Specialization: Transportation Engineering and Logistics
Report number: 2017.TEL.8163
Title: Collision Detection for Automated Guided Vehicles
Author: T.D. Hogenboom

Title (in Dutch) Botsingsdetectie voor geautomatiseerde voertuigen

Assignment: research
Confidential: no
Supervisor: ir. M.B. Duinkerken
Date: August 31, 2017

This report consists of 44 pages and 3 appendices. It may only be reproduced literally and as a whole. For commercial purposes only with written authorization of Delft University of Technology. Requests for consult are only taken into consideration under the condition that the applicant denies all legal rights on liabilities concerning the contents of the advice.
Subject: AGV laboratory

The section Transport Engineering and Logistics runs a laboratory for the study of intelligent transport systems. In this laboratory, automated transport systems utilising automated guided vehicles (AGVs) are studied. These AGVs are scale models (1:25), based on the AGVs used at the container terminals in Rotterdam. Each AGV is equipped with servo's for speed control and (independent) control of both wheel axes and an Arduino controller board.

In the laboratory students can do research assignments for theme 3 related topics; f.i. coordinated or cooperative routing of vehicles, interaction between transport and handling equipment, advanced vehicle and fleet controllers etc.. The list of possible topics is changing frequently based on the progress made by other students and new insights gained. If you are interested contact me directly.

Studying relevant literature, developing and implementing a model, verification and validation of the model, experimenting with AGVs in the laboratory, presenting solid conclusions and recommendations and reporting the research work are all part of this assignment.

The report should comply with the guidelines of the section. Details can be found on the website. For more information, contact M.B. Duinkerken (34B-3-320; m.b.duinkerken@tudelft.nl).

The supervisor,

M.B. Duinkerken
Summary

A vehicle collision is the (usually unwanted) event of contact between multiple vehicles and/or a vehicle with another object from its surroundings (e.g. a static obstacle or a pedestrian) [1]. Conflict Detection and Resolution (CDR) systems are used to (semi-)automatically avoid or mitigate the risks of an imminent vehicle collision. As human drivers have limited capabilities when it comes to collision avoidance and automated vehicle systems become ever more present, the need for CDR systems increases.

At the Delft University of Technology, the section of Transportation Engineering and Logistics operates an Automated Guided Vehicle laboratory (AGV lab) where model AGVs are used to test advancements in AGV technology and scientific research in a controlled environment. The model AGVs used are 1:25 scale models of the AGVs that are commonly found on container terminals. An OptiTrack Motion capturing system is available in the AGV lab to track entities across a dedicated testing area.

The main goal of this research was to develop and test a system for detecting imminent collisions of model AGVs in real-time by predicting future vehicle states based on observed data through an OptiTrack motion capturing system. The focus hereby was on detecting imminent vehicle-to-vehicle (V2V) collisions between model AGVs using observable information and basic vehicle specifications (e.g. length, width) only.

A literature survey on CDR has been done such that well considered decisions could be made in the design of a CDR system for the model AGVs. The literature survey showed that a CDR system consists of the following four main building blocks:

1. **State estimation**
   First the current state (e.g. position, speed) of every vehicle and objects in its surroundings needs to be determined. Uncertainty is often present in state estimation due to limited, noisy or erroneous sensor data [2].

2. **State propagation**
   Using the current state, a prediction can be made of how this situation will propagate in the future. This can be done using physics-based, maneuver-based and/or interaction-aware models [3]. Physics-based models link future vehicle states to control inputs, vehicle properties and external conditions by using dynamic and/or kinematic evolution models. The simplicity of physics-based models makes them well suited for real-time applications but the accuracy is only sufficient for short-term predictions (up to a few seconds). Maneuver-based models are more suited for long-term prediction as they predict the intended maneuver of a vehicle based on training data. Interaction-aware models also take into account the interactions between entities and are therefore the most advanced type of models. Interaction-aware models are however not yet suited for real-time applications.

3. **Conflict detection and risk assessment:**
   After the future states of the vehicles and the objects in their surroundings are computed, conflicts can be detected and the risks of those can be assessed. Conflicts can be detected binary or probabilistic using a variety of methods. When a conflict is detected, additional risk indicators such as the Time-To-Collision or the vehicle speed (i.e.
high-speed collisions usually imply a higher risk than low-speed collisions) may be used to indicate the risk of the upcoming conflict.

4. Conflict resolution
   When a conflict has been detected with an unacceptably high risk, some measures need to be taken to avoid or mitigate it. Conflict resolution can be done prescribed, optimized, using a force field or manual. It may however also be decided to do nothing and thus neglect the risk.

In the AGV lab, a CDR system for the model AGVs has been developed and implemented using MATLAB. The remainder of this section will describe the design and the performance of this system as it is implemented.

State Estimation
For state estimation the x,z location and yaw of every model AGV are directly read-out from OptiTrack. This method was found to be sufficient for use in the AGV lab with a frame rate between 5 and 10 frames per second (FPS). Below 5FPS the actual trajectory is insufficiently reflected while above 10FPS the variance/noise in the data becomes excessive. Although the direct read-out method was sufficient during this research, it is probably not suited for practical applications outside of a laboratory. It is therefore recommended to incorporate a more advanced method for state estimation in the future based on for example Kalman filtering.

State propagation
A trajectory prediction algorithm based on a constant velocity model uses the estimated states to predict future trajectories for each model AGV. With this physics-based, non-probabilistic, single-trajectory model, the current velocity vector is extrapolated as a straight trajectory every time a new state has been estimated. These predicted trajectories are discretized into $i$ steps with a predefined time interval $T_{predI}$ and continue until the prediction horizon ($T_{predH}$) is reached (i.e. $i \times T_{predI} = T_{predH}$).

Through a number of experiments it was shown that the performance of the trajectory prediction algorithm based on the constant velocity model only generates useful results for straight trajectories with a constant velocity or on arbitrary trajectories with a prediction horizon of less than one second. These results were sufficient to test the CDR functionality in the AGV lab. For more advanced (and practical) applications it is however recommended to test a more advanced trajectory prediction algorithm based on for example a Kalman filter with a bicycle model.

Conflict Detection and Resolution
For conflict detection a non-probabilistic binary method is chosen with the Time-To-Collision (TTC) as the only risk indicator. AGVs are represented by a rectangular protected zone of the length and width of an AGV. Two AGVs are considered to be in conflict when their protected zones overlap and the $TTC \leq TTC_{limit}$ (where the $TTC_{limit}$ is a predefined limit for the Time-To-Collision). Overlap between every pair of protected zones is checked using the Separating Axes Theorem (SAT) on every i-th step of the discretized future trajectories. If a conflict between two AGVs is detected, a conflict resolution script is called that stops both AGVs.

A number of driving scenarios were executed to evaluate the performance of the CDR system in total. These scenarios were set up in such a way that two AGVs would collide without
CDR functionality or would just miss each other. The CDR system performed really well in all cases where the trajectory prediction algorithm was accurate (i.e. on straight trajectories with a constant velocity or on arbitrary trajectories with a prediction horizon of less than a second). The detection method based on the SAT worked flawless. A minimum \(TTC_{\text{limit}}\) of 0.4s was however found because AGVs cannot be stopped in time if it is set to a lower value. When the trajectory prediction algorithm was off (e.g. on curved trajectories) false alarms were detected.

A major drawback of testing the CDR functionality by manually setting up driving scenarios was that it was difficult to state statistically substantiated figures for the number of late/missed detections and the false alarm rate. Executing the number of variations and replications needed for this is simply too laborious and time-consuming. It is therefore recommended that a generic CDR testing module is made in the future from which these figures can be obtained.

Finally, it is worth mentioning that the conflict resolution method used during this research can only be used as an emergency stop because human intervention is needed to start up again. Although implementing an advanced resolution method wasn’t the goal of this research, it can make human intervention superfluous in the future.
# Table of Contents

Summary iii

1 Introduction 1
  1.1 Research background ................................................. 1
  1.2 Research goal and scope ........................................... 2

2 Literature survey on CDR 3
  2.1 Introduction .......................................................... 3
  2.2 State estimation ..................................................... 3
  2.3 State propagation ................................................... 3
    2.3.1 Physics-based models ........................................... 3
    2.3.2 Maneuver-based models ......................................... 5
    2.3.3 Interaction-aware models ....................................... 6
  2.4 Conflict detection and risk assessment ................................. 6
    2.4.1 Binary conflict detection ....................................... 6
    2.4.2 Probabilistic conflict detection ............................... 7
    2.4.3 Additional risk indicators ....................................... 7
    2.4.4 Conflict detection and risk assessment performance .......... 7
  2.5 Conflict Resolution .................................................. 8
  2.6 Conclusion ............................................................ 9

3 The AGV laboratory 11
  3.1 The AGV lab at a glance ............................................. 11
  3.2 The model AGVs used ............................................... 12
  3.3 The OptiTrack motion-capturing system .............................. 12
    3.3.1 Setup ........................................................... 12
    3.3.2 Accuracy ........................................................ 13

4 Design of the CDR system 15
  4.1 Introduction .......................................................... 15
  4.2 General design ....................................................... 15
  4.3 State estimation using Matlab and OptiTrack ......................... 16
    4.3.1 Method .......................................................... 16
    4.3.2 Implementation ............................................... 17
    4.3.3 Checking the accuracy .......................................... 17

Master of Science Research Assignment T.D. Hogenboom
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4 Trajectory Prediction Algorithm</td>
<td>18</td>
</tr>
<tr>
<td>4.4.1 Method</td>
<td>18</td>
</tr>
<tr>
<td>4.4.2 Implementation</td>
<td>19</td>
</tr>
<tr>
<td>4.5 Conflict detection</td>
<td>20</td>
</tr>
<tr>
<td>4.5.1 Method</td>
<td>20</td>
</tr>
<tr>
<td>4.5.2 Implementation</td>
<td>21</td>
</tr>
<tr>
<td>4.6 Conflict resolution</td>
<td>22</td>
</tr>
<tr>
<td>4.7 Conclusion</td>
<td>23</td>
</tr>
<tr>
<td>5 Experiments</td>
<td>25</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>25</td>
</tr>
<tr>
<td>5.2 State estimation performance</td>
<td>25</td>
</tr>
<tr>
<td>5.2.1 Experimental setup</td>
<td>25</td>
</tr>
<tr>
<td>5.2.2 Results</td>
<td>26</td>
</tr>
<tr>
<td>5.2.3 Evaluation</td>
<td>29</td>
</tr>
<tr>
<td>5.3 Trajectory prediction algorithm performance</td>
<td>29</td>
</tr>
<tr>
<td>5.3.1 Experimental setup</td>
<td>29</td>
</tr>
<tr>
<td>5.3.2 Results</td>
<td>30</td>
</tr>
<tr>
<td>5.3.3 Evaluation</td>
<td>32</td>
</tr>
<tr>
<td>5.4 Conflict Detection and Resolution performance</td>
<td>32</td>
</tr>
<tr>
<td>5.4.1 Experimental setup</td>
<td>32</td>
</tr>
<tr>
<td>5.4.2 Results</td>
<td>33</td>
</tr>
<tr>
<td>5.4.3 Evaluation</td>
<td>35</td>
</tr>
<tr>
<td>5.5 Conclusion</td>
<td>35</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>37</td>
</tr>
<tr>
<td>7 Recommendations</td>
<td>39</td>
</tr>
<tr>
<td>Bibliography</td>
<td>41</td>
</tr>
<tr>
<td>Glossary</td>
<td>43</td>
</tr>
<tr>
<td>List of Acronyms</td>
<td>43</td>
</tr>
<tr>
<td>List of Symbols</td>
<td>43</td>
</tr>
<tr>
<td>A Full experiment results</td>
<td>45</td>
</tr>
<tr>
<td>A.1 Results state estimation performance</td>
<td>46</td>
</tr>
<tr>
<td>A.2 Results trajectory prediction performance</td>
<td>49</td>
</tr>
<tr>
<td>A.3 Results CDR performance</td>
<td>52</td>
</tr>
<tr>
<td>B Driving scenarios</td>
<td>53</td>
</tr>
</tbody>
</table>
### C MATLAB code

<table>
<thead>
<tr>
<th>Section</th>
<th>File Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1</td>
<td>MAINPROGRAM.m</td>
<td>55</td>
</tr>
<tr>
<td>C.2</td>
<td>FrameReadyCallBack.m</td>
<td>57</td>
</tr>
<tr>
<td>C.3</td>
<td>StateEstimation.m</td>
<td>59</td>
</tr>
<tr>
<td>C.4</td>
<td>TrajectPrediction.m</td>
<td>61</td>
</tr>
<tr>
<td>C.5</td>
<td>ConflictDetection.m</td>
<td>63</td>
</tr>
<tr>
<td>C.6</td>
<td>CheckPolygOverlap.m</td>
<td>64</td>
</tr>
<tr>
<td>C.7</td>
<td>ConflictResolution.m</td>
<td>66</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Example of the conflict detection and resolution process in air traffic [2]</td>
</tr>
<tr>
<td>2.2</td>
<td>Physics-based state propagation models [3]</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of the AGV lab</td>
</tr>
<tr>
<td>3.2</td>
<td>One of the model AGVs used</td>
</tr>
<tr>
<td>3.3</td>
<td>Bottom of the same AGV</td>
</tr>
<tr>
<td>3.4</td>
<td>Camera setup and three model AGVs in Motive</td>
</tr>
<tr>
<td>4.1</td>
<td>Overall design of the CDR system</td>
</tr>
<tr>
<td>4.2</td>
<td>State estimation using OptiTrack</td>
</tr>
<tr>
<td>4.3</td>
<td>Checking OptiTrack accuracy using a setup of three AGVs</td>
</tr>
<tr>
<td>4.4</td>
<td>State propagation method used</td>
</tr>
<tr>
<td>4.5</td>
<td>Conflict detection method used</td>
</tr>
<tr>
<td>4.6</td>
<td>Example of how the SAT works [4]</td>
</tr>
<tr>
<td>5.1</td>
<td>Linearized trajectory using OptiTrack data</td>
</tr>
<tr>
<td>5.2</td>
<td>Example state estimation results: straight path 10FPS</td>
</tr>
<tr>
<td>5.3</td>
<td>Driving scenario 1</td>
</tr>
<tr>
<td>5.4</td>
<td>Mean results scenario 1</td>
</tr>
<tr>
<td>A.1</td>
<td>Velocity estimate driving straight for 12 seconds</td>
</tr>
<tr>
<td>A.2</td>
<td>Velocity estimate driving a circle clockwise for 15 seconds (front steering axis)</td>
</tr>
<tr>
<td>A.3</td>
<td>Velocity estimate driving a circle clockwise for 11 seconds (both steering axes)</td>
</tr>
<tr>
<td>A.4</td>
<td>Location prediction accuracy driving case 101</td>
</tr>
<tr>
<td>A.5</td>
<td>Location prediction accuracy driving case 102</td>
</tr>
<tr>
<td>A.6</td>
<td>Location prediction accuracy driving case 103</td>
</tr>
<tr>
<td>A.7</td>
<td>Yaw prediction accuracy driving case 101</td>
</tr>
<tr>
<td>A.8</td>
<td>Yaw prediction accuracy driving case 102</td>
</tr>
<tr>
<td>A.9</td>
<td>Yaw prediction accuracy driving case 103</td>
</tr>
<tr>
<td>B.1</td>
<td>Driving scenario 1</td>
</tr>
<tr>
<td>B.2</td>
<td>Driving scenario 2</td>
</tr>
<tr>
<td>B.3</td>
<td>Driving scenario 3</td>
</tr>
<tr>
<td>B.4</td>
<td>Driving scenario 4</td>
</tr>
</tbody>
</table>
# List of Tables

