Delft University of Technology

FACULTY MECHANICAL, MARITIME AND MATERIALS ENGINEERING
Department Marine and Transport Technology

Mekelweg 2
2628 CD Delft
the Netherlands
Phone +31 (0)15-2782889
Fax +31 (0)15-2781397
www.mtt.tudelft.nl

Specialization:	Transport Engineering and Logistics
Report number:	2015.TEL.7946
Title:	**On calibration of model AGVs**
Author:	T.J.W. Bentvelsen

Title (in Dutch)	Over kalibratie van model AGVs

Assignment:	Research
Confidential:	No
Initiator (university):	Ir. M.B. Duinkerken
Initiator (company):	-
Supervisor:	Ir. M.B. Duinkerken
Date:	23rd of June, 2015

Delft University of Technology

FACULTY OF MECHANICAL, MARITIME AND MATERIALS ENGINEERING
Department of Marine and Transport Technology

Mekelweg 2
2628 CD Delft
the Netherlands
Phone +31 (0)15-2782889
Fax +31 (0)15-2781397
www.mtt.tudelft.nl

Student:	T.J.W. Bentvelsen	Assignment type:	Research
Supervisor (TUD):	Ir. M.B. Duinkerken	Creditpoints (EC):	15
Supervisor (Company):	-	Specialization:	TEL
		Report number:	2015.TEL.7946
		Confidential:	No

Subject: **Calibration of sensors and actuators on AGVs**

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 equiped with servo's for speed control and (independent) control of both wheel axes and an Arduino controller board.

During setup of the Automated Guided Vehicle (AGV) laboratory at the Mechanical Engineering department of the Delft University of Technology, it was noticed that a deviation between the desired location and the actual location of an AGV occurred after performing a Dubins trajectory. Before the test, the AGV was calibrated manually. This shows that manual calibration of the electronic actuators is not sufficient and above that a time consuming process.

In order to eliminate manual calibration from the setup of the laboratory, an automatic calibration method should be proposed. This calibration method must be based on available literature which can be acquired and compared to a manually calibrated AGV. During the assignment, two questions are of particular interest:

- Which methods are available for the automatic calibration of AGV's?
- How can this be integrated with the current AGV laboratory setup?

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

Preface

After completing the literature study on container terminals, my interest in port operations remained. As luck would have it, the department Transport Engineering and Logistics (TEL) at that time had recently purchased a motion capture systems for the Automated Guided Vehicle (AGV) laboratory. With the purchasing of this system, several new research assignments became available. One of the assignments, the calibration of AGVs with a motion capture system triggered my interest.

The final deliverable of the research assignment, the report you are reading, contains the proceedings and conclusion to the calibration of model AGVs. The first part of this research investigates what has been written on calibration of mobile robots in literature. This is then used to determine which approach will be used for the calibration of model AGVs. The implementation of this approach is then discussed, after which tests are performed to determine the feasibility of calibration on the AGVs.

As a final comment, I would like to express my gratitude to Ir. M.B. Duinkerken and N.P. Schoorl for their time. Mr. Duinkerken has thoughtfully read this report and commented on parts that could be added or improved. During the assignment, Nick and I have had many discussions about the subject. Without our discussions, this research would not be as comprehensive as it currently is. With this preface, I would like to express my gratitude to both Nick and Mr. Duinkerken for their help and insights.

Ted Bentvelsen
June 21, 2015

Abstract

The research of which this is the final report, focusses on the calibration of model AGVs. In previous assignments it is observed that there is a deviation between the desired and actual trajectory of AGVs. As the positioning of AGVs is preferably estimated with data from its sensors, the deviation between the servo input and output must be minimized. This can be achieved with calibration.

From literature is obtained that calibration of mobile robots with an external observer is best performed by establishing a kinematic model. The kinematic model of AGVs can be approximated with a bicycle model as is stated in Wang and Qi (2001). With the use of several equations and observations by a motion capture system, the input and output for the steering angle and the velocity can be related.

After performing several tests with an AGV having been calibrated and uncalibrated, the results showed that in most cases the accuracy of the AGV improved. In some cases however, the calibrated AGV was slightly less accurate than uncalibrated AGV. This is an unexpected result and there is no conclusion as to why this happens. One hypothesis which may explain this behaviour is that the servo controller works at a specific frequency and thus not always performs a command instantaneous.

It is recommended to have this new hypothesis investigated further. Also, during the testing of model AGVs it appeared that there sometimes is a large spread between the results of individual tests. It is suspected that this is caused by a cumulative tolerance between different parts. It would be of interest to investigate the effect of tolerance reduction on the spread of the results.

Abstract (Dutch)

Het gedane onderzoek waarvan dit het afrondende rapport is, kijkt naar de kalibratie van model AGVs. In voorafgaande opdrachten is opgevallen dat er een afwijking bestaat tussen de gewenste en gerealiseerde trajecten van AGVs. Sinds de positie bepaling van AGVs bij voorkeur wordt gedaan op basis van sensor data, kan de afwijking tussen de actuator invoer en uitvoer heb beste worden geminimaliseerd. Dit kan worden bereikt door middel van kalibratie.

Uit literatuur is verkregen dat kalibratie van mobiele robots aan de hand van een extern observatie systeem het beste kan worden gedaan met een kinematisch model. Het kinematische model van AGVs kan worden benaderd met het fiets model, zoals wordt aangegeven in Wang and Qi (2001). Door gebruik te maken van enkele vergelijkingen, kan de invoer en uitvoer voor zowel de stuur hoek als de snelheid aan elkaar worden gerelateerd.

Na het doen van enkele tests met een AGV in gekalibreerde en ongekalibreerde toestand, toonden de resultaten dat in de meeste gevallen de gekalibreerde AGV een grotere nauwkeurigheid gaf. In enkele gevallen gaf kalibratie een slechter resultaat. Dit is een onverwacht resultaat en er is geen conclusie over de oorzaak hiervan. Een hypothese voor dit gedrag kan zijn dat de controller op een specifieke frequentie werkt en zo dus niet altijd meteen een commando zal uitvoeren.

Het wordt aangeraden om deze nieuwe hypothese verder te onderzoeken. Daarnaast werd tijdens het testen van de AGV een grote spreiding tussen de resultaten van individuele testen geobserveerd. De verwachting is dat dit wordt veroorzaakt door een cumulatieve speling van verschillende onderdelen in de AGV. Het is daarom interessant om te onderzoeken welk effect deze tolerantie op de resultaten heeft.

Nomenclature

AGV	Automated Guided Vehicle
COM	Center Of Mass
DEM	Discrete Elements Modelling
IMU	Inertial Measurement Unit
IR	Infrared
SDK	Software Development Kit
TEL	Transport Engineering and Logistics

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the renewed Automated Guided Vehicle laboratory at the Delft University, one of the requirements for model AGVs is to accurately follow prescribed trajectories. The overall objective of the system is to simulate the operation on a seaport container terminal. As a multi-agent control system, it is desired that the positioning of AGVs is mainly based on internal sensors that register motion. The position is estimated with the data from the sensors. This is called odometry. It is desired to have an external observer correct the AGVs as little as possible.

Several attempts have been made to increase the accuracy of path following AGVs by using additional sensors for odometry. In Gerritse (2014), the positioning of models is improved with an accelerometer. Further research is performed into the use of an inertial measurement unit (IMU). However there exists a deviation between the desired and the realized steering angle as well as the perceived and actual velocity. This difference is partially caused by the incorrect assembly of the drive mechanism but may have other causes as well.

