 2001.

[41] E. Hyun and J.-H. Lee, "Multi-target tracking scheme using a track management table for automotive radar systems," in *2016 17th International Radar Symposium (IRS)*, 2016, pp. 1–5. DOI: 10.1109/IRS.2016.7497283.

[42] G. Latif-Shabgahi and A. Hirst, "A fuzzy voting scheme for hardware and software fault tolerant systems," *Fuzzy Sets and Systems*, vol. 150, no. 3, pp. 579–598, 2005, ISSN: 0165-0114. DOI: https://doi.org/10.1016/j.fss.2004.02.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165011404000831.

[43] A. Schörgendorfer and W. Elmenreich, "Extended confidence-weighted averaging in sensor fusion," Jan. 2006.

[44] H. H. Andersen, *Linear and Graphical Models for the Multivariate Complex Normal Distribution*, ser. Lecture Notes in Statistics 101. New York: Springer-Verlag, 1995, ISBN: 978-0-387-94521-7.

[45] D. Mori, H. Sugiura, and Y. Hattori, "Adaptive sensor fault detection and isolation using unscented kalman filter for vehicle positioning," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, (Oct. 27–30, 2019), Auckland, New Zealand: IEEE, 2019, pp. 1298–1304.

[46] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," Sep. 2011.

[47] R. Patton and J. Chen, "Observer-based fault detection and isolation: Robustness and applications," *Control Engineering Practice*, vol. 5, no. 5, pp. 671–682, 1997, ISSN: 0967-0661. DOI: https://doi.org/10.1016/S0967-0661(97)00049-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S096706619700049X.

[48] M. Abid, "Fault detection in nonlinear systems: An observer-based approach," Feb. 2022.

[49] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, vol. 17, no. 1, pp. 168–192, 2021. [Online]. Available: https://doi.org/10.1016/j.aci.2018.08.003.

[50] F. Cazes, M. Chabert, C. Mailhes, *et al.*, "Flight control system improvement based on a software sensor derived from partial least squares algorithm," *IFAC Proceedings Volumes*, vol. 45, no. 20, pp. 1041–1046, 2012, 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20120829-3-MX-2028.00114. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016348911.

[51] H. Blom and Y. Bar-Shalom, "The interacting multiple model algorithm for systems with markovian switching coefficients," *IEEE Transactions on Automatic Control*, vol. 33, no. 8, pp. 780–783, 1988. DOI: 10.1109/9.1299.

[52] C. Murguia and J. Ruths, "Cusum and chi-squared attack detection of compromised sensors," in *2016 IEEE Conference on Control Applications (CCA)*, 2016, pp. 474–480. DOI: 10.1109/CCA.2016.7587875.

[53] B. Southall, B. Buxton, and J. Marchant, "Controllability and observability: Tools for kalman filter design," Jan. 1998. DOI: 10.5244/C.12.17.

Appendix A

System description

A.1 Assumptions

In the following, a list of assumptions and constraints used in the project are listed. These are used to model the system, environment and to develop fault detection schemes. The following constraints are defined.

- The number of objects N_k in the world is not known to the perception system. .
- Objects in the environment are static.
- The field of view (FoV) per sensor is limited.
- Existence and localization faults can occur at any time instant and at any number.
- The sensor readings are fused centrally.
- Process noise is set to zero.

The following sensor model assumptions are defined.

- It is unknown what sensor reading belongs to what object in the real world (ground truth).
- Measurements received by a sensor, are point objects in a two-dimensional plane, with respect to the global coordinate system.
- Measurement noises are independent, i.e. there is no cross-correlation. Measurement noise can be described with a multivariate normal distribution.
- An object in the world can cause at most 1 sensor reading, and every sensor reading originates from at most 1 object.

A.2 Perception system description

Figure A.1 shows a detailed diagram of the high-level system architecture of the considered platform perception system. Reliable environment perception is needed to navigate the environment safely. The output of the data fusion is sent to the path planner to decide on what actions to take to warrant such safety. The host state estimation provides the object observers with state values of the platform, that is needed for a transformation to the global coordinate system. The data fusion consists of the following steps, after it receives an object observation from a perception sensor. First, a prediction step is performed on the states of all objects that exist in the current status of the world model. Then, in the data association step the noisy smart sensor readings are associated to the predicted states and finally an update step is performed.