2.1 State propagation design options ........................................ 10
2.2 Conflict detection and risk assessment design options ............. 10

5.1 Experiments state estimation performance ............................. 26
5.2 Steady-state results speed estimation ................................. 28
5.3 Steady-state results angular velocity estimation ..................... 28
5.4 Experiments trajectory prediction accuracy ........................... 30
5.5 Mean steady-state trajectory prediction performance ............... 31
5.6 Remaining distance between AGVs scenario 1 ....................... 34
5.7 Overview results driving scenarios ................................. 35

A.1 Full data scenario 1 ..................................................... 52
Chapter 1

Introduction

1.1 Research background

A vehicle collision occurs when a vehicle comes into contact with another vehicle or with another object from its surroundings (e.g. a pedestrian or other obstacle) [1]. Vehicle collisions may result in injury, death and/or property damage and are therefore usually avoided. Collision avoidance can be accomplished through a Conflict Detection and Resolution (CDR) system. With an increasing level of automation in vehicles, the need for automated CDR systems increases. Whenever there is no human in-the-loop anymore to avoid or mitigate an imminent collision, reliable automated systems should take over these tasks. Also, even in vehicles where a person is still (mostly) in control, automated CDR systems can be used as an extra safety measure for the limited capabilities of the human driver.

Before a CDR system can intervene to avoid or mitigate an imminent collision, it must be detected that a collision is imminent first. I.e., a CDR system should be able to predict whether or not the current situation will lead to a collision in the future before it can take appropriate measures. Due to the inherent difficulty of predicting the future, uncertainty is ever present in CDR systems. A wide variety of methods can be chosen in the design of a CDR system with an even greater choice in sensors, models and algorithms. Up until today, no single method is known of that is capable of handling all different situations in a (near) optimal way. Different CDR systems may prove to be beneficial in certain types of situations and the robustness of the total system might be improved by backing-up CDR systems with others that use different methods.

This research focuses on the development of a vehicle-to-vehicle (V2V) CDR system through an external observer that uses only its own observations of the environment and basic vehicle specifications (e.g. length, width). This means that the CDR system is configured centrally such that it is able to measure and predict the state of a demarcated piece of environment and the entities (in this case vehicles) that are in there. Using its own observations implies that only observable information is used such as the position, speed and direction of the entities. Such a system operates on a minimum of information and doesn’t need to be provided with unobservable information such as the planned trajectories of entities for example. In this way the collision detection system can operate independently (i.e. irrespective of, for example, the planning and control systems used). Because the CDR is in this case an independent central system, only basic communication is needed between this system and the individual entities (e.g. a simple conflict resolution command is sent to a pair of entities). No communication is needed between entities themselves.

The section of Transportation Engineering and Logistics from the Delft University of Technology operates an Automated Guided Vehicle laboratory (AGV lab) where advancements in AGV technology and scientific research can be tested in a controlled environment using model
AGVs. A motion-capturing camera system called OptiTrack is available in the laboratory that can be used to track entities within the boundaries of a controlled area through a Matlab interface. These facilities will be used to demonstrate the capabilities and shortcomings of the system developed.

1.2 Research goal and scope

The main goal of this research is to develop and test a system for detecting imminent collisions of model AGVs in real-time by predicting future vehicle states based on observed data through the OptiTrack motion capturing system.

Which leads to the following main research question:

- What is a suitable design for a system that detects imminent collisions between model AGVs in real-time by predicting future vehicle states based upon observed data only; and how does it perform when implemented in the AGV lab?

To answer this main question, the following central research questions need to be answered:

1. What is the general setup of CDR systems and which design options exist for them?
2. How can the state (e.g. position, speed, orientation) of the model AGVs be estimated using OptiTrack and MATLAB?
3. How can the future states (e.g. future position, orientation) of the model AGVs be predicted using the estimated states?
4. How can imminent collisions between model AGVs be detected using the predicted states?
5. How does the developed CDR system perform when implemented in the AGV lab?

The model AGVs are traced in the laboratory using an OptiTrack motion capturing system. The accuracy of this system is taken as a given, there is no intention of improving this. In real systems other state-estimation techniques might be used. One example is the use of GPS combined with odometry/dead reckoning. Furthermore the following demarcations are set for this research:

- The focus will be on V2V collisions between ground-based model AGVs. This work can be extended in the future to take into account collisions between AGVs and other types of entities such as static obstacles and pedestrians.
- The collision detection system operates as an independent system and uses only observable information and basic vehicle specifications (e.g. length, width).
- Resolution of detected imminent collisions will not be implemented during this research other than very simple solutions (e.g. stop the AGVs) that are required to successfully execute experiments.
Chapter 2

Literature survey on CDR

2.1 Introduction

From the moment that a collision is considered imminent (i.e. there is a certain risk of collision), the concerning vehicle is in a conflict. In air traffic for example, aircraft are in conflict when they experience a loss of minimum separation [2]. Conflict detection and resolution (CDR) systems are used for avoidance or mitigation of conflicts (and therewith collisions) [2]. In figure 2.1, an example is presented of the processes that play a role in CDR systems. Various design options exist for these systems, all with their own characteristics. This chapter therefore aims to answer the first central research question: what is the general setup of CDR systems and which design options exist for them? The findings in this chapter lay the foundation for a well-considered design of the CDR system to implement in the AGV lab.

2.2 State estimation

The first step in a CDR system is to estimate the current states (e.g. position, speed, orientation) of the vehicle and objects in its environment. Uncertainties often remain in the state estimation because of limited, noisy or erroneous sensor data [2].

2.3 State propagation

After the state estimation, the next challenge in a CDR system is to predict how the current state will propagate in the future. Even under the assumption of a perfectly known current state, reliable (long term) predictions can be very hard to make due to the inherent difficulty of predicting the future [5]. Lefèvre et al. [3] have performed a survey on motion prediction and risk assessment for intelligent vehicles (mainly for road-use). They identified three categories of state propagation methods: physics-based models, maneuver-based models and interaction-aware models.

2.3.1 Physics-based models

The first and also the simplest types of models are the physics-based motion models. Physics-based models predict future motion by using dynamic and/or kinematic evolution models to link control inputs (e.g. steering), vehicle properties (e.g. weight) and external conditions (e.g. road surface friction) to the future vehicle state (e.g. position, speed, orientation) [3]. Depending on the way in which uncertainties are handled, different ways can be distinguished

---

1 Risk is considered as a combination of the likelihood that an event is going to happen and the severity of damage resulting from when the event happens [3].
in which the dynamic and kinematic evolution models can be used to predict future trajectories: single-trajectory simulation, Gaussian noise simulation and Monte Carlo simulation.

**Single-trajectory simulation**
Perhaps the most simplistic way of (physics-based) state propagation is to predict a single trajectory without considering any uncertainty in the state estimation or evolution model [3] (i.e. the predicted trajectory has a probability of one [2]). According to Lefevre et al. [3], single-trajectory simulation is, due to its simplicity, well suited for real-time applications. It is however limited to a very short prediction horizon (depending on the complexity/variability of the actual motion). A feasible prediction horizon is typically less than a second [3] or only a few seconds into the future [2]. Figure 2.2a illustrates the concept of single-trajectory simulation with a very simple constant velocity model. The future trajectory is predicted by simply extrapolating the current velocity vector in a straight line.

**Gaussian noise simulation**
Gaussian noise simulation takes into account the uncertainty in the state estimation by modeling it as a normal distribution [3]. Bayes filters, Kalman Filters (KFs) and the Interacting Multiple Model Kalman Filter (IMM-KF) are popular in state propagation with Gaussian noise simulation [3, 5]. Using these methods, it is possible to obtain the uncertainty in each predicted time step as shown in figure 2.2b. Uncertainties in the predicted trajectory caused
by future changes in the control inputs (i.e. performing a maneuver) or in the environment (e.g. obstacles) are however usually still not taken into account [3,5].

Monte Carlo simulation
With Monte Carlo simulations, future trajectories are generated by randomly sampling from the inputs (e.g. steering angle, acceleration) of the evolution model [3]. According to Lefevre et al. [3], unfeasible trajectories can be filtered out by penalizing trajectories that do not take into account environmental constraints and/or by taking the physical limits of the vehicle into account. Figure 2.2c shows the concept of trajectories predicted by Monte Carlo simulation.

Worst-case approach
A well-known example of a physics-based model is the worst-case model. With this model it is assumed that vehicles try to collide actively [5]. The worst-case model maps out all the positions that can be reached within a certain time horizon by varying all control inputs to their limits. Each physically feasible trajectory is thus considered with equal likelihood [2]. These characteristics make the worst-case approach very conservative [2,5]. Consequently, applicability is limited to a very short time-horizon due to the very high false alarm rate [5]. Also a significantly reduced overall traffic capacity can be expected because of this [2].

![Diagram of Physics-based state propagation models]

Figure 2.2: Physics-based state propagation models [3]

2.3.2 Maneuver-based models
Maneuver-based models assume that the motion of a vehicle corresponds to a series of maneuvers executed independently from other vehicles [3]. With maneuver based models, it is tried to recognize the intended maneuver of a vehicle. Future trajectories are then predicted assuming that the future motion will match the intended maneuver. For example the suggestion of Kuchar & Yang [2] to incorporate procedural information as a flight plan into an air traffic CDR system can be considered a maneuver-based method. Because of the a priori trajectories, maneuver-based models are more suited for long-term prediction than physics-based models [3]. Two approaches are distinguished by Lefevre et al. [3] in maneuver-based models: prototype trajectories and maneuver intention estimation and execution.
With prototype trajectories, it is assumed that clusters of typical motion patterns can be discovered in the environment. These clusters are learned through (observed) data after which trajectory prediction can be done by finding the most likely future maneuver from a partial trajectory that is executed so far. One drawback of using prototype trajectories is the training phase in which motion patterns are learned. Generating or observing the data needed takes time and effort and so there is little adaptability to changes in environment layout or changes in behavior.

With maneuver intention estimation and maneuver execution, the intended maneuver is estimated using various indicators. For example the current state of the vehicle (position, velocity, turn signals etc.), information about the environment (geometry, traffic rules etc.) and driving behavior (head movement, driving style etc.) can be used [3]. Using these higher-level characteristics for prediction makes it easier to generalize training data for arbitrary layouts and motion patterns. The assumption however that vehicles move independently of others doesn’t necessarily hold in practice.

2.3.3 Interaction-aware models

Interaction-aware models represent the interactions between vehicles and assume that the motion of a vehicle is influenced by the motion of other vehicles in its vicinity [3]. Interaction-aware models are the most advanced type of models. Just like the maneuver-based models, they are more suited for long-term predictions than the physics based models. Interaction-aware models are however more reliable for risk assessment than maneuver-based models in situations where vehicles don’t act as totally independent entities, but adapt their movement to the movement of other vehicles. The high computational requirements make them however (for now) unsuited for real-time conflict detection applications [3].

2.4 Conflict detection and risk assessment

With the predicted future states of the vehicle and the objects in its surroundings computed, it becomes possible to check for future conflicts and to assess the risks of those. Conflict detection can be binary (i.e. a conflict is either going to happen or not going to happen) or probabilistic (i.e. taking into account the uncertainty in the prediction of future motion) [3].

2.4.1 Binary conflict detection

When simple linear physics-based propagation models are used, binary conflict detection can be achieved by solving the intersection point of predicted trajectories from the model’s linear differential equations [3]. If the motion equations become too complex to solve in this way; a solution can be to approximate trajectories by piece-wise straight lines [3]. It is however more common to discretize the trajectories in time and to check whether a collision is bound to happen at each time-step [3]. This method fits well with the use of measured observations that are received with a certain framerate. A conflict can then be defined to happen if the distance between two points of different trajectories is below a certain threshold at a certain time-step. This simple method doesn’t however take the shape of the tracked vehicle(s)/objects into account. In order to do this, a shape (e.g. a polygon) can be used to represent the shape of the vehicle. Conflicts can then be detected by checking for overlap of multiple vehicle shapes at each time-step. This method corresponds with the concept of a protected zone (PZ) around the vehicle mentioned by Kuchar et al. [2]. With such a PZ,
the shape representing the vehicle doesn’t need to match the physical shape of the vehicle. A conflict is then defined as the moment the PZ is invaded. In this way the PZ is providing a safety buffer in the conflict detection method. The protective zone doesn’t necessarily have to be specified in units of distance, other units such as time may also be used [2].