To compensate for these deviations, a calibration process for the model AGVs will be developed in this research. The project is guided by the following main research question:

> *Can the accuracy of model AGVs be improved with the calibration of actuators?*

The AGV's accuracy will be measured as the difference between the desired and the actual end-position after performing a trajectory. The approach for following such a trajectory will be similar to the algorithm used in de Groot (2015), in which Dubins-trajectories are calculated and performed.

The contents of the research assignment will be partitioned into chapters of this report according to the following sub-questions:

- What is the laboratory set-up and equipment?
- Which method is suited for the calibration of mobile robots?
- How will the calibration method be implemented for the calibration of model AGVs?
- What are the results for the model AGV's calibration?
- Is a calibrated AGV more accurate than an uncalibrated AGV?

These sub-questions will be answered in the chapters according to this structure: chapter 2 elaborates the methodological approach that will be used throughout this report. This includes the set-up of the independent observer, the literature survey approach and approach for performing the implementation and experiments. Hereafter the laboratory set-up, containing the model AGVs, environment, computational system and assumptions are discussed in chapter 3. To gain knowledge on calibration and its application on mobile robots, a literature survey is performed in chapter 4.

This knowledge is then used for the explanation of the kinematic model of AGVs and the implementation of it in chapter 5 and chapter 6 respectively. The implemented calibration procedure is used in chapter 7 to calibrate a single AGV for the purpose of testing. In chapter 8 the results to these tests are depicted, of which the conclusion is presented in chapter 9. This final chapter also discusses the results and provides recommendations for future research.

Chapter 2

Methodology

This chapter clarifies the methodical approach that will be used during this research. The method for systematically setting up the external observer system is discussed in section 2.1. This is later used to analyse the effect of calibration in chapter 8.

Before implementing a calibration method, a survey is conducted regarding the literature of available calibration methods for autonomous vehicles. Articles are found according to the approach described in section 2.2. The literature survey will provide a guideline for the implementation of a calibration method, which is elaborated upon in chapter 6. In section 2.3 is discussed how the implementation, experimentation and evaluation of results will occur.

2.1 Independent observer

For the calibration of model AGVs, an external observer will be used to capture the position and rotation. This independent observer is required to produce an identical coordinate system each time it is set-up and calibrated. The proposed OptiTrack Motion Capture system, elaborated in chapter 3, is able to achieve this requirement.

A methodical approach for setting up and calibrating the OptiTrack system is presented in Appendix section D.1 and Appendix section D.2. This will ensure that the results presented in chapter 8 are comparable, even if the independent observer is displaced between tests.

Table 2.1: Results from literature survey

	TU Delft Library	Web of Science	Science Direct	SpringerLink
Results	130	349	1239	242
Of interest	1	4	24	12

2.2 Survey approach

In order to find which methods are available for the calibration of model AGVs, a literature survey is performed. This study will be presented in chapter 4. Several sources such as the TU Delft Library, Web of Science, Science Direct and SpringerLink are consulted for publications. For the search, a combination of the words: *self*, automat*, calibrat*, mobile*, robot*, servo** and *sensor** is used.

The results from this literature survey are presented in Table 2.1. As it is not possible to read every single article, a strict selection procedure was applied to filter which articles are of interest. After reading the abstract of an article, it was decided to keep or discard the article. Every article that was kept is marked as "of interest" in Table 2.1. These articles are then fully read for relevant information and presented in chapter 4. Out of all the interesting articles, one article was unobtainable due to confidentiality.

To categorize the found calibration methods and expand the scope further, a search for articles containing both *calibrat** and *method** is performed. This yielded for TU Delft Library, Web of Science, Science Direct and SpringerLink, a total of 2529, 839, 348 and 3976 articles respectively. As the results on these websites are sorted on relevance, only the first 100 entries are evaluated similar to the previous search. From this selection, 15 articles are considered interesting, based on the title and abstract. However, a total of six articles could not be obtained due to confidentiality. The available information is further discussed in chapter 4.

2.3 Implementation and experiments

Literature regarding calibration approaches yields information about which method is best applied in the case of model AGVs. In chapter 6 is elaborated how the implementation occurs. Hereafter, tests will be carried out to determine the effect of calibration on the vehicles. The calibration method is implementable for every model AGV, however the experiments will be conducted with a single vehicle to eliminate the build quality differences between various models.

The tests to determine if calibration has benefits, consist of the AGV completing several trajectories with both initial values and calibrated values. Deviation between predicted final position and actual final position of a trajectory is denoted in two horizontal plane coordinates and the orientation of the vehicle.

A schematic of the experimental procedure is presented in Figure 2.1. After calibration of a specific model AGV, the model is returned to its initial, uncalibrated values. A trajectory is then performed and repeated 10 times. The same trajectory is performed but now with the calibrated servos. Hereafter, the AGV is again set to its initial values and the test is repeated for a different trajectory. The test ends when all trajectories are performed 10 times for both a calibrated and an uncalibrated AGV.

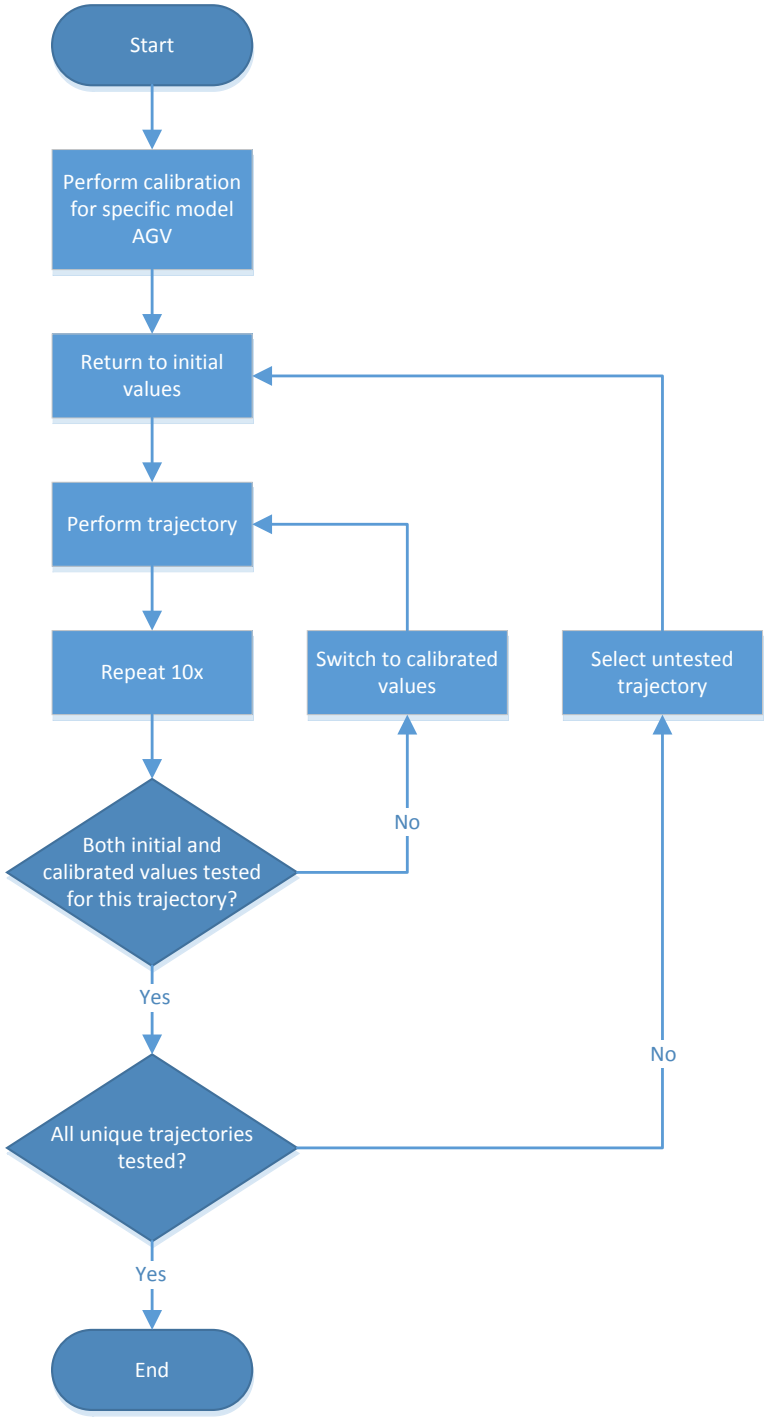


Figure 2.1: A flowchart depicting the experimentation procedure that will be used to analyse the effects of actuator calibration on model AGVs.