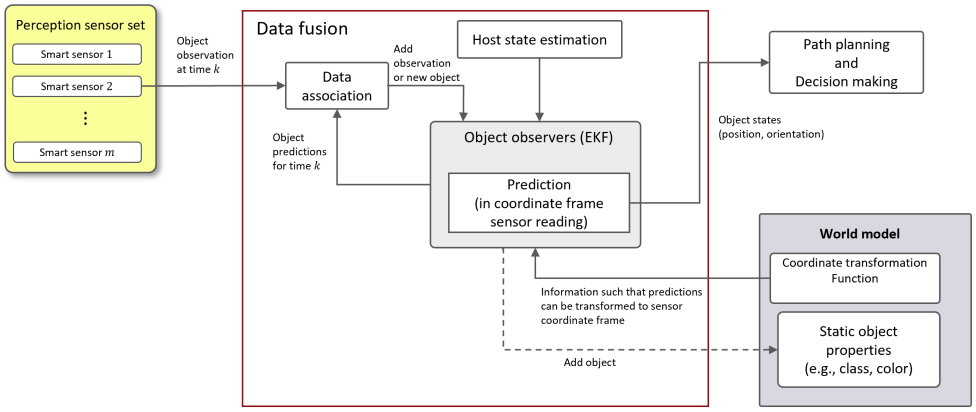

Figure A.1: High level system architecture.

Figures A.2 and A.3 show in two steps how the data fusion module process an object observation. The figures are based on Figure A.1, but some parts are omitted for brevity. Each node in the world model is associated to its own Extended Kalman Filter (EKF), and this EKF can be used to predict the state of the node with respect to its parent at any given timestamp, using all observations that are currently stored in memory and associated to this specific object. The following notation is introduced: y_k is the measurement value from a smart sensor at timestamp k, $\hat{x}^{j_w}_{k,\text{trans}}$ and $\hat{\mathbf{P}}^{j_w}_{k,\text{trans}}$ are the predicted state and covariance values of objects for timestamp k in the reference frame of the sensor belonging to the received sensor reading (indicated with subscript 'trans'), with $j_w \in \{1, \ldots, n\}$, where n is the number of previously objects.

As shown in Figure A.2, after a detection arrives, the state values of all n objects that currently exist in the world model are predicted in the same coordinate frame as the measurement for timestamp k. With help of the world model, it is possible to transform the coordinates of $\hat{x}^{j_w}_k$, $\hat{\mathbf{P}}^{j_w}_k$ into the same coordinate frame as the sensor measurement. The data association block verifies whether the detected object's state falls within one of the prediction's covariance ellipsoids. If this is true, the detection is associated to the closest prediction point, measured with the Mahalanobis distance [37]. Figure A.5 shows an exemplary situation, wherein detection y_k falls within both the covariance ellipsoids of predictions $\hat{x}^1_{k,\text{trans}}$ and $\hat{x}^2_{k,\text{trans}}$, but is associated to prediction $\hat{x}^1_{k,\text{trans}}$ since it has the smallest Mahalanobis distance with y_k. In Figure A.4, the other situation can be observed: detection y_k is not associated with any prediction $\hat{x}^{j_w}_{k,\text{trans}}$.

Figure A.3 shows how the detection y_k is subsequently added to one of the observers, or initialized as a new object. The data association blocks informs the switch that either the object observation is associated with a prediction, or a newborn object. The switch sends the observation y_k to the correct block. If the data association check succeeds, the observation is added to the EKF of the object with the closest $\hat{x}^{j_w}_{k,\text{trans}}$, $\mathbf{P}^{j_w}_{k,\text{trans}}$. If the check fails, a new object $n+1$ is added to the world model, and a new EKF is constructed for which the initial state is computed based on the detected object state. In Figure A.3 the situation of Figure A.5 is depicted, i.e., detection y_k is associated with prediction $\hat{x}^1_k$, and thus the detection y_k is added to *EKF object 1*.