2.4.2 Probabilistic conflict detection

Probabilistic conflict detection can, depending on the method of probabilistic state propagation, take place in different ways. One example given by Lefevre et al. [3] is the case in which a probability distribution is used over sample trajectories for the state propagation (e.g. in methods that rely on Monte Carlo simulations or Gaussian Processes). In this case the probability of a conflict can be computed by "integrating over all possible future trajectories and detecting collisions between each possible pair". According to Kuchar et al. [2] the advantage of probabilistic conflict detection is that decisions in conflict resolution can be based upon the likelihood of the event. Modeling the probabilities of different trajectories can however be difficult.

2.4.3 Additional risk indicators

Besides the detection of the conflict itself (and possibly its probability), many additional indicators can be used to indicate the risk of an upcoming conflict. Common examples of these are [3]:

- **Vehicle speed**
  High-speed collisions are generally more dangerous than low-speed collisions.

- **The amount of overlap between protected zones**
  The amount of overlap between protected zones in a prediction can be a measure for how bad a collision is going to be (i.e. a slight touch of a corner vs a full crash).

- **The configuration of the collision**
  The configuration of the collision is the setup in which the collision takes place. A full head-on collision might for example impose a greater risk than a rear-on collision. Another example can be found in the types of entities that are involved in the collision (e.g. a car-pedestrian vs a car-car collision).

- **Time-To-Collision\(^2\) (TTC)**
  As predictions often get better when they are less far in the future, collisions with a lower TTC often have a higher probability of happening. A lower TTC also means that there is less time (if any left) to avoid or mitigate the collision.

- **Time-To-React\(^3\) (TTR)**
  Imposes the same risk characteristics as the TTC but takes into account the possible measures that can be taken to avoid/mitigate the collision.

2.4.4 Conflict detection and risk assessment performance

As was seen from the previous sections, many ways exist in how predictions about the future state can be made, how a conflict can be detected and how risk assessment can take place. Despite all of these choices, the performance of a conflict detection system can, according to Kuchar et al. [2], be reviewed with two main key performance indicators (KPIs):

---

\(^2\)The TTC is the time left before the conflict/collision is predicted to happen

\(^3\)TTR is the time left in which the predicted conflict/collision can still be avoided
1. The number of late/missed\(^4\) detections
2. The false alarm rate

Just about all other performance indicators can be redirected to just these two. The speed in which the algorithms for prediction and detection execute can for example be a helpful indicator in designing a CDR system; but in the end, all that really matters is that the conflict is detected in time. Kuchar et al. don’t however consider additional risk assessment in CDR explicitly beyond the detection of a conflict. Therefore, the performance assessment is lacking if only these two KPIs are used while the quality of the risk assessment plays a role in the further processing of conflicts. An imminent collision might for example be detected, but if the risk is assessed too low, measures might be taken that are too mild. And conversely, when the risk is assessed too high, overly conservative measures might be taken. In such cases, the *quality of the risk assessment* would thus be the third main KPI.

### 2.5 Conflict Resolution

When an imminence conflict is detected, the risk can be avoided or mitigated by taking some actions, or it can be neglected. Conflict detection and resolution cannot always be considered separately [2]. When action should be taken may for example depend on which actions are taken; and conversely, the most appropriate action may depend on when the conflict is detected. The TTR (or other indicators related to whether a conflict is still avoidable or not) can also only be computed when it is known which actions are going to be taken.

Kuchar & Yang [2] distinguish the following five categories over which conflict resolution methods can be divided:

- **Prescribed**
  A standard, prespecified procedure is used to deal with the threat. A simple example is to just stop the vehicle(s) when a conflict is imminent. The procedure executed can also be prescribed to the context of the upcoming conflict. For example when a head-to-head collision is imminent between two vehicles, a procedure may be used in which both steer right (from a local perspective) to avoid a crash.
  Prescribed methods are usually simple, fast in execution and can easily be executed automated or by human operators. In general they are however less effective than methods that determine a solution in real-time; this is due to that procedures cannot be adapted to the specific situation. It might for example be possible that the open-loop actions create a new conflict.

- **Optimized**
  A cost function is defined and solved (typically with a kinematic model) such that the actions taken are sufficient to avoid the conflict but lead to the lowest costs. These costs can be expressed monetary but also in other units such as time or the aggressiveness of the maneuver. Typical techniques used for this are linear programming, genetic algorithms, game theory, expert systems and fuzzy control.

- **Force field**
  Force field methods model vehicles as a charged particle and use modified electrostatic

\(^4\)Late and missed detection events are considered the same thing; both result in the undesirable event of a conflict/collision actually happening.
equations to generate resolution maneuvers". These methods have a resolution continuously available but can be complicated because they may for example require additional processing or filtering.

- **Manual**
  Manual methods have a human in the loop to determine the appropriate actions. The algorithm thus only has an informative role and may also provide feedback about whether the intended actions are acceptable.

- **None**
  It might be possible that conflict detection is in place but resolutions are not offered. In this case an upcoming conflict is simply neglected and will occur if the predictions match the actual case.

### 2.6 Conclusion

This chapter aimed to answer the first central research question: what is the general setup of CDR systems and which design options exist for them?

It was found that CDR systems exist of the following four main building blocks and corresponding design options:

1. **State estimation**
   Can be accomplished through a variety of sensors. The OptiTrack motion capturing system will be used during this research. Other methods include for example the use of GPS combined with odometry.

2. **State propagation**
   An overview of the design options is presented in table 2.1.

3. **Conflict detection and risk assessment**:
   An overview of the design options is presented in table 2.2.

4. **Conflict resolution**
   Can be accomplished in the following ways:
   - Prescribed
   - Optimized
   - Using a force field
   - Manual
   - None (conflict neglected)
### Table 2.1: State propagation design options

<table>
<thead>
<tr>
<th>Model type</th>
<th>Physics-based</th>
<th>Maneuver-based</th>
<th>Interaction-aware</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Methods</strong></td>
<td>Single-trajectory</td>
<td>Prototype trajectories</td>
<td>Most advanced models</td>
</tr>
<tr>
<td></td>
<td>Probabilistic</td>
<td>Maneuver intention estimation and maneuver execution</td>
<td>Long-term prediction</td>
</tr>
<tr>
<td></td>
<td>Worst-case</td>
<td></td>
<td>Needs training data regarding regular patterns and/or maneuver intentions</td>
</tr>
<tr>
<td><strong>Characteristics</strong></td>
<td>Short-term prediction</td>
<td>Long-term prediction</td>
<td>Most advanced models</td>
</tr>
<tr>
<td></td>
<td>Strong real-time applicability</td>
<td></td>
<td>Long-term prediction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not (yet) suited for real-time applications</td>
</tr>
</tbody>
</table>

### Table 2.2: Conflict detection and risk assessment design options

<table>
<thead>
<tr>
<th>Method</th>
<th>Additional risk indicators</th>
<th>KPIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>For example:</td>
<td>Number of late/missed detections</td>
</tr>
<tr>
<td></td>
<td>Vehicle speed</td>
<td>False alarm rate</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>Configuration of the collision</td>
<td>Quality of risk assessment (if applicable)</td>
</tr>
<tr>
<td></td>
<td>Time-To-Collision (TTC)</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3

The AGV laboratory

As this research aims to implement a CDR system in a controlled environment, this short chapter presents an overview of the AGV laboratory and the equipment that is used.

3.1 The AGV lab at a glance

At the faculty of Mechanical, Maritime and Materials engineering from the Delft University of Technology, the section of Transportation Engineering and Logistics operates an Automated Guided Vehicle laboratory (AGV lab). This laboratory provides a controlled environment in which AGV technology and advancements in scientific research on this topic can be implemented and tested with scale model AGVs.

The AGV lab features a dedicated ground surface of around 5.6 by 4 meters over which moving objects can be tracked using an OptiTrack motion capturing camera system. This OptiTrack system is connected to a desktop PC dedicated to the AGV lab. This desktop PC is again connected to the Eduroam network such that communication with other PCs/laptops is also possible (however this connection wasn’t working due to firewall issues at the time of this research). A schematic representation of this setup is presented in the work of Bentvelsen [6]. The model AGVs used for this research are 1:25 scale models of the AGVs commonly used on container terminals. Figure 3.1 shows an overview of the AGV lab with the dedicated tracking area, the OptiTrack cameras, three model AGVs and the desktop PC used.

![Figure 3.1: Overview of the AGV lab](image-url)
3.2 The model AGVs used

The model AGVs used have a length of 0.62m and a width of 0.145m. They are equipped with two steering axes, five OptiTrack markers, a battery and an Arduino with an Xbee connected to it for wireless communication. The two steering axes are independently controllable through two servo motors while the forward/backward movement can also be controlled independently through another servo motor. Figures 3.2 & 3.3 show respectively the top and bottom of one of the AGVs. The other AGVs used during this research are identical to this one with the exception of the placement of the reflective markers and the color. A more detailed technical description of the model AGVs can be found in the work of Gerritse [7].

![Figure 3.2: One of the model AGVs used](image1)

![Figure 3.3: Bottom of the same AGV](image2)

3.3 The OptiTrack motion-capturing system

3.3.1 Setup

In the current AGV lab setup, twelve motion-capturing OptiTrack cameras of the Flex13 type are suspended around the dedicated tracking area. The OptiTrack camera system works with Motive software that runs on the dedicated AGV lab desktop PC. During this research Motive version 1.7.3 is used. Figure 3.4 shows a screen-shot from within Motive of how the cameras are positioned relative to the x,z ground plane. This x,z plane corresponds to the dedicated floor surface in the AGV lab and is the only plane of interest during this research (only 2D movement is considered).

In order to track the position and orientation of the model AGVs using OptiTrack, each model AGV is equipped with five infrared reflecting markers (see figure 3.2). Four of the markers are setup in a unique pattern such that the OptiTrack system is able to distinguish different model AGVs when multiple are used at the same time and no issues arise in determining the orientation. The fifth marker is placed on top of the center of the rear axis such that it provides a fixed reference point that is equal for each AGV.

For rigid bodies, Optitrack/Motive provides the measured data as x,y,z coordinates of a pivot point (in meters) and the rotation of the rigid body (expressed in quaternions) as opposed to a user-defined zero. Standard the pivot point is set by Motive at the centroid of the markers used to form a rigid body. In this research the pivot point is however manually put to the fifth marker that is located on the rear axis and thereafter displaced with Motive such that it is located in the centroid of the model AGV in the x,z plane.
3.3.2 Accuracy

Despite the careful placement of the markers and the setup and calibration of the OptiTrack system, some inaccuracies remain inevitable. Because placing the markers and displacing the pivot point is done manually, the pivot point and the actual centroid of the model AGV won’t coincide exactly. It is estimated however that this discrepancy is not more than a couple of mm, up to around one cm on the model AGVs used (bodies 1, 2 and 3 are used). For the yaw of the AGVs, also a 'point of zero' has to be set manually through Motive. This results in a slight discrepancy between the orientation of the OptiTrack grid and the actual orientation of the model AGV. For this discrepancy it is estimated that the yaw is not further off than a couple of degrees. Finally, the OptiTrack system itself also has a finite precision and doesn’t provide a continuous measurement. The setup with Flex13 cameras captures and sends data with 30-120 frames per second (depending on the frame rate the user sets in Motive). A maximum distance measurement error for a marker of 0.1 - 2mm can be expected [6]. The calibration of the current OptiTrack setup was done on 05-07-2017 with the following results:

- Overall reprojection mean 3D error = 0.575mm
- Worst camera mean 3D error = 0.536mm
- Triangulation recommended = 5.6mm
- Residual mean error = 0.6mm
- Overall wand error = 0.324mm
- Ray length suggested = 7.5m

So, based upon the estimated accuracy of the manual marker setup and the specified OptiTrack accuracy, the \([x,z]\) coordinates given by OptiTrack should not differ by more than 12mm from the actual \([x,z]\) location of the centroid of the AGV (which is probably already a wide estimate). The yaw is estimated not to be more off than 5mm. Currently there is no equipment available in the AGV lab that allows for a quantitative validation of these estimates with sufficient accuracy. A simple accuracy check is however performed and described in section 4.3.3.
4.1 Introduction

From chapter 2 it became clear how CDR systems work in general and which options there are in designing one. This chapter elaborates the design of a CDR system and the implementation thereof in the AGV laboratory. At first the general outline of the design will be presented in section 4.2. Thereafter, in respectively sections 4.3, 4.4 and 4.5, the following three research questions will be answered:

- How can the state (e.g. position, speed, orientation) of the model AGVs be estimated using OptiTrack and Matlab?
- How can the future states (e.g. future position, orientation) of the model AGVs be predicted using the estimated states?
- How can imminent collisions between model AGVs be detected using the predicted states?

4.2 General design

As became clear from the the previous chapter, a CDR system consists of a number of interacting main building blocks. The schematic representation of these main building blocks presented in figure 4.1 provides the overall design of the CDR system for the AGV lab.

Matlab implementation

The CDR system is implemented using Matlab for the purpose of testing its performance in a number of experiments. The main file for executing these experiments is MAINPROGRAM.m. MAINPROGRAM.m first loads the user-defined experiment parameters, sets up the connections with Motive and the Xbees, creates a location plot and sets up a frame-ready event listener.

The frame-ready event listener presents the main "loop" of the program. Every time a new frame of data is received from Motive, the listener calls the function FrameReadyCallBack. From within FrameReadyCallBack, the Motive frame rate is (if necessary) transformed into the desired data processing frame rate. Thereafter FrameReadyCallBack calls the functions for state estimation, state propagation and conflict detection and resolution (the implementation of these functions will be discussed in the upcoming sections).

Finally, when an experiment is over, the location figure is closed and MAINPROGRAM.m terminates the connections with Motive and the Xbees and prints the number of dropped and duplicate Motive frames to indicate possible problems with the experiment. The MATLAB code for MAINPROGRAM.m and FrameReadyCallBack.m can respectively be found in appendices C.1 and C.2.
4.3 State estimation using Matlab and OptiTrack

4.3.1 Method

As is displayed in figure 4.2, state estimation uses the OptiTrack observations to estimate the current state of all the model AGVs. As this research only considers 2D movement, the only OptiTrack state variables of interest are the x,z coordinates of the pivot point, the vehicle yaw and the time-stamp of the current frame of data. (Angular) velocity and acceleration can, if needed, be derived from this basic data when taking into account the time between two OptiTrack data frames.