Chapter 3

Laboratory and set-up

To make this calibration experiment transparent and repeatable, a detailed model description and experiment set-up are given in this chapter. To assure correctness of the method and applicability, the assumptions that are made for the model and the observing system are discussed in section 3.2. The model itself, the observer and the computational system are discussed in section 3.1.

3.1 Experiment set-up

In this section, the different materials and equipment for the calibration of model AGVs are discussed. The first sections discusses the model AGVs, whereafter the environment is elaborated. Finally, the computational system is discussed.

3.1.1 Model AGVs

The vehicles that will be used in this research assignment are model AGVs. These scale models of actual AGVs have been used to simulate container terminal operations previously. The current system is operated by an Arduino microprocessor which is depicted in Figure 3.1.

A more detailed description of the system is given in van Blijswijk (2015) and Gerritse (2014). The latter additionally gives a kinematic model of the AGV that are required for deadreckoning.

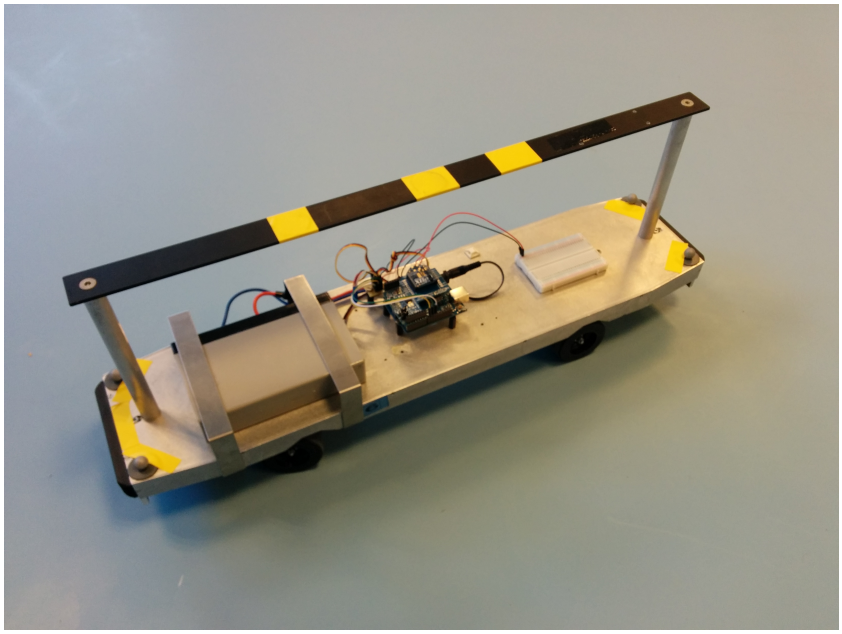


Figure 3.1: One of the 25 model AGV's with mounted Arduino processor, breadboard and battery.

3.1.2 Environment

The model AGVs are operated and tested at the faculty of Mechanical Engineering at the Delft University of Technology. For the tracking of the AGVs, a system with "plug-and-play" characteristics has been purchased. It was decided that the NaturalPoint Inc Motion Capture system with Flex13 cameras would achieve this goal.

The system may be set-up with 4 to 24 cameras, each additional camera increasing the maximum size and accuracy of the system. According to the NaturalPoint Inc website, the capture volume can encompass a 6,1 diameter circle when using six cameras. This is suitable for the dimensions of the workspace depicted in Figure 3.2.

The listed materials are required for setting up an OptiTrack system with six cameras:

6 OptiTrack Flex13 cameras
6 tripods
6 tripod mounting pieces
6 camera-to-hub data cables
2 OptiTrack OptiHubs
2 power adapters for OptiHub
2 hub-to-computer data cables
2 data cable extenders
1 hub synchronization cable

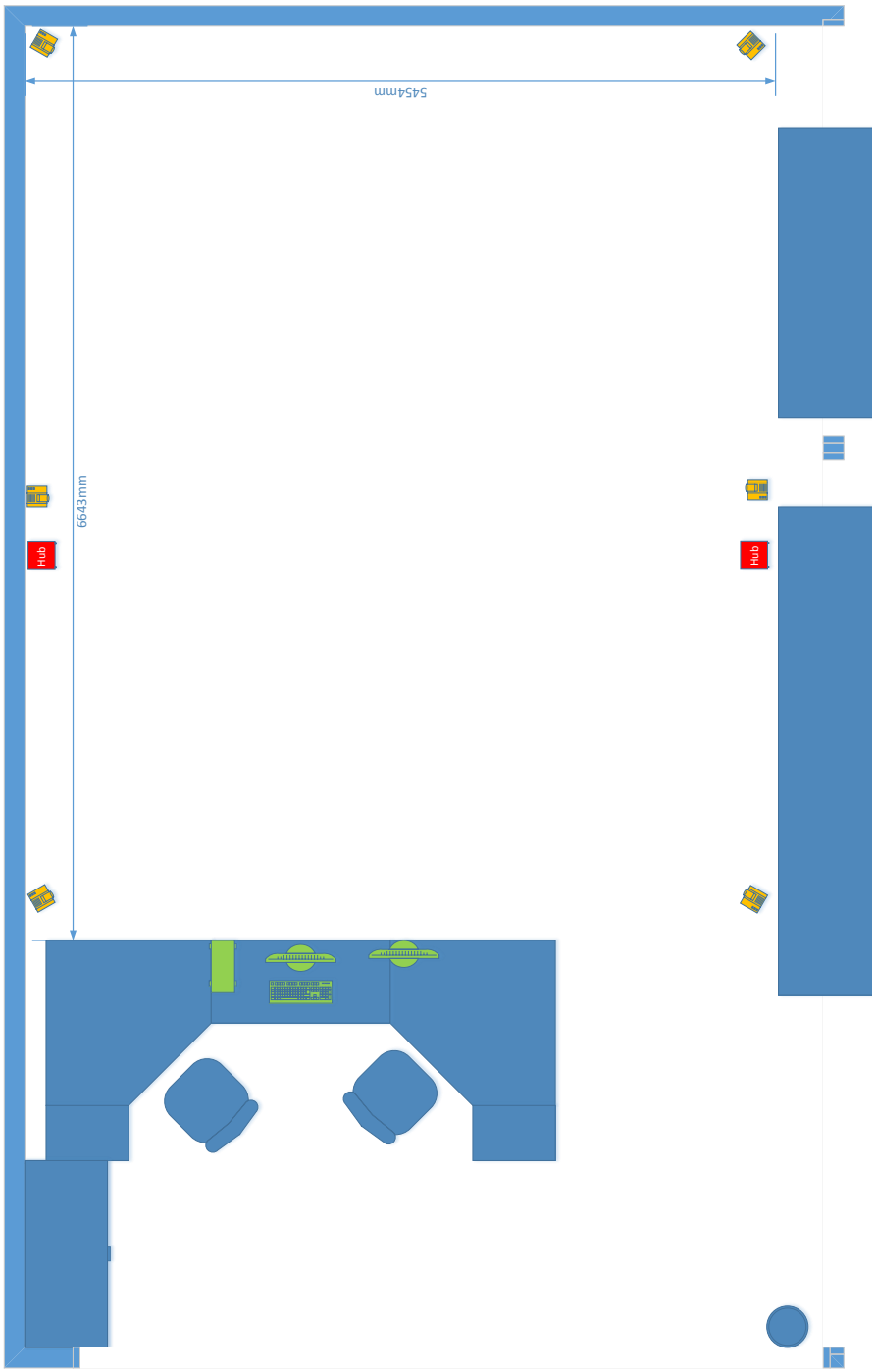


Figure 3.2: Dimensions of the workspace for the AGV laboratory. Tables and other office equipment is depicted in blue, the computer system is depicted in green, the OptiTrack Flex13 cameras are depicted in orange and the OptiHub is depicted in red.