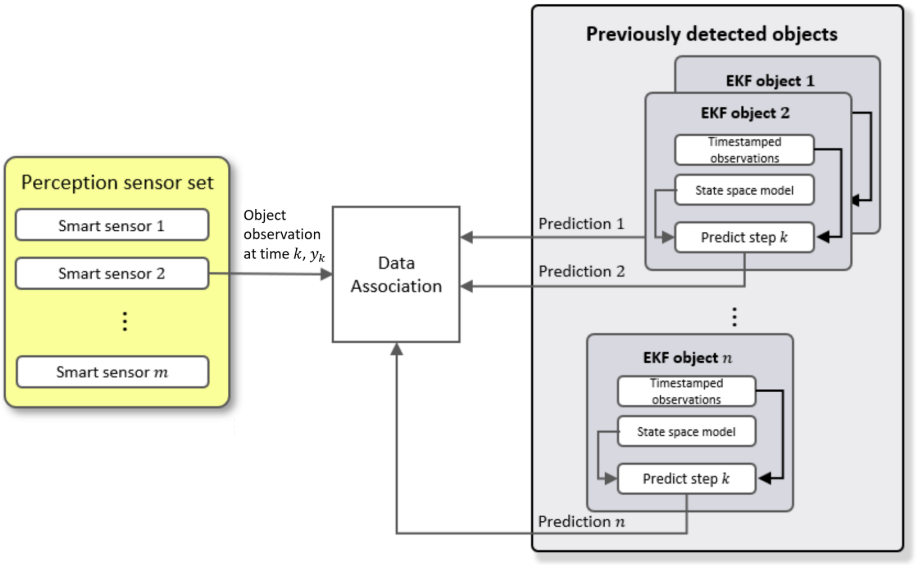

Figure A.2: Inputs for data association.

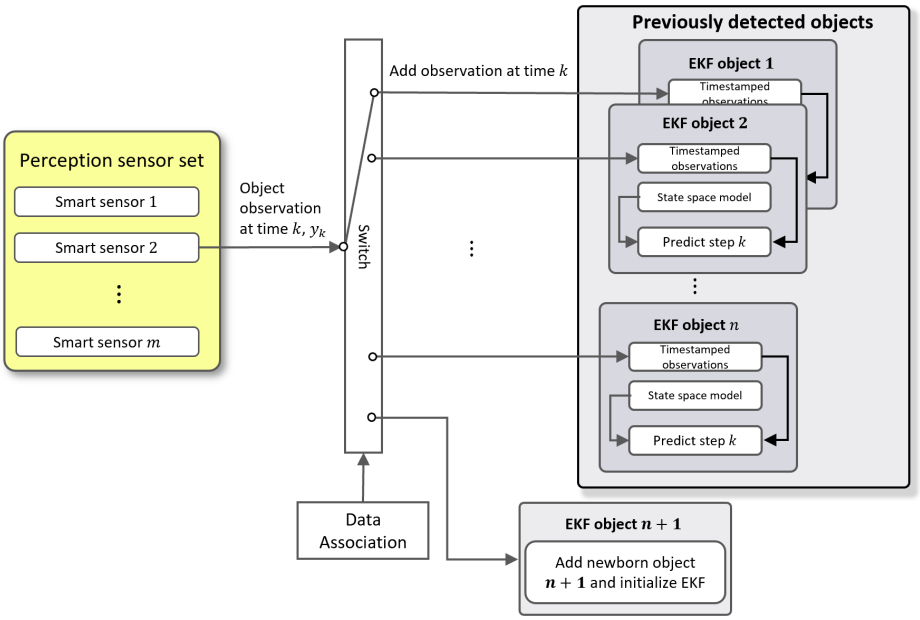

Figure A.3: Output of data association, based on situation 2, see Figure A.5.

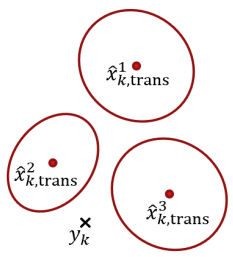

Figure A.4: Data association, situation 1.

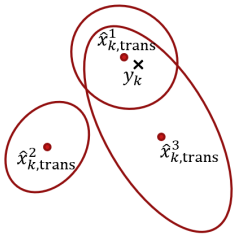

Figure A.5: Data association, situation 2.