Where:
- $X_{\text{cur}}$: Current X-coordinate [m]
- $Z_{\text{cur}}$: Current Z-coordinate [m]
- $\theta_{\text{cur}}$: Current yaw [rad]
- $T_{\text{cur}}$: Timestamp of the current frame of data [s]

During this research only the sensor data from the OptiTrack motion capturing system is used. The OptiTrack data is streamed to Matlab (via Motive) using a standard NatNet library. More information about this connection can be found in the work of Bentvelsen [6]. During this research Matlab r2014b is used with the NatNet SDK 2.7 library.
4.3.2 Implementation

The state estimation block is implemented in Matlab as the function StateEstimation and is called through the FrameReadyCallBack function.

**Code outline function StateEstimation**

**On** every new Motive frame of data; **for** every model AGV **do**:

- Retrieve the rigid body data from Motive
- Write the x,z coordinates of the centroid (pivot point) to the AGV data array
- Calculate the yaw, regarding singularities
- Write the yaw to the AGV data array
- Determine the x,z coordinates of the corner points of the rectangle that represents the AGV
- Write the corner point coordinates to the AGV data array
- Plot the location of the AGV

The actual MATLAB implementation of the function StateEstimation can be found in appendix C.3.

4.3.3 Checking the accuracy

For any CDR system to work, the state of the vehicles needs to be estimated with reasonable accuracy. As the estimated inaccuracies of the OptiTrack setup are relatively small (see section 3.3.2), it is expected that the position and orientation of the AGVs are estimated with a sufficient accuracy to test the CDR system developed. A simple setup has been made however of three AGVs (where a gap of 1-2cm was left in between them) to build more confidence as if this is really the case. The setup made is shown in figure 4.3a. Figure 4.3b shows the corresponding setup as perceived through OptiTrack and Matlab. Although no exact measurements have been taken, this image confirms that the accuracy of the state estimation is sufficient to test the CDR system developed. No overlap or excessive misalignment is shown by the rectangles that represent the model AGVs.
4.4 Trajectory Prediction Algorithm

4.4.1 Method

Section 2.3 has shown various ways of how the estimated current state can be propagated into the future. For collision detection in the AGV lab, the only future state variables of interest are the x,z coordinates and yaw over time (i.e. the trajectory and corresponding orientation). It is decided to first implement a simple propagation method based on extrapolating the current velocity vector. The concept of such a constant velocity model was shown in figure 2.2a.

The advantages of using this simple model are the ease of implementation and the low computational time. The biggest disadvantage is the expected low accuracy. Any future changes of control inputs (i.e. future maneuvers) are also not predicted with this model. Although the constant velocity model is fully correct when driving in a straight line with constant velocity, it is inherently wrong in all cases where the path isn’t straight or the (angular) velocity isn’t constant. For a short prediction horizon it may however be sufficient to test the initial design of the CDR system. In the future, more advanced propagation methods can replace the simple constant velocity model if the accuracy is found to be too low. The computational time needed for the simple constant velocity method can then serve as a benchmark to review the computational performance of the advanced algorithm.

Computational performance is an important parameter in the design of a CDR system. Latency and calculation time impose a delay between the measurements and the actions taken. The shorter this period of uncertainty is, the more reactive the CDR system will be [8]. This implies that the faster the algorithms are, the better the CDR performs in terms of the false alarm rate and number of late/missed detections.

The constant velocity model chosen is a non-probabilistic, single-trajectory model (see section 2.3). With every new state estimation, a new trajectory is predicted for each AGV using only
the current state and the previous state. The predicted trajectories are discretized into \( i \) steps with a specific time interval \( T_{\text{pred}I} \). This continues until the prediction horizon \( T_{\text{pred}H} \) is reached (i.e. \( i \times T_{\text{pred}I} = T_{\text{pred}H} \)). Figure 4.4 presents a schematic representation of this. The mathematical equations for the model are stated in equations 4.1 and 4.2.

\[
\begin{align*}
\text{Current state AGVn} & = \begin{bmatrix} X_{\text{cur}} \\ Z_{\text{cur}} \\ \theta_{\text{cur}} \\ T_{\text{cur}} \end{bmatrix} \\
\text{Previous state AGVn} & = \begin{bmatrix} X_{\text{prev}} \\ Z_{\text{prev}} \\ \theta_{\text{prev}} \\ T_{\text{prev}} \end{bmatrix}
\end{align*}
\]

\[
\text{State propagation} \rightarrow \begin{bmatrix} X_{\text{pred,i}} \\ Z_{\text{pred,i}} \\ \theta_{\text{pred,i}} \\ T_{\text{pred,i}} \end{bmatrix} = \begin{bmatrix} X_{\text{cur}} + i \times T_{\text{pred}I} \times \frac{dX}{dT} \\ Z_{\text{cur}} + i \times T_{\text{pred}I} \times \frac{dZ}{dT} \\ \theta_{\text{cur}} + i \times T_{\text{pred}I} \times \frac{d\theta}{dT} \\ T_{\text{cur}} + i \times T_{\text{pred}I} \end{bmatrix}
\]

Figure 4.4: State propagation method used

\[
\begin{align*}
\frac{dX}{dT} & = X_{\text{cur}} - X_{\text{prev}} \\
\frac{dZ}{dT} & = Z_{\text{cur}} - Z_{\text{prev}} \\
\frac{d\theta}{dT} & = \theta_{\text{cur}} - \theta_{\text{prev}} \\
\frac{dT}{dT} & = T_{\text{cur}} - T_{\text{prev}} \\
X_{\text{pred,i}} & = X_{\text{cur}} + i \times T_{\text{pred}I} \times \frac{dX}{dT} \\
Z_{\text{pred,i}} & = Z_{\text{cur}} + i \times T_{\text{pred}I} \times \frac{dZ}{dT} \\
\theta_{\text{pred,i}} & = \theta_{\text{cur}} + i \times T_{\text{pred}I} \times \frac{d\theta}{dT} \\
T_{\text{pred,i}} & = T_{\text{cur}} + i \times T_{\text{pred}I}
\end{align*}
\]

Where:

- \( X_{\text{prev}} \): X-coordinate of the previous frame of data [m]
- \( Z_{\text{prev}} \): Z-coordinate of the previous frame of data [m]
- \( \theta_{\text{prev}} \): yaw of the previous frame of data [rad]
- \( T_{\text{prev}} \): Timestamp of the previous frame of data [s]
- \( i \): Index to indicate the \( i \)-th step of the predicted trajectory [-]
- \( T_{\text{pred}I} \): Prediction time interval [s]
- \( X_{\text{pred,i}} \): Predicted X-coordinate at \( T_{\text{pred,i}} \) [m]
- \( Z_{\text{pred,i}} \): Predicted Z-coordinate at \( T_{\text{pred,i}} \) [m]
- \( \theta_{\text{pred,i}} \): Predicted yaw at \( T_{\text{pred,i}} \) [rad]
- \( T_{\text{pred,i}} \): Timestamp of the \( i \)-th step of the predicted trajectory [s]

4.4.2 Implementation

The trajectory prediction algorithm is implemented in Matlab as the function \textit{TrajectPrediction}. It is called right after the \textit{StateEstimation} function through the \textit{FrameReadyCallback} function.

Code outline function \textit{TrajectPrediction}
For every model AGV do:

Master of Science Research Assignment
T.D. Hogenboom
20 Design of the CDR system

- **If** the current measurement is the first measurement
  - Set the predicted location and yaw for all $i$ steps equal to the current location/yaw
  - Plot the initial prediction

- **Else**
  - Extrapolate the current velocity vector using equations 4.1 and 4.2
  - Determine the $x,z$ coordinates of the corner points of the rectangle that represents the AGV at each step and put them in a 3D array
  - Write the predicted trajectory to the AGV data array
  - Plot the predicted trajectory

The actual MATLAB implementation of the function $\text{TrajectPrediction}$ can be found in appendix C.4.

### 4.5 Conflict detection

#### 4.5.1 Method

For conflict detection, a non-probabilistic, binary method is chosen based upon the TTC only (see section 2.4). The AGVs will be represented in Matlab by rectangular PZs which have the length and the width of an AGV. Overlap of all combinations of two PZs will be checked every time new trajectories are predicted. Two AGVs are considered to be in conflict when, at a certain moment within the prediction horizon, two PZs overlap and the $TTC \leq TTC_{\text{limit}}$ (where the $TTC_{\text{limit}}$ is a predefined limit for the Time-To-Collision). If a conflict is detected, a simple conflict resolution method (see section 4.6) will be called. Figure 4.5 presents a schematic overview of this.

![Figure 4.5: Conflict detection method used](image)

To check whether the PZs of two AGVs overlap, an algorithm is needed that can check if two rectangles overlap or not. In the work of Terstegge [9], a method is proposed for this where an equation for every line segment of the PZs is retrieved whereafter overlap is checked by solving the intersection of line segments. Although this method can be directly applied to the case of the AGV lab, using it has a two main disadvantages. The first is that it is computationally burdensome; the algorithm of Terstegge needs to check the intersection of every pair of line segments (of different AGVs) on every data frame by solving a system of equations. The second disadvantage is that if two PZs have a different shape or size, the case that one PZ is contained by the other (without any intersecting edges) is not detected as a
4.5 Conflict detection

Because of these disadvantages, a different method is used for checking the overlap of PZs during this research. The method of choice is based on the separating axis theorem (SAT). This method is computationally more efficient and is frequently used for collision detection purposes. Also it doesn't suffer from the second disadvantage of the algorithm of Terstegge; it works for any configuration of two convex polygons.

The separating axis theorem shortly states that in 2D, two convex polygons don't overlap when an axis can be found that separates them; if this axis can't be found, the polygons overlap [4,10]. The SAT works by projecting the shapes of the pair of polygons onto a separating axis. The polygons don't overlap if the projections on the separating axis don't overlap. Figure 4.6 shows the concept of this. The separating axes to check are the normal vectors to each line segment of the polygons. As in any normal situation no collisions are happening, this algorithm is efficient because it can be exited when one separating axis has been found on which there is no overlap. The computations are also fairly simple as, for example, no systems of equations need to be solved.

![Figure 4.6: Example of how the SAT works [4]](image)

4.5.2 Implementation

The conflict detection algorithm is implemented in Matlab using two functions: *ConflictDetection* and *CheckPolygOverlap*. Function *ConflictDetection* is called through the *FrameReadyCallback* function every time new trajectories have been predicted. It checks every possible combination of pairs of PZs for overlap along the predicted trajectories and calls conflict resolution when a conflict has been detected. Function *CheckPolygOverlap* is called by *ConflictDetection* for every pair of PZs that need to be checked for overlap. It is decided to make the *CheckPolygOverlap* function generic for use with any pair of convex polygons (not only
for rectangles).

**Code outline function** *ConflictDetection*

- Determine all possible combinations of pairs of AGVs
- **For** every combination of two AGVs on every i-th step of the predicted trajectories **do**
  - Call *CheckPolygOverlap* to check if the two PZs overlap in the predicted future state
  - **If** a overlap is found **and** the \( TTC \leq TTC_{\text{limit}} \) **then**
    * Call *ConflictResolution* for the pair of AGVs

**Code outline function** *CheckPolygOverlap*

- Initially assume that overlap = true
- Determine the number of edges of each polygon
- **For** every line segment of the first polygon **do**
  - Find the corresponding edge vector and the left normal vector of it
  - Project all corner points of the first polygon onto the normal vector
  - Project all corner points of the second polygon onto the normal vector
  - Check if the projections of the two polygons overlap
  - **If** the projections do **not** overlap **then**
    * overlap = false, return to the invoking function
- **For** every line segment of the second polygon **do**
  - Find the corresponding edge vector and the left normal vector of it
  - Project all corner points of the first polygon onto the normal vector
  - Project all corner points of the second polygon onto the normal vector
  - Check if the projections of the two polygons overlap
  - **If** the projections do **not** overlap **then**
    * overlap = false, return to the invoking function

The actual MATLAB implementations of the functions *ConflictDetection* and *CheckPolygOverlap* can be found respectively in appendices C.5 and C.6.

**4.6 Conflict resolution**

As the focus of this research is towards conflict detection, only a very simple conflict resolution method will be implemented. The model AGVs will be stopped when a collision is detected such that experiments can be executed successfully.

Conflict resolution is implemented in Matlab as the function *ConflictResolution* and is called by *ConflictDetection* for every pair of AGVs that is in conflict.

**Code outline function** *ConflictResolution*
4.7 Conclusion

- Send "stop" command to the first AGV
- Send "stop" command to the second AGV
- Return to ConflictDetection for the next pair of AGVs

The actual MATLAB implementation of the function ConflictDetection can be found in appendix C.7.

4.7 Conclusion

This chapter aimed to answer the following central research questions:

- How can the state (e.g. position, speed, orientation) of the model AGVs be estimated using OptiTrack and Matlab?
- How can the future states (e.g. future position, orientation) of the model AGVs be predicted using the estimated states?
- How can imminent collisions between model AGVs be detected using the predicted states?

A method for state estimation has been chosen that uses a direct read-out of the OptiTrack observations for $[X_{\text{cur}}, Z_{\text{cur}}, \theta_{\text{cur}}, T_{\text{cur}}]$. Implementation in Matlab is achieved through the function StateEstimation.m.

A simple constant velocity model has been chosen to predict discretized future trajectories based upon both the estimated current and previous states. This non-probabilistic, single-trajectory model has as main advantages that implementation is relatively easy and the computational requirements are low. The expected accuracy is however low for curved trajectories and/or non-constant (angular) velocities. Matlab implementation is achieved through the function TrajectPrediction.m.

For detection and resolution of imminent collisions, a non-probabilistic, binary method is chosen that is based upon the Time-To-Collision (TTC) only. Two AGVs are defined to be in conflict when the rectangular protected zones that represent them overlap in the predicted future trajectories and the $TTC \leq TTC_{\text{limit}}$. Overlap of two protected zones is checked using the Separating Axis Theorem. A simple conflict resolution method is called that stops the pair of AGVs when a conflict has been detected. Implementation in Matlab of this conflict detection and resolution method is achieved through the functions ConflictDetection.m, CheckPolygOverlap.m and ConflictResolution.m.
Chapter 5

Experiments

5.1 Introduction

In chapter 4 the design of a CDR system for model AGVs was discussed. It was also shown how this system is implemented in the AGV lab through MATLAB. The goal of this final chapter is to review the performance of this CDR system through a number of experiments. Therewith this chapter answers the final central research question.