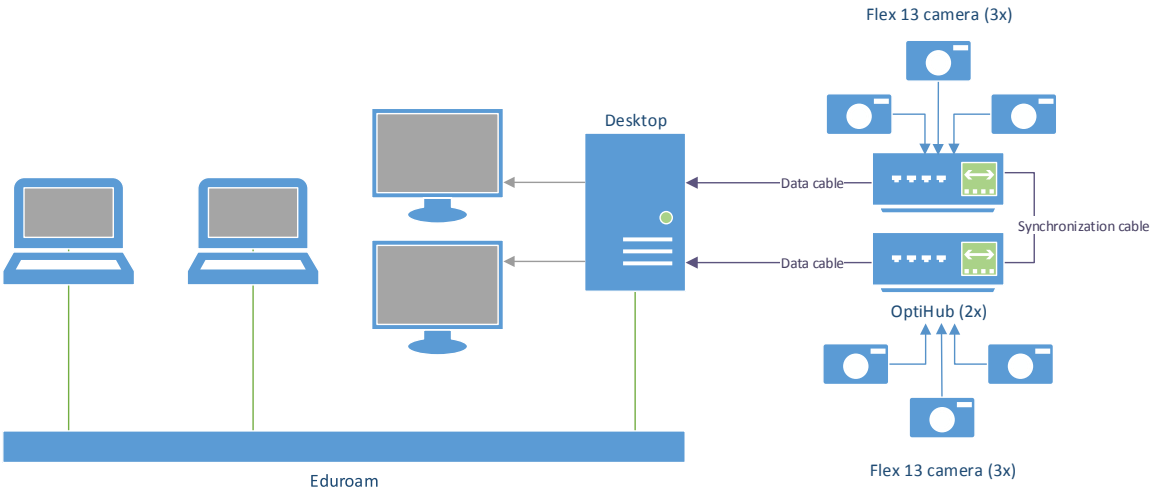


Figure 3.3: Schematic representation of the experiment set-up. The lines depict cables and wireless connections. Arrows show the flow of information from the cameras to the computational system.

A schematic representation of the system is depicted in Figure 3.3. The instructions for installing the system are present in Appendix D.1.

3.1.3 Computational system

Software provided by the OptiTrack manufacturer is installed on the computer and processes the camera data into a three dimensional image of the workspace. However before this image can be constructed, the system needs to be calibrated. Instructions for the calibration can be found in Appendix D.2 and need to be repeated every time one or more cameras are displaced.

The system is set-up such that the gathered position data is broadcast over the local area network. By doing this, multiple experiments can take place at the same time. In Appendix D.3, the instructions for establishing a connection between OptiTrack and Matlab are presented.

3.2 Assumptions

In the course of this research, several assumptions are made that simplify the model. To ensure correctness of the approach, it is important that these are mentioned. The list provides an overview of the assumptions that are made, and these are substantiated later.

- Model AGV
    1. The AGVs can be modelled according to the bicycle model
    2. The slip of the models wheels is negligible
    3. The varying size of wheels is negligible
    4. The front axle is equal to the rear axle velocity of the model AGV
    5. The effects of acceleration and deceleration are negligible
    6. The Kinematic model from Gerritse (2014) is sufficiently accurate
    7. The performance dependency on battery power is negligible
- OptiTrack system
    1. The motion tracking system does not drift
    2. The accuracy of the OptiTrack system is within 10 millimetres

3.2.1 Model AGV

The Kinematic model of the scale AGVs is elaborated in Gerritse (2014). This report substantiated why wheels slip and size are negligible. It is assumed that this model is correct for the purpose of calibration, as no significant changes have occurred since.

One major obstacle in the calibration of the AGVs is that the battery power could have an impact on the performance of the servo motors that actuate the model. To verify that this is indeed the case, an endurance test is performed with the model AGV and a full battery. The AGV is instructed to drive circles in the test area at maximum speed. The results are elaborated in section 8.1.

3.2.2 OptiTrack system

The manufacturer of the OptiTrack system provides information on the accuracy of the system and software. When properly set-up, the only variable in the marker positioning is the measurement error. Depending on the quality of the camera calibration and amount of cameras, an error of 0,10 - 2,0 mm/marker can be observed. The AGV models are set up to have a total of five marker, of which a minimum of three is required to recognise the unique vehicle. This results in a maximum error of 6 millimetre and is well within the set 10 millimetre boundary.

Chapter 4

Calibration methods

This chapter elaborates on the literature that is available for calibration. Calibration is used to compare the magnitude of a sensor or the motion of an actuator with a known magnitude or motion. Many robotic devices use sensors and actuators to control the robot. To comprehend how calibration affects these, a robotic arm is used as an example. This is then extended to the model AGVs.

The robotic arm, similar to the one proposed in Duelen and Schröer (1991), may be used in manufacturing environments. The arm has multiple joints and these joints are powered by actuators. The actuators rotate according to an electric signal, but cannot derive the arm's position from this signal. It is therefore required to compare the magnitude of the electric signal with the actual rotation of the arm. This rotation may be subject to an offset, linear behaviour or even non-linear behaviour. During calibration, this behaviour is analysed and processed into a correct signal for the actuators.

Calibration works as well for calibration of sensors and is not confined to robotic arms. It is possible, even necessary, to calibrate any actuator and sensor. In the case of mobile robots, like the model AGVs used in the laboratory setup, the assembly of individual vehicles can cause minor deviations between them. Calibration may reduce these deviations, in the case of model AGVs the steering angle and actual velocity. It is however appropriate to consult about established calibration procedure before a method is conceived.

In section 2.2 is described how the literature survey for calibration methods is performed and filtered. A selection of the resulting articles is represented in this chapter. Several definitions are discussed in section 4.1, whereafter section 4.2 and section 4.3 elaborate the found calibration approaches. Issues that occur during calibration approaches are discussed in section 4.4. The gained knowledge is then used to define and implement the calibration approach for this report which is discussed in chapter 5 and chapter 6

4.1 Definitions

In order to ensure that calibration is well understood and no misunderstandings occur, this section elaborates on the definitions of several words. In Rykiel (1996), the definitions for *model, calibration, verification* and *validation* are discussed. This publication focuses mainly on the practise of Discrete Element Modelling (DEM) but is also applicable to other fields.

A publication by Merlet (2006) discusses the calibration of robotic arms. The model, calibration, verification and validation are defined similarly to Rykiel (1996). In addition, the terms *domain* and *poses* are elaborated.

Model A model can represent different virtual or physical objects. It often serves as simplification for a selected part of reality. Computer simulations, robotic equipment or dynamics, among others, are examples of subjects that are modelled frequently. The model is then used to predict future behaviour of the object.

Domain A model is only applicable in the domain that is prescribed by the made assumptions. Recalling the robot arm example, a calibration method for a 2-joint robot arm is not applicable to a 3-joint robot arm. The assumptions are related to the model and influence the domain of its applicability.