Appendix B

Performance analysis

B.1 Redundancy based fault detection

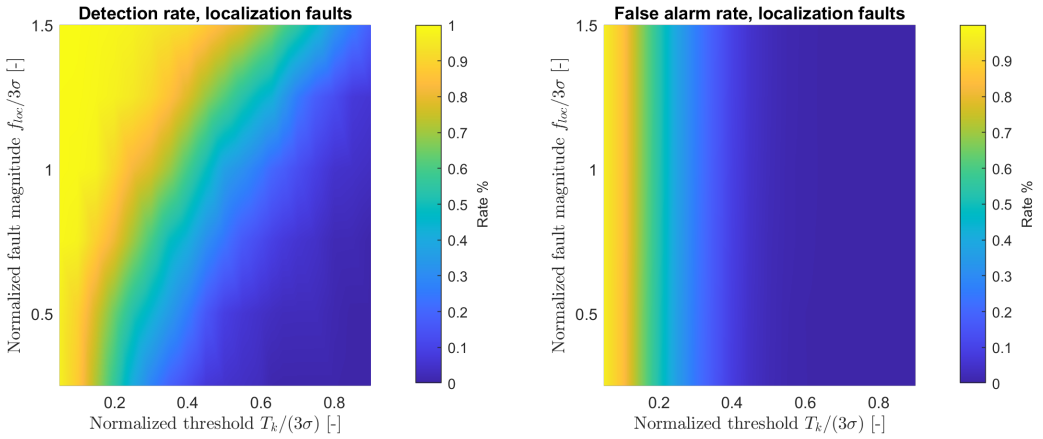

Figure B.1

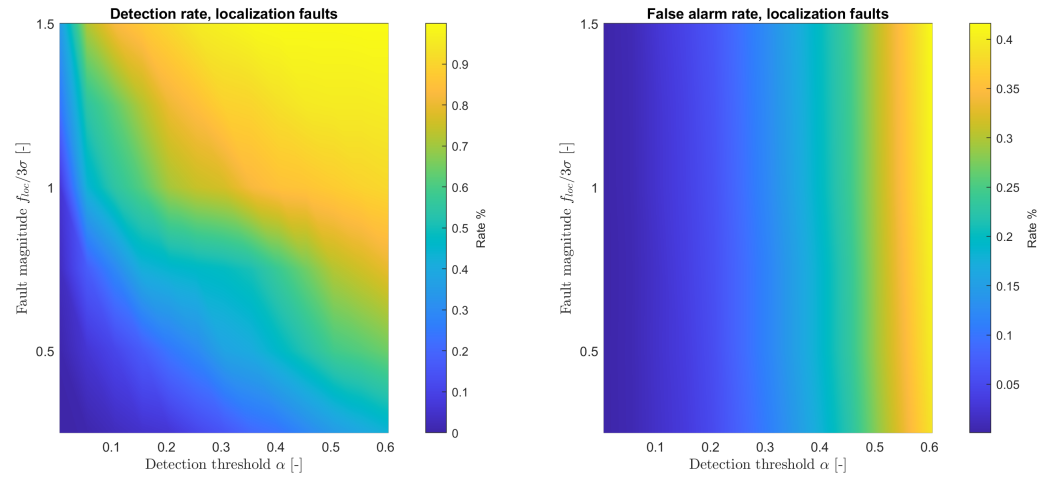

Figure B.2

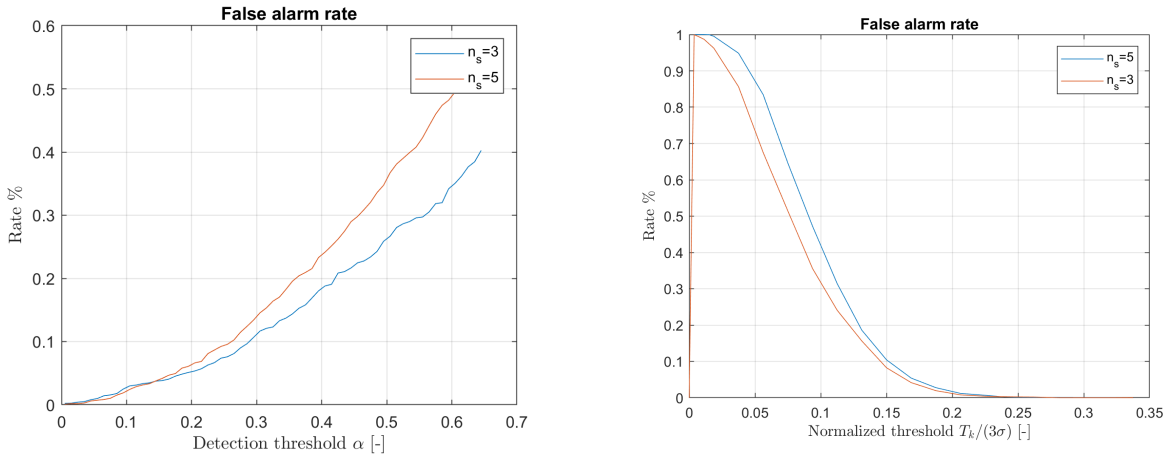

Figure B.3: Problem setting considered for performance analysis clustering algorithm.

B.2 Performance model based fault detection

Figure B.4 shows the sensor readings for several sample of two objects in a two dimensional plane, with $n_s = 2$ sensors. A localization fault is injected in sensor 1, causing an offset with constant magnitude but random angle. Furthermore, the gating regions used for data association are plotted.

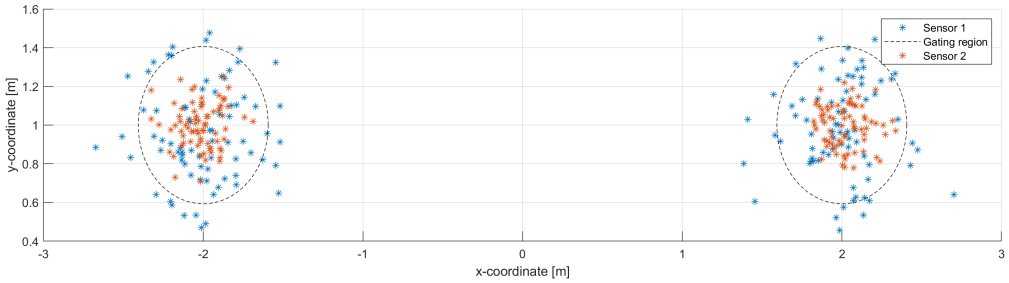

Figure B.4

Appendix C

Other

C.1 Observability

A system is observable if it is possible to reconstruct the internal states x_k of the system at time step k, based on the observed (a measurement of the system) space z_k at time step k. For discrete time linear time-invariant systems, the test for observability is defined as follows.

If the observability matrix