To perform experiments in the AGV lab, open-loop control functionality has been build that provides each AGV with a predefined task list that it executes from the moment an experiment is started. A task list consists of a number of successive commands for the two steering servos and the driving servo with a specified time interval. In this way driving tasks can be made that for example look like the following: "wait 2 seconds" then "drive straight for 3 seconds" then "circle clockwise with a minimum turning radius using front axis for 3 seconds" then "stop". Routes are thus not fixed in space and no feedback-controller with route-following capabilities is needed. As the CDR system developed operates using observable data only, the task list-based method chosen is sufficient for the experiments while providing consistency throughout multiple runs.

Reviewing the performance of the CDR system is split up into three parts. First, experiments with only one AGV are executed that address the performance of the state estimation. The setup, results and evaluation of these experiments are discussed in section 5.2. Thereafter, in section 5.3, the same is done to review the performance of the trajectory prediction algorithm. Finally, in section 5.4, the setup and results of experiments are discussed in which (non-)collision scenarios with multiple AGVs are executed.

5.2 State estimation performance

5.2.1 Experimental setup

The method used for state estimation is actually a direct read-out of the OptiTrack sensor measurements for the x,z location of the rigid body. The yaw is also more-or-less a direct read-out from OptiTrack, except that it is translated from quaternions to an Euler angle. Currently there is no possibility in the AGV lab of checking how accurate the measurements of the x,z location and yaw actually are beyond simple checks such as the one described in section 4.3.3. For this research there is also little benefit of knowing the accuracy with a higher precision than shown in section 4.3.3. As with any sensor measurements however, a certain amount of noise can be expected in the OptiTrack measurements. Excessive amounts of noise in the OptiTrack measurements will have a significant negative effect on the current design of the CDR system because no filtering is used at all. The current trajectory prediction algorithm indirectly computes the (angular) velocity for the AGVs based on the (angular)
distance progressed between two data frames. In this way, without filtering, noise in the data can even be amplified by the trajectory prediction algorithm.

To test the amount of noise in the OptiTrack data, the (angular) speed of one AGV will be estimated directly in a number of pre-defined driving cases. The main parameter of interest for these experiments is the frame rate. As was explained in section 4.4.1, the lower the frame rate, the lower the responsiveness of the CDR system will be. Also, because the actual continuous trajectory is interpreted as a piece-wise linear trajectory (through discrete OptiTrack observations), the lower the framerate, the less accurate the actual trajectory (and therewith the velocity) is reflected. This behavior is illustrated in figure 5.1. At the other side, the higher the frame rate, the more noise is expected from the measurements. This is because the (angular) distance progressed between two data frames becomes smaller such that interference with the expected OptiTrack accuracy (as discussed in section 3.3.2) is expected to be greater. The experiments that are executed to evaluate the noise in the OptiTrack measurements are displayed in table 5.1. It is chosen to let the AGV start from a standstill whereafter it drives at constant (angular) velocity for a time period that permits the evaluation of the steady-state behavior.

<table>
<thead>
<tr>
<th>Driving case</th>
<th>Description</th>
<th>Experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Start from standstill at $\sim[0;-2.5m]$ in $+z$ dir. Drive straight for 10s</td>
<td>$1, 5, 10, 30, 60, 120$FPS</td>
</tr>
<tr>
<td>102</td>
<td>Start from standstill at $\sim[-0.5m;-0.5m]$ in $+z$ dir. Circle clockwise for 15s (front steering axis)</td>
<td>$1, 5, 10, 30, 60, 120$FPS</td>
</tr>
<tr>
<td>103</td>
<td>Start from standstill at $\sim[-0.5m;-0.5m]$ in $+z$ dir. Circle clockwise for 11s (both steering axes)</td>
<td>$1, 5, 10$FPS</td>
</tr>
</tbody>
</table>

**Figure 5.1**: Linearized trajectory using OptiTrack data

**Table 5.1**: Experiments state estimation performance

**5.2.2 Results**

Figure 5.2 shows an example of the results of one of the experiments from table 5.1. The full results are displayed in appendix A.1. The steady-state results for the speed and angular velocity estimates are displayed in tables 5.2 and 5.3 respectively.\(^1\)

\(^1\)The 95% confidence intervals are stated under the assumption of normally distributed data
The results are very consistent throughout the three different driving cases and also match the expectations very well. The mean steady-state results for the velocity are also similar to the results presented by Bentvelsen [6]. No other methods of (angular) velocity estimation have been used during this research for quantitative validation of the results due to time restrictions. The remainder of this section therefore presents a qualitative analysis of the results to determine how the current method for state estimation performs.

At the low frame rate of 1FPS, the steady-state variance is low. It is however clear that the effect of the behavior illustrated in figure 5.1 is large; the acceleration (≈0-2s) is not reflected in the estimation at all (see figures A.1a, A.2a and A.3a) and the estimated speed on curved trajectories is significantly lower than at higher frame rates (see table 5.2). Higher frame rates lead (as expected) to an increasing variance (noise) in the data. From 30FPS, the estimated (angular) velocity doesn’t reflect plausible steady-state behavior anymore. Some short-term variation is expected in the (angular) velocity due to the dynamical behavior of the physical model; from 30FPS this short-term variation is however interpreted as excessive (i.e. noise). The results of the experiments at 5 and 10FPS show a plausible approximation of the actual (angular) velocity of the AGV.
### Table 5.2: Steady-state results speed estimation

<table>
<thead>
<tr>
<th>Frame rate</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>30</th>
<th>60</th>
<th>120</th>
<th>[FPS]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.299</td>
<td>0.301</td>
<td>0.303</td>
<td>0.301</td>
<td>0.302</td>
<td>0.302</td>
<td>[m/s]</td>
</tr>
<tr>
<td>Variance (*1E-5)</td>
<td>0.648</td>
<td>0.702</td>
<td>0.880</td>
<td>2.505</td>
<td>7.290</td>
<td>21.444</td>
<td>[m/s]^2</td>
</tr>
<tr>
<td>95% CI (min)</td>
<td>0.297</td>
<td>0.301</td>
<td>0.302</td>
<td>0.301</td>
<td>0.302</td>
<td>0.301</td>
<td>[m/s]</td>
</tr>
<tr>
<td>95% CI (max)</td>
<td>0.301</td>
<td>0.302</td>
<td>0.303</td>
<td>0.302</td>
<td>0.303</td>
<td>0.303</td>
<td>[m/s]</td>
</tr>
<tr>
<td>Case 102</td>
<td>Mean</td>
<td>0.265</td>
<td>0.270</td>
<td>0.271</td>
<td>0.270</td>
<td>0.271</td>
<td>0.271</td>
</tr>
<tr>
<td>Variance (*1E-5)</td>
<td>0.169</td>
<td>0.266</td>
<td>0.654</td>
<td>2.481</td>
<td>11.039</td>
<td>34.662</td>
<td>[m/s]^2</td>
</tr>
<tr>
<td>95% CI (min)</td>
<td>0.265</td>
<td>0.269</td>
<td>0.27</td>
<td>0.27</td>
<td>0.27</td>
<td>0.27</td>
<td>[m/s]</td>
</tr>
<tr>
<td>95% CI (max)</td>
<td>0.266</td>
<td>0.271</td>
<td>0.271</td>
<td>0.272</td>
<td>0.272</td>
<td>0.272</td>
<td>[m/s]</td>
</tr>
<tr>
<td>Case 103</td>
<td>Mean</td>
<td>0.215</td>
<td>0.226</td>
<td>0.226</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variance (*1E-5)</td>
<td>0.112</td>
<td>0.322</td>
<td>0.768</td>
<td></td>
<td></td>
<td></td>
<td>[m/s]^2</td>
</tr>
<tr>
<td>95% CI (min)</td>
<td>0.214</td>
<td>0.226</td>
<td>0.226</td>
<td></td>
<td></td>
<td></td>
<td>[m/s]</td>
</tr>
<tr>
<td>95% CI (max)</td>
<td>0.216</td>
<td>0.227</td>
<td>0.227</td>
<td></td>
<td></td>
<td></td>
<td>[m/s]</td>
</tr>
</tbody>
</table>

### Table 5.3: Steady-state results angular velocity estimation

<table>
<thead>
<tr>
<th>Frame rate</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>30</th>
<th>60</th>
<th>120</th>
<th>[FPS]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (*1E-2)</td>
<td>0.0116</td>
<td>-0.0872</td>
<td>-0.0989</td>
<td>-0.4454</td>
<td>-0.0461</td>
<td>0.0004</td>
<td>[rad/s]</td>
</tr>
<tr>
<td>Variance (*1E-5)</td>
<td>0.373</td>
<td>1.700</td>
<td>5.423</td>
<td>42.312</td>
<td>129.059</td>
<td>431.443</td>
<td>[rad/s]^2</td>
</tr>
<tr>
<td>95% CI (min)</td>
<td>-0.001</td>
<td>-0.002</td>
<td>-0.003</td>
<td>-0.007</td>
<td>-0.004</td>
<td>-0.004</td>
<td>[rad/s]</td>
</tr>
<tr>
<td>95% CI (max)</td>
<td>0.001</td>
<td>0.000</td>
<td>0.001</td>
<td>-0.002</td>
<td>0.003</td>
<td>0.004</td>
<td>[rad/s]</td>
</tr>
<tr>
<td>Case 102</td>
<td>Mean</td>
<td>0.651</td>
<td>0.651</td>
<td>0.653</td>
<td>0.651</td>
<td>0.652</td>
<td>0.651</td>
</tr>
<tr>
<td>Variance (*1E-5)</td>
<td>0.607</td>
<td>2.325</td>
<td>7.257</td>
<td>51.342</td>
<td>235.468</td>
<td>566.693</td>
<td>[rad/s]^2</td>
</tr>
<tr>
<td>95% CI (min)</td>
<td>0.649</td>
<td>0.65</td>
<td>0.651</td>
<td>0.649</td>
<td>0.648</td>
<td>0.646</td>
<td>[rad/s]</td>
</tr>
<tr>
<td>95% CI (max)</td>
<td>0.652</td>
<td>0.653</td>
<td>0.654</td>
<td>0.656</td>
<td>0.656</td>
<td>0.656</td>
<td>[rad/s]</td>
</tr>
<tr>
<td>Case 103</td>
<td>Mean</td>
<td>1.174</td>
<td>1.166</td>
<td>1.166</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variance (*1E-5)</td>
<td>2.342</td>
<td>5.570</td>
<td>17.952</td>
<td></td>
<td></td>
<td></td>
<td>[rad/s]^2</td>
</tr>
<tr>
<td>95% CI (min)</td>
<td>1.171</td>
<td>1.164</td>
<td>1.163</td>
<td></td>
<td></td>
<td></td>
<td>[rad/s]</td>
</tr>
<tr>
<td>95% CI (max)</td>
<td>1.177</td>
<td>1.169</td>
<td>1.169</td>
<td></td>
<td></td>
<td></td>
<td>[rad/s]</td>
</tr>
</tbody>
</table>
5.2.3 Evaluation

The direct read-out method for state estimation using the OptiTrack sensor measurements is sufficient when the frame rate is between 5 and 10FPS. Lower frame rates can be used but will lead to a significant decrease in performance. Using higher frame rates leads to excessive variance/noise in the data and is therefore discouraged.

5.3 Trajectory prediction algorithm performance

5.3.1 Experimental setup

As was explained in section 4.4.1, the trajectory prediction algorithm used during this research assumes a simple constant velocity model. The performance of this method is given by a comparison between the predicted location and yaw to the actual location and yaw at the corresponding moment in time. The accuracy of the location prediction is measured by computing the absolute rectilinear distance between the predicted and actual location. The accuracy of the yaw prediction is given by the absolute difference in the predicted vs. the actual yaw.

The performance of the trajectory prediction algorithm can be influenced by the following parameters:

1. **Accuracy of the state estimation**
   Inaccuracy (including noise) in the state estimation might be propagated or may even be amplified by the trajectory prediction algorithm. A frame rate that is too low can also lead to a decrease in accuracy.

2. **Prediction horizon**
   Accurate predictions usually become harder to make when they are further into the future.

3. **Trajectory**
   Some trajectories might be harder to predict accurately than others for a specific algorithm.

The influence of the trajectory and prediction horizon on the performance of the TPA will be evaluated with a number of experiments. The purpose of this is two-fold. The first is to see whether the implemented model shows the expected behavior (i.e. to verify the model). The second is to review how the TPA performs under certain conditions.

The accuracy (including the frame rate) of the state estimation is already discussed in section 5.2.2 and will not be tested any further due to the simplicity of the TPA used. Testing the performance of the TPA with respect to this parameter would be of greater value if more advanced methods (e.g. Kalman filtering) for state estimation and propagation would be used.

Three different driving cases (cases 101, 102 and 103) are used to test the TPA. The prediction horizon is varied from 0.5 to 3 seconds into the future with steps of 0.5 seconds. A frame rate of 10FPS is used. Table 5.4 presents an overview of these experiments.
Table 5.4: Experiments trajectory prediction accuracy

<table>
<thead>
<tr>
<th>Driving case</th>
<th>Experiments (prediction horizon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>0.5; 1; 1.5; 2; 2.5; 3s</td>
</tr>
<tr>
<td>102</td>
<td>0.5; 1; 1.5; 2; 2.5; 3s</td>
</tr>
<tr>
<td>103</td>
<td>0.5; 1; 1.5; 2; 2.5; 3s</td>
</tr>
</tbody>
</table>

5.3.2 Results

The full results of the experiments from table 5.4 can be found in appendix A.2. Table 5.5 displays the mean steady-state results\(^2\). Steady-state is defined here as driving with a constant speed and angular velocity (except for the dynamical behavior that’s inherent to the physical build of the model AGV itself). The range of prediction frame 20-70 is taken to reflect the steady-state behavior. For the location prediction, the mean error in table 5.5 represents the mean absolute distance between the predicted and actual location. Besides in meters, the mean location error is also displayed as a percentage of the AGV width such that interpretation of scaling is easier. The mean error for the yaw represents the mean absolute difference between the predicted and actual yaw.