Calibration According to Rykiel (1996), a general definition for calibration is the estimation and adjustment of model parameters and constants to improve the agreement between model output and a data set. Applying this to the case of mobile robots would imply minimizing the deviation between the presumed and the actual position of each actuator. The predicted trajectory of the AGV is defined as model output, whereas the actual trajectory is defined as the data set. Calibration of the actuators may lead to an agreement between this model output and data set.

Verification By verifying the model, it is demonstrated that the model itself is correctly formulated. This is best represented by asking oneself: “Is the model right?”. Improper formulation of the model may be caused by incorrect calculations or logical steps. Certain errors only appear under specific circumstances and may not have been anticipated. Verification is therefore important before the model is used.

Validation Contrary to verification, validation of a model is concerned with the applicability of the model to the intended domain. It is therefore important to ask: “Is it the right model?". As the model represents a part of reality, it is necessary to demonstrate that the model is applicable in the set domain and that the range of accuracy is consistent with the intended application of the model.

Poses Calibration may require a subject to assume poses. This means that the subject attains a certain position and/or rotation. Differing poses may help improve the accuracy of a calibration.

4.2 Calibration approaches

In this section, several calibration approaches that are found in literature are discussed. The focus is put on the calibration of robots with servos as the knowledge will be used to produce a calibration approach for model AGVs in chapter 6. According to Merlet (2006), three different types of calibration approaches can typically be distinguished. These types are:

- External calibration
- Constrained calibration
- Self-calibration

External calibration methods use external equipment to partially or fully measure the poses of the subject in order to calibrate it. An example of such a method is detecting mobile robots with an external camera system. These methods are considered to be expensive and lengthy, as external equipment must be precise to achieve decent results. Robots are required to be off-line, not performing any tasks, when calibrated externally and need to be in the workspace of the external observer during calibration.

Methods concerning constrained calibration rely on a system that constrains the robots motion during the calibration process. Similar to external calibration methods, constrained calibration is a lengthy process, must be performed off-line and can only be performed in the workspace that limited by the system that constrains. Constrained calibration is however considered to be the least expensive calibration method. Forcing a robotic arm into predefined positions and measuring the input variables for such a pose is an illustration of constrained calibration.

With the coming of smaller and higher precision sensors, self-calibration of robots has become feasible. The robot carries internal sensors with it that can determine the poses in several or all degrees of freedom. For a full calibration, it is required to have every degree of freedom of the robot covered by at least one sensor. An example of self calibration is the calibration of a servo based on a rotary encoder. This can be performed while the robot is in normal operation and is possible in the entire workspace. Self-calibration is relatively low cost, but it may happen that not all parameters can be calibrated.

4.3 External calibration in literature

This research is limited by the materials that are available. In chapter 3 is discussed that an OptiTrack camera system is present as independent observer. The following literature is therefore focussed on external calibration of mobile robots.

A total of three articles that are found describe external calibration procedures which use kinematic models. The three articles are Varziri and Notash (2007), Duelen and Schröer

(1991) and Lee et al. (2010). A kinematic model portrays the behaviour of the vehicle with mathematical equations. These equations are derived from the object's kinematics and contain unknown variables. The variables may then be estimated based on recorded movements of the vehicle. Taking the case of a model AGV as an example, the kinematic model would output the position of the AGV based on its previous position, the actuator inputs and the passed time. By using the kinematic model in reverse, it becomes possible to determine the model inputs based on the observed output. This can then be used to compare the presumed and observed actuator positions for calibration.

In Varziri and Notash (2007) the Gauss-Newton and Levenberg-Marquardt methods are compared to identify the parameters of the kinematic model. The established kinematic model is complex and requires multiple iterations to solve the unknown parameters. These iterations are performed with the previously mentioned Gauss-Newton and Levenberg-Marquardt methods. It is concluded that both methods provide a sufficient improvement for calibration.

Duelen and Schröer (1991) develop a continuous differential equation to describe the behaviour of a robot arm. The robot arm is modelled on a computer based on its kinematic behaviour. A theodolite system, which measures angles in horizontal and vertical planes, is used to observe the robots spatial coordinates. The input of the robot is then compared with the spatial coordinate to determine the kinematic model parameters.

Lee et al. (2010) address the kinematic model of a car-like robot in order to improve the position accuracy. In the kinematic model, significant changes in wheel diameter and slip are taken into account. The robot is driven by an open loop controller which is instructed to perform predesignated trajectories. The planned and actual trajectory are compared and calibration values could be retrieved.

An alternate notation for the kinematic model is performed in the form of an invariant Jacobian in Batlle et al. (2010). The mathematical model is presented in matrix form and constructed for a three-degree of freedom vehicle. Performed tests are used to estimate the calibration parameters.

Different approaches are used in (Khalil and Dombre, 2002). The chapter describes both the simple, the end-effector coordinates and distance measurement calibration approach. In the simple approach, the calibration is performed through a trial-and-error guessing process. This approach can be improved upon by using the end-effector coordinates of a robot to limit the guessing process. Similar to end-effector coordinates, the distance measurement uses coordinate data but takes more data into consideration. End-effector coordinates and distance measurement are however typically used in robotic arms. Application to mobile robots is not documented in these publications.

Several other studies involving cameras and robot calibration perform the tests inverted. In Chen et al. (2007), an environmental camera system is calibrated by using a mobile robot. After calibration of the cameras, the system is used for the odometry of the mobile robot.

To provide a quick overview of the relevant calibration approaches mentioned in literature, the following list is constructed from the approaches that were discussed earlier in this section. The listed methods are complemented by the relevant article.

- Estimating the calibration parameters - (Khalil and Dombre, 2002)
  - Using end-effector coordinates - (Khalil and Dombre, 2002)
  - Using distance measurement - (Khalil and Dombre, 2002)
- Constructing a kinematic model - (Varziri and Notash, 2007; Lee et al., 2010)
  - Using an invariant Jacobian - (Batlle et al., 2010)

4.4 Calibration issues

When performing a calibration process, Merlet (2006) argues that several issues are of importance. These issues should be resolved by the applied calibration approach, or the effect should at least be considered by the user.

Observability of parameters For the calibration of robots it is important to identify all the parameters that influence the motion. It may not always be possible to identify every parameter with a specific calibration method and it is therefore important to consider this before selecting a calibration approach. It can also occur that a specified parameter has no influence on the calibration subject or that the parameter is affected by a combination of other parameters.

Sensitivity of measurements Estimation of parameters is affected by the precision of the instruments and the generated sensor noise. Data from each measurement experiences an uncertainty and this needs to considered in the calibration process by repetition.

Accuracy of calibration There is no standard available for measuring and indexing the accuracy of a calibration procedure. To provide objective data, an indication of accuracy should be determined before implementing and performing calibration procedures.

Choice of calibration poses The accuracy of a calibration is influenced by the the diversity and quantity of calibration poses. Increasing the amount of poses gives a larger data set which can be analysed. Additionally, more diverse poses will allow for better interpolation of the data.

Unicity In previous part, the importance of having several different calibration poses is discussed. These poses are required to be unique in the sense that two differing poses should not have the same set of calibration parameters.

Chapter 5

Kinematics in theory

The experimental equipment and set-up, discussed in chapter 3, form a boundary condition for the possible calibration methods. Methods sufficing to these boundary conditions are presented in the literature survey that is conducted in chapter 4. From the literature it appears that kinematic models are typically used for the calibration of mobile robot systems similar to AGVs. As kinematic models are well established and understood, and a kinematic model is already available for the studied model AGVs, this chapter will elaborate on the theory behind a kinematic model for AGVs, to be used in the calibration thereof.