$$\begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

has row rank n, with n the dimension of the state vector x_k, then the system is observable [53].

C.2 Sum of normal distributions

Assume two multivariate normal random variables $X \sim \mathcal{N}(\mu_X, \Sigma_X)$ and $Y \sim \mathcal{N}(\mu_Y, \Sigma_Y)$. The expression of the characteristic function for X is:

$$\phi_X(u) = \exp\left(iu^T\mu_X - \frac{1}{2}u^T\Sigma_X u\right).$$

A property of the characteristic function is that the sum of two independent random variables is equal to the product of their respective characteristic functions, hence $\phi_X\phi_Y = \phi_{X+Y}$. This can be derived as follows:

$$\begin{aligned} \phi_{X+Y}(u) &= \exp\left(iu^T\mu_X - \frac{1}{2}u^T\Sigma_X u\right)\exp\left(iu^T\mu_Y - \frac{1}{2}u^T\Sigma_Y u\right). \\ &= \exp\left(iu^T\mu_X - \frac{1}{2}u^T\Sigma_X u + iu^T\mu_Y - \frac{1}{2}u^T\Sigma_Y u\right) \\ &= \exp\left(iu^T(\mu_X + \mu_Y) - \frac{1}{2}u^T(\Sigma_X + \Sigma_Y)u\right). \end{aligned} \tag{C.1}$$

Therefore we find that $X + Y \sim \mathcal{N}(\mu_x + \mu_Y, \Sigma_X + \Sigma_Y)$.