As can be expected from a constant (angular) velocity prediction model, the predictions are far off and/or inconsistent during (angular) acceleration (prediction frame ≈0-10). For a prediction horizon of 0.5s, the maximum errors in the predicted location and yaw are however not greater than around 0.1m and 20deg respectively.

For the steady-state behavior, the results are also as expected. With a straight path (case 101), the mean absolute location error is at most around 6% of the vehicle width (9.2mm) for a prediction horizon of 3s. As all driving cases have a constant angular velocity during the steady-state part, the mean absolute errors in the predicted yaw remain below 2 degrees.

For curved trajectories the location prediction accuracy quickly deteriorates with an increasing prediction horizon. For a prediction horizon of one second, the mean absolute location errors in driving a circle with one and two axes are already respectively around 66 and 96% of the vehicle width. This increases to respectively 508 and 588% of the vehicle width at three seconds; making the algorithm totally ineffective for predicting curved trajectories at more than 0.5-1s into the future.

As a side note, it is nice to see how figure A.7 presents an example of how noise in the state estimation data can get amplified through the trajectory prediction algorithm. As the path driven in this case is straight, only minor short-term changes in the actual yaw occur. These minor changes are however amplified into large peak-errors for longer prediction horizons.

---

\(^2\)The 95% confidence intervals are stated under the assumption of normally distributed data
Table 5.5: Mean steady-state trajectory prediction performance

<table>
<thead>
<tr>
<th>Location</th>
<th>0.5</th>
<th>1</th>
<th>1.5</th>
<th>2</th>
<th>2.5</th>
<th>3</th>
<th>[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Case 101</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean error</td>
<td>1.6</td>
<td>3.1</td>
<td>4.6</td>
<td>6.2</td>
<td>7.8</td>
<td>9.2</td>
<td>[mm]</td>
</tr>
<tr>
<td>1% 2% 3% 4% 5% 6%</td>
<td>% of AGV width</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variance</td>
<td>0.9</td>
<td>3.6</td>
<td>7.0</td>
<td>10.8</td>
<td>16.8</td>
<td>26.3</td>
<td>[mm^2]</td>
</tr>
<tr>
<td>95% CI min</td>
<td>1.3</td>
<td>2.5</td>
<td>3.9</td>
<td>5.3</td>
<td>6.6</td>
<td>7.8</td>
<td>[mm]</td>
</tr>
<tr>
<td>95% CI max</td>
<td>1.8</td>
<td>3.6</td>
<td>5.4</td>
<td>7.1</td>
<td>8.9</td>
<td>10.7</td>
<td>[mm]</td>
</tr>
<tr>
<td><strong>Case 102</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean error</td>
<td>26.4</td>
<td>96.0</td>
<td>206.3</td>
<td>353.5</td>
<td>532.2</td>
<td>736.1</td>
<td>[mm]</td>
</tr>
<tr>
<td>18% 66% 142% 244% 367% 508%</td>
<td>% of AGV width</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variance</td>
<td>1.2</td>
<td>4.5</td>
<td>10.4</td>
<td>18.8</td>
<td>30.2</td>
<td>42.9</td>
<td>[mm^2]</td>
</tr>
<tr>
<td>95% CI min</td>
<td>26.1</td>
<td>95.4</td>
<td>205.5</td>
<td>352.3</td>
<td>530.7</td>
<td>734.3</td>
<td>[mm]</td>
</tr>
<tr>
<td>95% CI max</td>
<td>26.7</td>
<td>96.6</td>
<td>207.2</td>
<td>354.7</td>
<td>533.7</td>
<td>737.9</td>
<td>[mm]</td>
</tr>
<tr>
<td><strong>Case 103</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean error</td>
<td>39.1</td>
<td>139.1</td>
<td>288.8</td>
<td>471.4</td>
<td>666.6</td>
<td>853.3</td>
<td>[mm]</td>
</tr>
<tr>
<td>27% 96% 199% 325% 460% 588%</td>
<td>% of AGV width</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variance</td>
<td>3.4</td>
<td>13.6</td>
<td>32.3</td>
<td>58.0</td>
<td>88.1</td>
<td>122.3</td>
<td>[mm^2]</td>
</tr>
<tr>
<td>95% CI min</td>
<td>38.6</td>
<td>138.1</td>
<td>287.2</td>
<td>469.3</td>
<td>664.1</td>
<td>850.2</td>
<td>[mm]</td>
</tr>
<tr>
<td>95% CI max</td>
<td>39.6</td>
<td>140.2</td>
<td>290.4</td>
<td>473.5</td>
<td>669.2</td>
<td>856.3</td>
<td>[mm]</td>
</tr>
</tbody>
</table>

**Yaw**

<table>
<thead>
<tr>
<th>Location</th>
<th>0.2</th>
<th>0.3</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.9</th>
<th>[deg]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Case 101</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean error</td>
<td>0.01</td>
<td>0.05</td>
<td>0.12</td>
<td>0.20</td>
<td>0.30</td>
<td>0.44</td>
<td>[deg]</td>
</tr>
<tr>
<td>Variance</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.5</td>
<td>0.6</td>
<td>0.7</td>
<td>[deg]</td>
</tr>
<tr>
<td>95% CI min</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.7</td>
<td>0.9</td>
<td>1.1</td>
<td>[deg]</td>
</tr>
<tr>
<td>95% CI max</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.7</td>
<td>0.9</td>
<td>1.1</td>
<td>[deg]</td>
</tr>
<tr>
<td><strong>Case 102</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean error</td>
<td>0.3</td>
<td>0.5</td>
<td>0.8</td>
<td>1.0</td>
<td>1.3</td>
<td>1.5</td>
<td>[deg]</td>
</tr>
<tr>
<td>Variance</td>
<td>0.05</td>
<td>0.16</td>
<td>0.40</td>
<td>0.70</td>
<td>1.05</td>
<td>1.49</td>
<td>[deg]</td>
</tr>
<tr>
<td>95% CI min</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1.0</td>
<td>1.2</td>
<td>[deg]</td>
</tr>
<tr>
<td>95% CI max</td>
<td>0.3</td>
<td>0.6</td>
<td>0.9</td>
<td>1.2</td>
<td>1.6</td>
<td>1.9</td>
<td>[deg]</td>
</tr>
<tr>
<td><strong>Case 103</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean error</td>
<td>0.3</td>
<td>0.6</td>
<td>1.0</td>
<td>1.3</td>
<td>1.6</td>
<td>2.0</td>
<td>[deg]</td>
</tr>
<tr>
<td>Variance</td>
<td>0.06</td>
<td>0.21</td>
<td>0.48</td>
<td>0.86</td>
<td>1.43</td>
<td>2.10</td>
<td>[deg]</td>
</tr>
<tr>
<td>95% CI min</td>
<td>0.3</td>
<td>0.5</td>
<td>0.8</td>
<td>1.0</td>
<td>1.3</td>
<td>1.6</td>
<td>[deg]</td>
</tr>
<tr>
<td>95% CI max</td>
<td>0.4</td>
<td>0.8</td>
<td>1.2</td>
<td>1.6</td>
<td>2.0</td>
<td>2.4</td>
<td>[deg]</td>
</tr>
</tbody>
</table>

Master of Science Research Assignment T.D. Hogenboom
5.3.3 Evaluation

The constant velocity model used is sufficient for the CDR system in the AGV lab during this early design phase for simple rectilinear paths and/or curved paths with a very short prediction horizon (less than a second). For practical applications the constant velocity model is too much of an oversimplification of reality. The very poor performance with non-constant (angular) velocities and on curved paths with a prediction horizon of more than one second ahead make it indisputable that a more advanced method is needed for implementation in realistic cases.

5.4 Conflict Detection and Resolution performance

5.4.1 Experimental setup

In this last section the performance of the CDR system in total will be assessed through a number of driving scenarios with two AGVs. Different scenarios are set up in which a collision happens without a CDR system or in which the two AGVs just miss each other.

As was explained in section 2.4.4, the performance of a CDR system in total can be reviewed with two main KPIs, 1) the number of late/missed detections and 2) the false alarm rate. Unfortunately it is not possible to state statistically validated numbers for these main KPIs during this research due to the laborious execution of experiments and the high number of variables. Instead, the main purposes of the experiments stated in this section are to check if the CDR system behaves as expected (i.e. to verify the model) and to indicate where difficulties lie.

As became clear throughout this research, there are many things that can have an influence on the performance of the total CDR system. These things can roughly be divided over two groups: 1) characteristics that are inherent to the design of the system (e.g. the accuracy of the sensors and the Matlab calculation speed), and 2) parameters that can be set/varied. The first group is taken as a given throughout the experiments of this section. For the second group, the following parameters are of interest:

- **Scenario/trajectories driven**
  Some scenarios will be harder for the CDR system than others. Different scenarios will be tested with the model AGVs.

- **Frame rate**
  In section 5.2 it was shown that it is desired to keep the frame rate around 5-10FPS. A frame rate of 10FPS is chosen for the experiments described in this section.

- **Prediction horizon \( T_{predH} \)**
  Predictions further in the future are less accurate. \( T_{predH} \) will be kept at two seconds throughout the experiments.

- **Prediction time interval \( T_{predI} \)**
  If \( T_{predI} \) is too course, a collision may remain undetected because it happens in between two time-steps. \( T_{predI} \) will be kept at 0.2 seconds during the experiments as this corresponds to a frame rate of 5FPS (which was found to be the minimum desired frame rate).

- **Time-To-Collision limit \( TTC_{lim} \)**
  If the \( TTC_{lim} \) is too low, the AGVs won’t be able to stop in time for a collision. If it is
too high, a higher number of false alarms can be expected. The $TTC_{lim}$ will be varied throughout the experiments.

- **Size of the PZs that represent the AGVs**
  The size of the PZs will be kept equal to the length and width of an AGV throughout the experiments (as this is the minimum size needed). Increasing it can bring some extra safety margin into a final design.

The driving-scenario experiments end when the CDR system has stopped the AGVs, when the AGVs collide or when the AGVs finish their tasklist. The closest remaining distance between the AGVs will be measured when the CDR system has stopped the AGVs. A closest remaining distance of zero means that a collision has happened and that the CDR system thus has failed to stop the AGVs in time. If the CDR system stops the AGVs in a non-collision scenario, this is marked as a false alarm.

**Scenario 1**
The first scenario is displayed in figure 5.3. AGV2 stands still while AGV1 drives straight towards its side at full speed. A collision will happen without intervention of the CDR system in this scenario. The initial distance between the AGVs will be 1.8m. The $TTC_{lim}$ will be set to 0.2, 0.4, 0.6, 0.8, 1.2, 1.6 and 2.0 seconds with five replications each. This experiment shows if the implemented CDR functionality works like it should and provides the possibility of obtaining a suitable $TTC_{lim}$.

**Scenarios 2, 3 and 4**
After testing scenario 1, it became clear that the CDR functionality has been implemented correctly and it was shown how the remaining distance between the AGVs after conflict resolution increases with an increasing $TTC_{lim}$ (see section 5.4.2).

Scenarios 2, 3 and 4 are set up more dynamically and/or challenging to review the performance of the CDR functionality under various circumstances. The $TTC_{lim}$ will be kept at 0.8 seconds throughout these scenarios as this led to a remaining distance of around the AGV width in scenario 1 (which is viewed upon as a realistic minimum value). Three replications will be done for each scenario. The setup of scenarios 2, 3 and 4 can be found in appendix B. In scenario 2, the AGVs are set up to collide while in scenarios 3 and 4, the AGVs pass by each other closely (a gap of around the width of an AGV is kept in between).

**5.4.2 Results**

**Scenario 1**
Table 5.6 shows the statistics for the experiments of scenario 1. The full results are displayed...
in Table A.1. Figure 5.4 shows how the remaining distance between the AGVs increases with an increasing $TTC_{lim}$. The CDR system worked as expected as the imminent collision was detected in every run. At a $TTC_{lim}$ of 0.2 seconds, there was however not enough time left to stop the AGV in time and thus a collision happened on each of those replications. At a $TTC_{lim}$ of 0.4s, the AGV was stopped just in time whereafter the mean remaining stopping distance increases (as expected) linearly with an increasing $TTC_{lim}$. The minimum $TTC_{lim}$ for the model AGVs in this setup is thus 0.4 seconds. A maximum spread of around 30mm has been found in the stopping distance for identical replications.

Table 5.6: Remaining distance between AGVs scenario 1

<table>
<thead>
<tr>
<th>TTC</th>
<th>0.2</th>
<th>0.4</th>
<th>0.6</th>
<th>0.8</th>
<th>1.2</th>
<th>1.6</th>
<th>2</th>
<th>[s]</th>
<th>Mean</th>
<th>[mm]</th>
<th>0%</th>
<th>28%</th>
<th>73%</th>
<th>116%</th>
<th>199%</th>
<th>285%</th>
<th>375%</th>
<th>[% of AGV width]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>40.8</td>
<td>106.4</td>
<td>167.8</td>
<td>289.2</td>
<td>413.8</td>
<td>543.2</td>
<td>[mm]</td>
<td>0%</td>
<td>28%</td>
<td>73%</td>
<td>116%</td>
<td>199%</td>
<td>285%</td>
<td>375%</td>
<td>[% of AGV width]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>0</td>
<td>30</td>
<td>104</td>
<td>155</td>
<td>270</td>
<td>400</td>
<td>534</td>
<td>[mm]</td>
<td>0%</td>
<td>28%</td>
<td>73%</td>
<td>116%</td>
<td>199%</td>
<td>285%</td>
<td>375%</td>
<td>[% of AGV width]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>0</td>
<td>55</td>
<td>108</td>
<td>182</td>
<td>298</td>
<td>418</td>
<td>552</td>
<td>[mm]</td>
<td>0%</td>
<td>28%</td>
<td>73%</td>
<td>116%</td>
<td>199%</td>
<td>285%</td>
<td>375%</td>
<td>[% of AGV width]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0</td>
<td>10.6</td>
<td>1.7</td>
<td>10.8</td>
<td>11.7</td>
<td>7.8</td>
<td>8.3</td>
<td>[mm]</td>
<td>0%</td>
<td>28%</td>
<td>73%</td>
<td>116%</td>
<td>199%</td>
<td>285%</td>
<td>375%</td>
<td>[% of AGV width]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.4: Mean results scenario 1

Scenarios 2, 3 and 4
In scenario 2, the collision is detected in all three replications and the remaining distance between the AGVs over the three replications was respectively 147, 170 and 150 mm. No conflict was detected in scenario 3, the CDR system behaved like it should and no false
alarms were made. Scenario 4 led to a false alarm at two of the three replications with a remaining distance between the AGVs after resolution of 195 and 155mm. One time the CDR did not make a false alarm, this was probably due to a slightly different trajectory/initial position of AGV1; leading to a passing distance that was just sufficient for the $TTC_{lim}$ of 0.8s (a false alarm would have been made if the $TTC_{lim}$ was slightly higher).