The calibration is necessary for the laboratory model AGVs because deviations between the perceived steering angle and actual steering angle occur. This is also the case with perceived and actual velocity. In order to improve the deadreckoning of the vehicles, it is necessary to minimize these deviations.

This chapter will be subdivided into several sections. In section 5.1, a general depiction of a kinematic model is given. This ensures that the purpose of it is clear before adopting the kinematic model for AGVs.

5.1 Description of a kinematic model

The concept of mobile robot kinematics is not new. It is already mentioned in Muir and Neuman (1986) and defined as the study of geometry of motion. For example the motion of a robot from the geometry of the constrained wheels. A modern definition is given by Ribeiro and Lima (2002), defining kinematics as the study of mathematics of motion without considering the forces that affect the motion.

A kinematic model thus describes the motion of a constrained object, neglecting the forces that act upon it. In the example of an AGV, the constraints can be defined as the maximum steering angle and maximum speed. There exists a relation between these that describes the motion of the AGV. For calibration purposes, the kinematic model can also be used to calculate the input based on the observed output.

Prior work on kinematic models for model AGVs has been performed in Gerritse (2014). In this report is discussed which factors influence the vehicles operation and are thus required to be modelled in order to predict the vehicles location. The model is constructed assuming that it behaves like a bicycle with equal wheel diameters and no slip. These assumptions are listed in chapter 3 and will be presumed true from this point.

In Gerritse (2014) the input for the kinematic model is defined the current position, the rotation of the front axle, the rear axle, the velocity of the front wheels, the velocity of the rear wheels and the passed time. The kinematic model gives a new perceived position, orientation and velocity as an output. This output is then taken as the input for the next iteration of this model repeatedly.

For the calibration of AGVs, this process has to be reversed. The output is measured by using an independent observer and is used to estimate the inputs, specifically the overall velocity, orientation and motion. The inputs which can then be estimated are the rotation of the front and rear axle, and the velocity of the vehicle.

5.2 Kinematics

In section 5.1 is explained what a kinematic model is, which inputs it requires and what it returns. For calibration, the inputs and outputs of the kinematic model are reversed to compare the perceived actuator performance with the actual actuator performance. With this comparison, the deviation between perceived and actual steering angles can be minimized as well as the difference in the perceived and actual velocity.

The mathematical equations for the kinematic model of the AGVs are subdivided into two parts. The first part modelling the rotational mechanism and the second part modelling the motion mechanism. Combining both parts leads to the kinematic model of the AGVs, discussed by Wang and Qi (2001). This is further elaborated in section 5.3.

5.2.1 Rotational kinematics

The most difficult part of calibrating model AGVs is the steering mechanism. A front-wheel steered vehicle modelled as a bicycle would have a turning radius similar as depicted in Figure 5.1. This is presented in Jazar (2008). The center of rotation is in line with the rear axle of the vehicle and the rotation of the front axle can easily be deduced by using trigonometry.

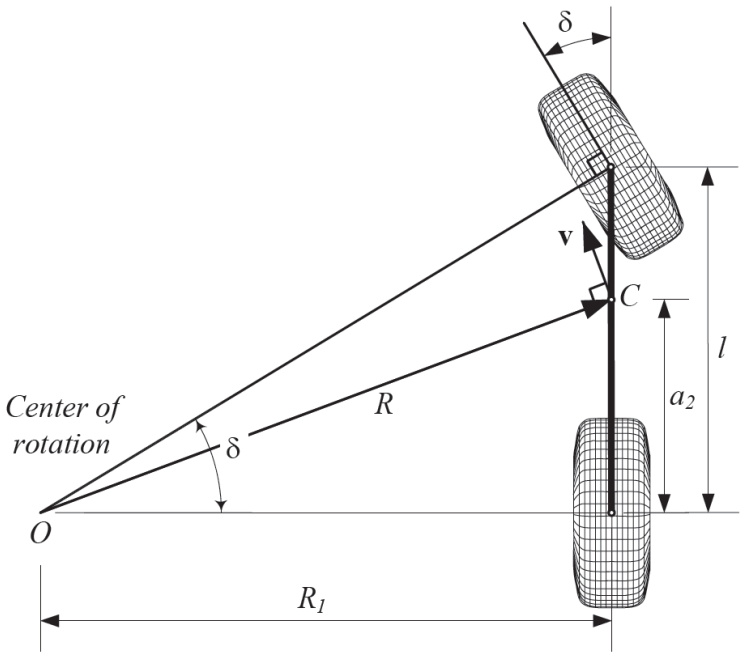


Figure 5.1: Depiction of a front-wheel steered bicycle model rotating around the center (Courtesy of Jazar (2008)).

Equation 5.1 may be used to calculate the rotation of the front or rear axle in the case of one fixed and one rotated axle. The subscripts f and r represent the front and rear axle respectively.

$$\delta = \arctan \frac{l}{R_1} \tag{5.1}$$

For the model AGVs, the rear axle can also be rotated. This introduces two additional steering options. These options are depicted in Figure 5.2 and Figure 5.3. Crab steer is excluded, as this would not result in an altering orientation and thus not return a center of rotation. The method for determining the axle angles will not hold then.

The two other cases can also be solved by using trigonometry. When the axles are rotated in a similar direction but with different magnitudes (see Figure 5.2), the angles may be calculated with Equation 5.2 and Equation 5.3.

$$\delta_f = \arctan \frac{a_1 + a_2 + c_2}{R_1} \tag{5.2}$$

$$\delta_r = \arctan \frac{c_2}{R_1} \tag{5.3}$$

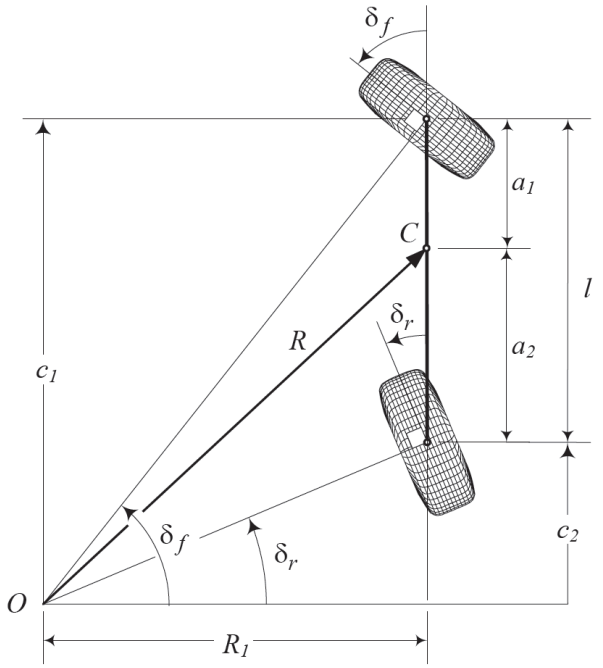


Figure 5.2: Illustration of axes rotating in a similar direction but with varying magnitude for a bicycle model (Courtesy of Jazar (2008)).

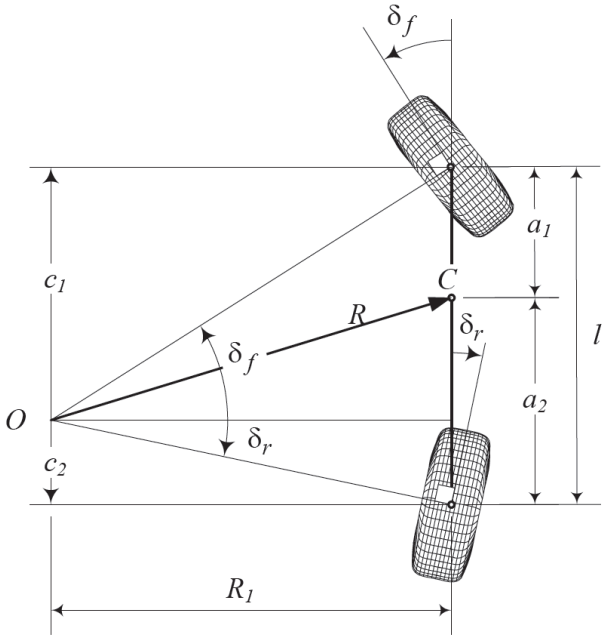


Figure 5.3: Opposite steering for a bicycle model, used to make tight turns (Courtesy of Jazar (2008)).

In the case of opposite steering wheels (see Figure 5.3), Equations 5.4 and 5.5 can be used to calculate the rotation angle of the axles. Both cases however require additional information of the vehicle. The implemented calibration method must be able to distinguish all cases to calculate the rotation of the front and rear axles.

$$\delta_f = \arctan \frac{c_1}{R_1} \tag{5.4}$$

$$\delta_r = \arctan \frac{c_2}{R_1} \tag{5.5}$$

5.2.2 Motion kinematics

As is explained in section 5.1, the model AGV is limited by the kinematic constrains of the vehicle. The length from center to front (l_f), the length from center to rear (l_r), the rotation of the front (δ_f) and rear axle (δ_r), as well as the front axle and rear axle assembly velocities (v_f and v_r) constrain the model AGVs in their motion. Wang and Qi (2001) describe the velocity of AGVs in a planar frame of reference with Equation 5.6. The angle β is defined in Equation 5.7 and is depicted in Figure 5.4. It should be noted that the assumption is made that the v_f and v_r are equal, due to a single controller driving both motors.

$$v = \frac{v_f \cos\delta_f + l_r \cos\delta_r}{2\cos\beta} \tag{5.6}$$

$$\beta = \arctan \frac{l_f \tan\delta_r + l_r \tan\delta_f}{l_f + l_r} \tag{5.7}$$

5.3 A kinematic model for AGVs

In the previous sections the definition of a kinematic model and the mathematical expressions for it are discussed. By combining the orientation and motion, Wang and Qi (2001) created the kinematic model for AGVs depicted in Figure 5.4. This model is based on the well known bicycle model.

In the model, it is assumed that l_f, l_r, δ_f, δ_r, v_f and v_r are known. Wang and Qi (2001) state that with this information, the heading angle can be determined by using Equation 5.8. The decomposed velocities can then be calculated with Equations 5.9 and 5.10. Displacements can be obtained by integrating both Equation 5.9 and Equation 5.10. This however requires an initial position for each AGV (x_0, y_0).

$$\psi = \frac{v\cos\beta(\tan\delta_f + \tan\delta_r)}{l_f + l_r} \tag{5.8}$$

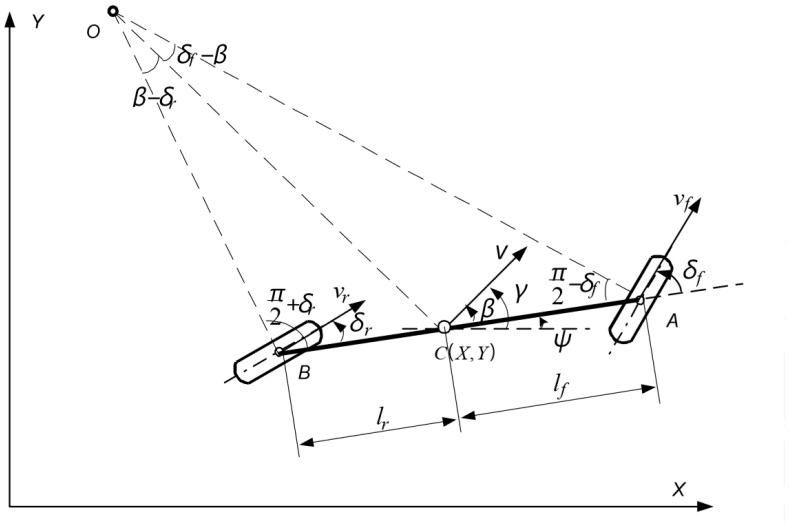


Figure 5.4: The kinematic model of a bicycle adopted for AGVs in reference to a fixed global coordinate system (Courtesy of Wang and Qi (2001)).

Table 5.1: A summary of all the used variables for the kinematic model and an explanation of their purpose.

Name	Symbol	Value (if applicable)	Unit
Vehicle velocity	v	-	$[m/s]$
Front axle velocity	v_f	-	$[m/s]$
Rear axle velocity	v_r	-	$[m/s]$
Vehicle orientation angle	ψ	-	$[rad]$
Velocity vector angle	β	-	$[rad]$
Front axle angle	δ_f	-	$[rad]$
Rear axle angle	δ_r	-	$[rad]$
Length front	l_f	$0,181$	$[m]$
Length rear	l_r	$0,121$	$[m]$

Table 5.2: A summary of the input change to pulsewidth.

Name	Previous variable	Symbol	Input range	Unit
Vehicle velocity	v	τ_v	900 - 2100	$[\mu s]$
Front axle angle	δ_f	τ_f	900 - 2100	$[\mu s]$
Rear axle angle	δ_r	τ_r	900 - 2100	$[\mu s]$

$$v_x = v\cos(\psi + \beta) \tag{5.9}$$

$$v_y = v\sin(\psi + \beta) \tag{5.10}$$

The kinematic model will be used vice versa mimicking the approach from Lee et al. (2010). With this, the relation between input and output given by the Arduino controller (Arduino) will be determined in chapter 6. Continuing from this point, the input of the kinematic model will be defined as the pulse width (τ) and is presented in Table 5.2. The position and orientation of the vehicle are recorded by the OptiTrack motion capture system and from this, the output values δ_f, δ_r and v can be determined. Combining these with the pulse width input should give the information that is necessary for calibration.

Chapter 6

Implementing the model

The workings of a kinematic model and application to AGVs are discussed in chapter 5. With this information, the kinematic model can be adapted for implementation in the AGV laboratory in order to determine the deviation between the servo input and the AGV's output. The system will be set up such that the following steps are performed, of which items 2 to 7 are automatic:

1. Servo inputs are manually set at a specific value (angle and velocity)
2. The model AGV describes an arc (preferably within the boundaries of the motion capture system)
3. The AGV's trajectory is registered with the motion capture system
4. Data frames from the motion capture system are sent to Matlab
5. Matlab registers the location and orientation of the vehicle and depicts these in a figure
6. After closing the figure, Matlab calculates the values for δ_f, δ_r and v based on the kinematic model mentioned in chapter 5
7. The values are printed on the screen
8. Curve fitting is performed to relate the input and output manually
9. The acquired curve is implemented in the Arduino code, completing the vehicle calibration

To achieve the proposed steps, Matlab code is written to perform these steps. Curve fitting the equation that circumscribes the relation between actuator input and output is best done manually, as this relation may differ per individual AGV. The curve fitting procedure and final code implementation (steps 8 and 9) in the controller will be further discussed in chapter 7.