Table 5.7 summarizes the results of all driving scenarios. It is clear that only in scenario 4, the CDR system did not perform as desired.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Collision setup?</th>
<th>Stopped in time?</th>
<th>False alarm?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>No</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>No</td>
<td>-</td>
<td>Yes</td>
</tr>
</tbody>
</table>

5.4.3 Evaluation

The CDR system implemented performed really well in all cases where the trajectory prediction was accurate (straight trajectories with a constant velocity and/or at a prediction horizon of 0.4 to 0.6s). However, when the prediction is off, this will lead to errors in the CDR functionality (which was shown in scenario 4). In the current setup, the trajectory prediction algorithm is thus the weakest link in the total CDR system.

5.5 Conclusion

The main goal of this chapter was to answer the final central research question: how does the developed CDR system perform when implemented in the AGV lab?

At first, the performance of the state estimation functionality was assessed. It was shown that the direct-read out method for the OptiTrack sensor measurements is sufficient when the frame rate is between 5 and 10FPS. A significant decrease in accuracy can be expected with lower frame rates. Higher frame rates lead to excessive variance/noise in the data.

Secondly, the performance of the trajectory prediction algorithm was assessed. The constant velocity model performed really well for simple rectilinear paths with a constant velocity. On curved paths or in situations with a non-constant velocity, the prediction was only acceptable within a very short prediction horizon (less than a second).

Finally, the performance of the CDR system in total was assessed. It was shown that the functionality of the CDR system in total is mostly compromised by the performance of the trajectory prediction algorithm. When the trajectory was predicted with a high accuracy, no false alarms were triggered and no collisions were missed. An incorrect predicted trajectory mainly leads to false alarms. Late/missed detections were not found because the AGVs can be stopped in time with a Time-To-Collision limit of only 0.4s (with such a short prediction horizon the prediction is accurate enough in all situations tested). No issues have been found with the conflict detection algorithm based on the Separated Axis Theorem.
This research aimed to answer the following main research question:
What is a suitable design for a system that detects imminent collisions between model AGVs in real-time by predicting future vehicle states based upon observed data only; and how does it perform when implemented in the AGV lab?

Design
A Conflict Detection and Resolution (CDR) system has been developed that consists of the following four main building blocks:

1. State estimation
2. State propagation
3. Conflict detection and risk assessment:
4. Conflict resolution

*State estimation* determines the current state (e.g. position, speed) of every AGV. For the AGV lab, a direct read-out method of the OptiTrack measurements for the x,z location and yaw of every model AGV has been chosen and implemented as the method for state estimation.

*State propagation* uses the estimated states to predict how the current situation will propagate in the future. For the model AGVs, this leads to predicted future trajectories consisting of the position and orientation of each model AGV over time. A trajectory prediction algorithm based on a constant velocity model has been used for this. The constant velocity model used is a non-probabilistic, single-trajectory model that extrapolates the current velocity vector as a straight trajectory into the future every time a new state has been estimated. Future trajectories are limited by a certain prediction horizon ($T_{\text{predH}}$) and are discretized into $i$ steps with a predefined time interval of $T_{\text{predI}}$ (i.e. $i \times T_{\text{predI}} = T_{\text{predH}}$).

*Conflict detection and risk assessment* checks whether a collision is imminent and, if so, checks whether the risk has surpassed a certain threshold. For the AGV lab, a conflict detection algorithm is chosen that uses a non-probabilistic, binary method (i.e. a collision is either predicted as going to happen or not going to happen). The Time-To-Collision (TTC) is the only risk indicator used. AGVs are represented in Matlab by a rectangular protected zone of the length and width of the AGV. The conflict detection algorithm uses the Separating Axis Theorem to check whether two protected zones overlap for every pair of AGVs on every $i$-th step of the discretized future trajectories. Two AGVs are considered to be in conflict (i.e. a collision is predicted to happen) when the protected zones that represent them overlap and the $TTC \leq TTC_{\text{limit}}$ (where the $TTC_{\text{limit}}$ is a predefined limit for the Time-To-Collision).

*Conflict resolution* is called when a risk has been detected that exceeds the threshold. Unacceptable risks of a detected conflict are avoided or mitigated by taking some actions. As this
research focused on conflict detection, only a very simple method for conflict resolution was used that was needed to successfully conduct experiments. The conflict resolution algorithm simply stops each pair of AGVs for which a conflict has been detected.

**Performance**
Before the performance of the total CDR system was assessed, the individual performance of the state estimation and the state propagation algorithms have been tested.

For *state estimation* it was shown through a number of experiments that the method chosen is sufficient for use with the model AGVs when the frame rate is between 5 and 10 frames per second (FPS). Below 5FPS the accuracy decreases significantly while above 10FPS the variance/noise in the data becomes excessive.

For *state propagation*, experiments have shown that, with curved trajectories and/or trajectories with a non-constant velocity, the trajectory prediction algorithm only generates useful results for a really short prediction horizon of less than a second. For straight trajectories with a constant velocity the algorithm performs really well; a mean absolute location error of only 6% of the AGV width and a mean absolute yaw error of only 0.9deg at a prediction horizon of three seconds were found. Despite the bad performance with curved trajectories and/or non-constant velocity, the results were good enough to develop and test the CDR functionality in the AGV lab. For practical applications with a prediction horizon of more than one second the constant velocity model doesn’t perform well enough.

The performance of the CDR system in total has been evaluated through a number of experiments in which driving scenarios with two AGVs were executed. These scenarios were set up in such a way that the AGVs would collide without CDR functionality or such that they would just miss each other. The experiments have shown that in cases where the trajectory prediction algorithm was accurate, the CDR functionality performed really well. The detection algorithm based on the Separating Axis Theorem worked flawless. When the $TTC_{\text{limit}}$ is set below 0.4 seconds however, the AGVs cannot be stopped in time anymore. There were no late/missed detections found for cases where the $TTC_{\text{limit}} \geq 0.4s$. In cases where the trajectory prediction algorithm was off (e.g. on curved trajectories) false alarms were detected. The trajectory prediction algorithm has thus been found as the weakest link in the current design of the CDR system. If however the predicted trajectories are accurate, the CDR system works properly.
Chapter 7

Recommendations

Generic CDR testing
One major drawback of using manually set up driving scenarios to test the total CDR functionality is that it not possible to state statistically substantiated figures for the number of late/missed detections and the false alarm rate. Executing a sufficient number of variations and replications in this way is simply too laborious. In order to be able to state these figures in the future, a generic CDR testing module should be made. Two different approaches are identified that may be chosen for this.

The first is to let a number of model AGVs drive in a demarcated area, following random collision and/or non-collision trajectories while automatically detecting late/missed detections and false alarms. This method ensures a high variety in possible (non-)collision scenarios. A drawback of this method is that the scenarios don’t necessarily reflect realistic behavior. A second approach is therefore to replicate an actual case where multiple AGVs are assigned the tasks they also fulfill in practice. To be able to automatically retrieve the number of late/missed detections and the false alarm rate, it may be necessary to implement a more advanced conflict resolution algorithm such that experiments aren’t interrupted when a conflict is detected. Simulation might also prove to be helpful for this.

Realistic state estimation
State estimation in the AGV laboratory was done through an OptiTrack motion capturing system. Although the direct OptiTrack read-out method used is sufficient for this research, it is probably not suited for practical applications for the following reasons. In real-life situations the accuracy of sensors (e.g. GPS) isn’t expected to be as good as when using OptiTrack in a controlled environment. The desired frame rate may also not always be reached due to computational/communication difficulties. Finally, the direct read-out method also doesn’t provide the possibility of combining sensor data from different sources (which is needed when for example GPS in combination with dead-reckoning is used for localization). More advanced methods for state estimation based on for example Kalman filtering may improve all of these points and lay the foundation for applications outside of a laboratory.

Improved trajectory prediction
It was shown that the current trajectory prediction algorithm (based upon a constant velocity model) is the weakest link in the CDR system developed. A logical follow-up step for this research would be to implement and test a more advanced trajectory prediction algorithm. A Kalman filter with a bicycle model can for example be used for this.

Advanced conflict resolution
The conflict resolution method used during this research (i.e. stop each pair of AGVs for which a conflict has been detected) can in practice only be used as an emergency stop as human intervention is needed to start up again. Implementing more advanced methods for conflict resolution can make human intervention superfluous.
Bibliography


## Glossary

### List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGV lab</td>
<td>Automated Guided Vehicle laboratory</td>
</tr>
<tr>
<td>CDR</td>
<td>Conflict detection and resolution</td>
</tr>
<tr>
<td>KF</td>
<td>Kalman Filter</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>IMM-KF</td>
<td>Interacting Multiple Model Kalman Filter</td>
</tr>
<tr>
<td>PZ</td>
<td>Protected zone</td>
</tr>
<tr>
<td>SAT</td>
<td>Separating Axis Theorem</td>
</tr>
<tr>
<td>TPA</td>
<td>Trajectory Prediction Algorithm</td>
</tr>
<tr>
<td>TTC</td>
<td>Time-To-Collision</td>
</tr>
<tr>
<td>TTR</td>
<td>Time-To-React</td>
</tr>
<tr>
<td>V2V</td>
<td>Vehicle to vehicle</td>
</tr>
</tbody>
</table>
Appendix A

Full experiment results
A.1 Results state estimation performance

Figure A.1: Velocity estimate driving straight for 12 seconds
A.1 Results state estimation performance

Figure A.2: Velocity estimate driving a circle clockwise for 15 seconds (front steering axis)

Master of Science Research Assignment  
T.D. Hogenboom
Figure A.3: Velocity estimate driving a circle clockwise for 11 seconds (both steering axes)
A.2 Results trajectory prediction performance

Figure A.4: Location prediction accuracy driving case 101

Figure A.5: Location prediction accuracy driving case 102
Figure A.6: Location prediction accuracy driving case 103

Figure A.7: Yaw prediction accuracy driving case 101
A.2 Results trajectory prediction performance

Figure A.8: Yaw prediction accuracy driving case 102

Figure A.9: Yaw prediction accuracy driving case 103
A.3 Results CDR performance

Table A.1: Full data scenario 1

<table>
<thead>
<tr>
<th>TTC [s]</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
<th>Run 4</th>
<th>Run 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.4</td>
<td>48</td>
<td>39</td>
<td>30</td>
<td>32</td>
<td>55</td>
</tr>
<tr>
<td>0.6</td>
<td>104</td>
<td>106</td>
<td>108</td>
<td>108</td>
<td>106</td>
</tr>
<tr>
<td>0.8</td>
<td>182</td>
<td>155</td>
<td>174</td>
<td>160</td>
<td>168</td>
</tr>
<tr>
<td>1.2</td>
<td>295</td>
<td>297</td>
<td>270</td>
<td>286</td>
<td>298</td>
</tr>
<tr>
<td>1.6</td>
<td>416</td>
<td>418</td>
<td>418</td>
<td>400</td>
<td>417</td>
</tr>
<tr>
<td>2</td>
<td>550</td>
<td>545</td>
<td>552</td>
<td>535</td>
<td>534</td>
</tr>
</tbody>
</table>
Appendix B

Driving scenarios

Figure B.1: Driving scenario 1

Figure B.2: Driving scenario 2
Figure B.3: Driving scenario 3

Figure B.4: Driving scenario 4
% Matlab main file for executing experiments within the TEL AGV lab.

%% v0 (original): Ted Bentvelsen - TrackingMain
%% v1 17-08-2017: Thijs Hogenboom - Major cleanup, added Xbee and CDR
%% functionality, some bugfixes

clear all
close all
clc

% Define global variables
global totalDropped;
global totalDuplicate;
global AGV; %Variable AGV is of type AGVclass

% Load user-defined parameters from UserParameters.m
UserParameters

% Initialize connections
%Xbee serial connection
if ControlData.driveRoute == true
    ControlData.serialPort = OpenXbeeConnection(ControlData);
end

%Optitrack client
[theClient, errCode] = OpenMotiveConnection(ClientIP, ServerIP, CastMethod);
if errCode ~= 0
    return
end

% Retrieval of information from OptiTrack
%Retrieve Motive framerate
ControlData.frameRate = GetFrameRate(theClient);
fprintf('\n[NatNet] OptiTrack Frame Rate : \%d fps\n\n', ControlData.frameRate)

% Retrieve the number of rigidbodies from Motive
ControlData.nRigidBodies = RetrieveObjectData(theClient);

%% Initialize location figure and hold
locationFigure = figure('Name','AGV Position Data','NumberTitle','off');
hold on % Better to put a hold off somewhere also??
title('AGV position tracking')
xlabel('X position [m]')
ylabel('Z position [m]')
set(gca,'YLim',[-2.5 2.5])
set(gca,'XLim',[-2.5 2.5])
set(gca,'XGrid','on','YGrid','on')

%% Create frame ready event handler
% This listener calls FrameReadyCallback every time a new frame from motive
% is received. This is the main "loop"
ls = addlistener(theClient,'OnFrameReady2',@(src,event) FrameReadyCallback(src,event,ControlData));
uiwait(locationFigure); % Wait until the locationFigure is closed

%% Termination of the connections
% Stop the event handler
if (~isempty(ls))
    delete(ls);
end

% Close Xbee serial connection
if ControlData.driveRoute == true
    CloseXbeeConnection(ControlData,AGV);
end

% Disconnect with NatNet server. Improper termination leads to Matlab errors!
CloseMotiveConnection(theClient);

%% Display total dropped frames and duplicate frames
% High numbers of dropped/duplicate frames may indicate issues
fprintf('\n[NatNet] Total Dropped Frames : \%d frame(s)', totalDropped);
fprintf('\n[NatNet] Total Duplicate Frames : \%d frame(s)', totalDuplicate);
C.2 FrameReadyCallBack.m

```matlab
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Matlab Function Frame Ready Callback
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%This function is called every time the addlistener recieves a
% frame from Motive
%
function FrameReadyCallback (src, event , ControlData)

%% Definition of variables
persistent fCounter
persistent initFrameTime
persistent lastFrameTime
persistent lastCalcTime
persistent arrayIndex

global AGV
global totalDropped
global totalDuplicate

%% Initialize / update function parameters
% Retrieve the motive data
frameOfData = event.data; %Copied from NatNetSample
%Retrieve the frame timestamp from motive
%Note: fLatency can be used but might not be the best. Maybe
%better to
%use fTimestamp or look into SMPTE timecodes?
frameTime = frameOfData.fLatency;

%% Initialize variables
if isempty(fCounter)
    fCounter = 0; %Counter for nr of times
    FrameReadyCallback is called
    arrayIndex = 0; %arrayIndex for the AGV data
    initFrameTime = frameTime; %Timestamp of first frame
    lastFrameTime = frameTime; %Timestamp of previous frame
    lastCalcTime = frameTime; %Timestamp of previous data
    processing
    totalDropped = 0;
    totalDuplicate = -1; %Compensate with one as startup is
    a fake duplicate
end
```

Master of Science Research Assignment

T.D. Hogenboom
%% Count the number of times FrameReadyCallback is called.
fCounter = fCounter + 1;

%% Determine if Motive frames are dropped or duplicate
% Note: High numbers of dropped/duplicate frames may indicate
% OptiTrack/communication issues. Method from NatNetSample.
calcFrameInc = round((frameTime - lastFrameTime) * ControlData.frameRate);
if(calcFrameInc > 1)
    % Number of frames missed is calcFrameInc - 1
    totalDropped = totalDropped + calcFrameInc - 1;
elseif(calcFrameInc == 0)
    totalDuplicate = totalDuplicate + 1;
end

%% Initialize and/or update AGVs
% Initialize
if fCounter == 1 % on first frame
    % Use class constructor to make the AGVs
    AGV = AGVclass.empty;
    for ID = 1 : ControlData.nRigidBodies
        AGV(ID) = AGVclass(ID);
        % Assign each AGV its TaskList
        if ControlData.driveRoute == true % UserParameter
            [AGV(ID).TaskList, AGV(ID).TaskListTimeLine] = 
            BuildDriveTask(ControlData.AGVTaskSelect(ID)) ;
            end
        end
    end

% Update
if ControlData.driveRoute == true % UserParameter
    currTime = frameTime - initFrameTime; % current time in [s] from first frame
    nFinished = 0; % Reset number of finished AGVs
    for i = 1 : ControlData.nRigidBodies
        % Update AGV "controller" (function in AGV class)
        UpdateAGV(AGV(i),currTime,ControlData)
        % Check if the AGV has finished its TaskList
        if AGV(i).TaskFinished == true
            nFinished = nFinished + 1;
        end
    end

% Stop the experiment by closing the locationFigure when all AGVs
% have finished their TaskList
if nFinished == ControlData.nRigidBodies
    close
end

%%% Perform data processing if the frame is not skipped
% Skip frames to get to the desired framerate
if rem(fCounter, round(ControlData.frameRate/ControlData.desiredFrameRate)) == 0
    if( frameOfData.RigidBodies.Length () > 0) %Unnecessary?
        %Set the arrayIndex for the AGV
        arrayIndex = arrayIndex + 1;

        %Perform state estimation calculations
        StateEstimation(frameOfData,arrayIndex,ControlData);

        %Time increment since last calculation
        dT = frameTime - lastCalcTime;

        %Perform conflict detection and resolution if desired. Set
        %true/false in UserParameters.m
        if ControlData.TrajectPredict == true
            %First predict trajectories for all AGVs
            TrajectPrediction(arrayIndex,dT,ControlData)
            %Check for future conflicts and solve them
            if ControlData.ConflictDetect == true
                ConflictDetection(ControlData)
            end
        end
    end
end
lastCalcTime = frameTime;
end
end
lastFrameTime = frameTime;
%%% Function calculates the current state of all AGVs every time it is called using the OptiTrack data.

function StateEstimation(frameOfData, arrayIndex, ControlData)

    % Definition of variables
    global AGV

    % Calculate the state of each AGV
    for ID = 1 : ControlData.nRigidBodies
        %Retrieve the rigidbodydata from Motive
        rigidBodyData = frameOfData.RigidBodies(ID);

        %Fill the AGV data array with position data
        AGV(ID).locationX(arrayIndex) = rigidBodyData.x;
        AGV(ID).locationZ(arrayIndex) = -rigidBodyData.z; %-z such that positive z is away from the desk

        %Calculate the yaw of the AGV, regarding singularities.
        pole = rigidBodyData.qx * rigidBodyData.qy +
              rigidBodyData.qz * rigidBodyData.qw;
        if pole > 0.499
            AGV(ID).yaw(arrayIndex) = 2 * atan2(rigidBodyData.qx, rigidBodyData.qw);
        elseif pole < -0.499
            AGV(ID).yaw(arrayIndex) = -2 * atan2(rigidBodyData.qx, rigidBodyData.qw);
        else
            AGV(ID).yaw(arrayIndex) = atan2(2 * rigidBodyData.qy * rigidBodyData.qw - 2 * rigidBodyData.qx * rigidBodyData.qz, 1 - 2 * rigidBodyData.qy^2 - 2 * rigidBodyData.qz^2);
        end

        %Determine locations of the cornerpoints of each AGV
        locX = AGV(ID).locationX(arrayIndex);
        locZ = AGV(ID).locationZ(arrayIndex);
        Yaw = AGV(ID).yaw(arrayIndex);
        AGV(ID).Corners = DetermineCorners(AGV(ID), locX, locZ, Yaw);

        %Make a closed rectangle for plotting
        AGV(ID).Rectangle = [AGV(ID).Corners AGV(ID).Corners(:, 1)];

    end

end
C.4 TrajectPrediction.m

```matlab
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Trajectory prediction algorithm
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function TrajectPrediction(arrayIndex,dT,ControlData)

  % Definition of variables
  global AGV

  % Predict the state of each AGV at t+predictionHorizon
  for ID = 1 : ControlData.nRigids
    if arrayIndex == 1 % First prediction
      % First prediction = initial location
      AGV(ID).xPred(1:ControlData.nStepsCDR,arrayIndex) = AGV(ID).locationX(arrayIndex);
      AGV(ID).zPred(1:ControlData.nStepsCDR,arrayIndex) = AGV(ID).locationZ(arrayIndex);
      AGV(ID).yawPred(1:ControlData.nStepsCDR,arrayIndex) = AGV(ID).yaw(arrayIndex);
      % Put CornersPred in a 3D array
      for j = 1 : ControlData.nStepsCDR
        AGV(ID).CornersPred(:,:,j) = AGV(ID).Corners;
      end

      % Make closed rectangle for plotting the AGV
      AGV(ID).RectanglePred = [AGV(ID).CornersPred(:,:,end) AGV(ID).CornersPred(:,:,1,end)];

      % Plot initial predicted location
      AGV(ID).PlotPredLoc = plot(AGV(ID).xPred(:,arrayIndex), AGV(ID).zPred(:,arrayIndex),'+-');
    else % Update location
      set(AGV(ID).PlotLoc,'XData',AGV(ID).locationX,'YData',AGV(ID).locationZ)
      set(AGV(ID).PlotRect,'XData',AGV(ID).Rectangle(1,:), 'YData',AGV(ID).Rectangle(2,:))
    end
  end
end
```
AGV(ID).PlotPredRect = plot(AGV(ID).RectanglePred(1,:), AGV(ID).RectanglePred(2,:));

else % Extrapolate current velocity vector
  % Increments since last measurement
  dx = AGV(ID).locationX(arrayIndex) - AGV(ID).locationX(arrayIndex - 1);
  dz = AGV(ID).locationZ(arrayIndex) - AGV(ID).locationZ(arrayIndex - 1);
  dYaw = wrapToPi(AGV(ID).yaw(arrayIndex) - AGV(ID).yaw(arrayIndex - 1));

  % Discretize the future trajectory in nSteps and extrapolate the
  % current velocity vector over these steps
  dxPred = dx / dT * ControlData.CDRstepTimes;
  dzPred = dz / dT * ControlData.CDRstepTimes;
  dYawPred = dYaw / dT * ControlData.CDRstepTimes;

  AGV(ID).xPred(:,arrayIndex) = AGV(ID).locationX(:,arrayIndex) + dxPred;
  AGV(ID).zPred(:,arrayIndex) = AGV(ID).locationZ(:,arrayIndex) + dzPred;
  AGV(ID).yawPred(:,arrayIndex) = wrapToPi(AGV(ID).yaw(:,arrayIndex) + dYawPred);

  % Calculate the coordinates of the cornerpoints for every
  % predicted location and put them in a 3D array
  for i = 1 : ControlData.nStepsCDR
    locX = AGV(ID).xPred(i,arrayIndex);
    locZ = AGV(ID).zPred(i,arrayIndex);
    Yaw = AGV(ID).yawPred(i,arrayIndex);
    AGV(ID).CornersPred(:,i) = DetermineCorners(AGV(ID),locX,locZ,Yaw);
  end

  % Make closed rectangle for plotting the AGV at the predicted
  % location at Tnow + ControlData.predHor. Intermediate
  % predictions are not plotted.
  AGV(ID).RectanglePred = [AGV(ID).CornersPred(:,end) AGV(ID).CornersPred(:,1,end)];

  % Update predicted location plot
  set(AGV(ID).PlotPredLoc,'XData',AGV(ID).xPred(:,arrayIndex),'YData',AGV(ID).zPred(:,arrayIndex))
function ConflictDetection(ControlData)
    % Define variables
    global AGV

    % Determine all possible combinations of pairs of rigidbodies
    IDs = 1 : ControlData.nRigidBodies;
    Combi = nchoosek(IDs,2);

    % Check collision between every possible combination of two rigidbodies
    for i = 1 : size(Combi,1)
        ID1 = Combi(i,1);
        ID2 = Combi(i,2);

        % Detect collisions for all predicted steps
        for j = 1 : ControlData.nStepsCDR
            overlap = CheckPolygOverlap(AGV(ID1).CornersPred(:, :, j),AGV(ID2).CornersPred(:, :, j));

            if overlap == true
                % Start conflict resolution if a collision is detected
                TTC = ControlData.CDRstepTimes(j);
% Start conflict resolution if is required from UserParameters.m and the TTC <= TTClimit
if ControlData.ConflictResolution == true && TTC <= ControlData.TTClimit
    fprintf('Conflict detected between %d and %d in %.2f seconds\n',ID1,ID2,round(TTC,2))
    ConflictResolution(ID1,ID2,ControlData)
    return
end
end
end
end
end

C.6 CheckPolygOverlap.m

function [overlap] = CheckPolygOverlap(PolygA,PolygB)
    overlap = true;

    nEdgesA = size(PolygA,2);
    nEdgesB = size(PolygB,2);

    % Check overlap along polygon A
    % Extend polygA such that a closed loop is possible
    extPolygA = [PolygA PolygA(:,1)];

    % Check for overlap projected to normal vectors of the edges of polygA
    for i = 1 : nEdgesA
        % Find the i-th edge vector and the left normal vector of it
```matlab
edgeVect = extPolygA(:,i+1) - extPolygA(:,i);
normVect = [-edgeVect(2,1); edgeVect(1,1)];
% Note: As this script only computes a true/false for overlap,
% normVect doesn't have to be a unit vector.

% Project all cornerpoints of polygA onto normVect
A = zeros(1,nEdgesA);  % Preallocate for speed
for j = 1 : nEdgesA
    A(j) = dot(PolygA(:,j),normVect);
end

% Project all cornerpoints of polygB onto normVect
B = zeros(1,nEdgesB);
for k = 1 : nEdgesB
    B(k) = dot(PolygB(:,k),normVect);
end

% Check for overlap
if min(B) > max(A) || min(A) > max(B)
    overlap = false;
    return % Guaranteed no overlap if one gap has been found
end

% Check overlap along polygon B
% Extend polygB such that a closed loop is possible
extPolygB = [PolygB PolygB(:,1)];

% Check for overlap projected to normal vectors of the edges of polygB
for i = 1 : nEdgesB
    % Find the i-th edge vector and the left normal vector of it
    edgeVect = extPolygB(:,i+1) - extPolygB(:,i);
    normVect = [-edgeVect(2,1); edgeVect(1,1)];
    % Note: As this script only computes a true/false for overlap,
    % normVect doesn't have to be a unit vector.
    % Project all cornerpoints of polygA onto normVect
    A = zeros(1,nEdgesA);
    for j = 1 : nEdgesA
        A(j) = dot(PolygA(:,j),normVect);
    end
```

---

Master of Science Research Assignment  T.D. Hogenboom
% Project all cornerpoints of polygB to normVect
B = zeros(1, nEdgesB);
for k = 1 : nEdgesB
    B(k) = dot(PolygB(:,k), normVect);
end

% Check for overlap
if min(B) > max(A) || min(A) > max(B)
    overlap = false;
    return
end

end

C.7 ConflictResolution.m

function ConflictResolution(ID1, ID2, ControlData)
   global AGV
   command = '<1500a>1';
   ControlData.serialPort.sendCommand(command, AGV(ID1).adress);
   ControlData.serialPort.sendCommand(command, AGV(ID2).adress);
end