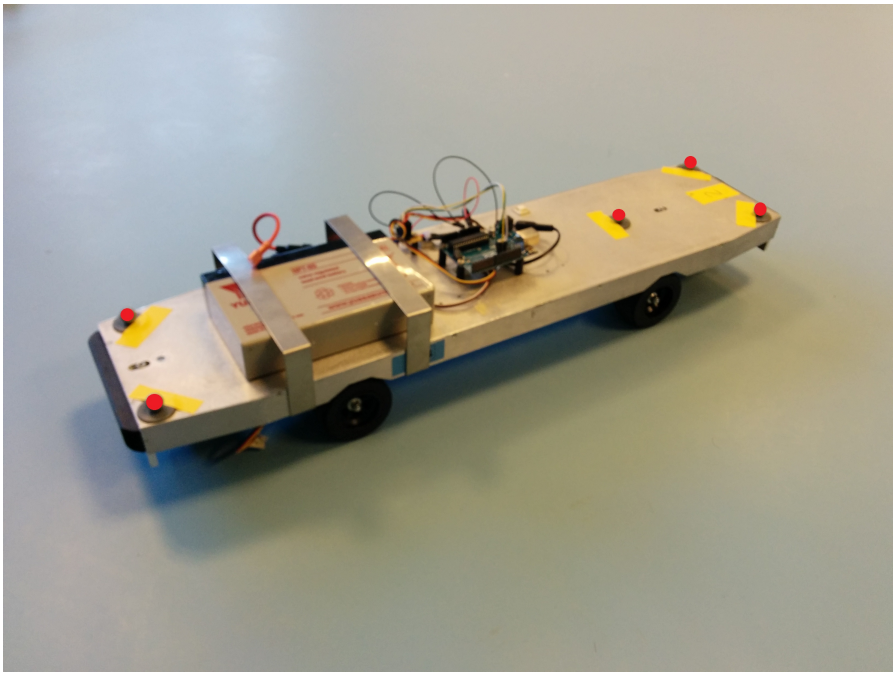


Figure 6.1: Model AGV with removed top part for better visibility. The markers are indicated with red dots.

This chapter is further built up as follows: the identification of AGVs by the motion capture system and available data is discussed in section 6.1. Implementing steps 2 to 7 is then performed in section 6.2. A short section is devoted to the implementation in the Arduino code in section 6.3. And to conclude this chapter, the program elaborated in section 6.2 is verified and validated in section 6.4.

6.1 AGV identification and data

The vehicle is identified by OptiTrack and MotiveTracker with reflecting markers. A total of five markers are placed on the model AGV. For better visibility of markers by the cameras, the top part of the AGV is removed as can be seen in Figure 6.1.

Motive:Tracker is able to distinguish different objects by measuring the distance between a specific number of markers. The minimum visible marker threshold can be set as desired but is automatically set at four in the case of five markers. This means that at least four markers have to be spaced at unique distances to track different objects. As long as the markers are not displaced, Motive:Tracker will always recognize the corresponding vehicle from the unique configuration of marker positions, even after the system is stopped or recalibrated.

Currently, only one AGV at each time is within the boundaries of the motion capture

system. It is therefore not a required to create multiple unique patterns of the IR reflecting markers. If this would be the case, the following should be noted:

- The precision of OptiTrack allows for a minimum marker spacing of 10 millimetres between individual marker locations
- Individual markers must be spaced at least 10 millimetres from each other to be recognized by the motion capture system
- The lengths between sets of two markers on an individual vehicle should be unique
- Markers on individual vehicles should not be placed symmetrically as OptiTrack will try to rotate the virtual rigid body around different axes to optimize the fit

For each individual AGV, Motive:Tracker registers seven variables. The first three variables represent the location data of the center of mass for the AGV's five markers. These x, y and z coordinates are measured in meters from the origin (dependent on the calibration). The x and z coordinates describe the ground plane, whereas the y coordinate circumscribes the vertical distance from this ground plane.

The remaining four variables, qx, qy, qz and qw are the quaternion of a vehicle. The founder of quaternions tried to find a unique three dimensional numbering system but failed to do so in three dimensions. In a four dimension numbering system however this could be achieved unambiguously (Baker, b; Rosenfeld, 1988). The four variables consist of one real (qw) and three imaginary dimensions (qx, qy and qz). The imaginary dimensions are all perpendicular and have a unit value of $\sqrt{-1}$. A more elaborate explanation on quaternions is given at Baker (b) and Rosenfeld (1988).

With several mathematical operations a quaternion can be converted into Euler angles (Baker, a). Euler angles represent three dimensional rotations with three numbers instead of four, this is however non-linear. The orientation of the model AGV is represented by its yaw, one of the three values associated with Euler angles. The conversion is further discussed in section 6.2.2.

The variables that were discussed can be broadcast from Motive:Tracker to Matlab on the same personal computer or to another computer over the network. NaturalPoint Inc has provided OptiTrack with an SDK that allows for this (NatNet). The SDK includes libraries that allow Matlab to retrieve the data and presents an example of how this is achieved. An instruction hereto is available in Appendix D.3.

6.2 Matlab program

The Matlab code example provided with the NatNet SDK, connects with Motive:Tracker and uses the data frames to depict the orientation of the subject. As there is little documentation available on the communication between Motive:Tracker and Matlab, the example was reverse-engineered for the implementation of the calibration procedure.

In subsection 6.2.1 is elaborated on the structure of the Matlab code. This is generated beforehand to reduce the risk of errors. The final programming code is presented in Appendix A. During the actual programming, several other issues were raised, which had to be processed. The mathematical solutions and foundation to these problems are discussed in subsection 6.2.2.

6.2.1 Conceptual model

Before the actual programming is performed, it is advisable to first determine the structure of the program in pseudo code, including statements and loops. This is done to reduce the risk of making errors in one's though process. The pseudo code for the two most important parts of the program, MainProgram and DataProcessing are elaborated in this section.

The **MainProgram** for calibration will perform steps 2 to 7 listed at the start of this chapter. It starts by closing all figures and clearing the workspace, after which it loads the parameters. Then, the rigid body data and frame rate are retrieved from OptiTrack. A figure and figure handle are created. The figure will be updated with the location of all rigid bodies by the function **DataProcessing**. This function is called by the data listener every single time a frame of data is ready in OptiTrack.

After closing the figure, the data from the figure is processed and a circle fit is applied to the location data. This yields a center point and a radius. With the center point and the orientation of an AGV, the angles of the front and rear axles can be calculated. Hereafter, the location data is processed into velocity, of which the median is calculated and all information is written to a file. The steps are listed hereunder.

MainProgram

1. Close figures and clear workspace
2. Load parameters
3. Retrieve rigid body data from OptiTrack
4. Retrieve frame rate from OptiTrack
5. Create figure of rigid body location
6. Set up data listener
7. Wait for figure to be closed
8. Delete data listener
9. Fit circle on location data
10. Calculate axle angle
11. Process location data into velocity data
12. Calculate median velocity
13. Write data to file

During the data listener callback, the function **DataProcessing** is activated every time Matlab receives a data frame. The function processes the data from the data frame and

stores it in variables. The data is then used to update the location figure after which the function is closed, to be recalled again by a new data frame. This will be performed in the following steps:

DataProcessing

1. if (no data in stored variable)
   (a) create initial data
   (b) create location figure information
2. end
3. if (frame is a duplicate)
   (a) end function
4. end
5. if (frames have been missed)
   (a) read rigid body data from data frame
   (b) process data frame information
   (c) store data in variable
   (d) fill missing information
6. else
   (a) read rigid body data from data frame
   (b) process data frame information
   (c) store data in variable
7. end
8. process stored data in figure

The entire program is terminated when the user closes the location data figure. An example of this location figure with added data is depicted in Figure 6.2. Hereafter the main code processes some additional data and stores this before terminating completely. The entire code is presented in Appendix A.

6.2.2 Additional calculations

As was mentioned, during programming several additional issues occurred which are not described in chapter 5. The first issue is dealing with quaternions instead of angles. The second issue that will be discussed in this section elaborates on the operations that are required to calculate the center of rotation for an AGV. With this center of rotation, it can be determined which case is applicable to an AGV during calibration. Finally, the velocity is calculated with the information from OptiTrack.

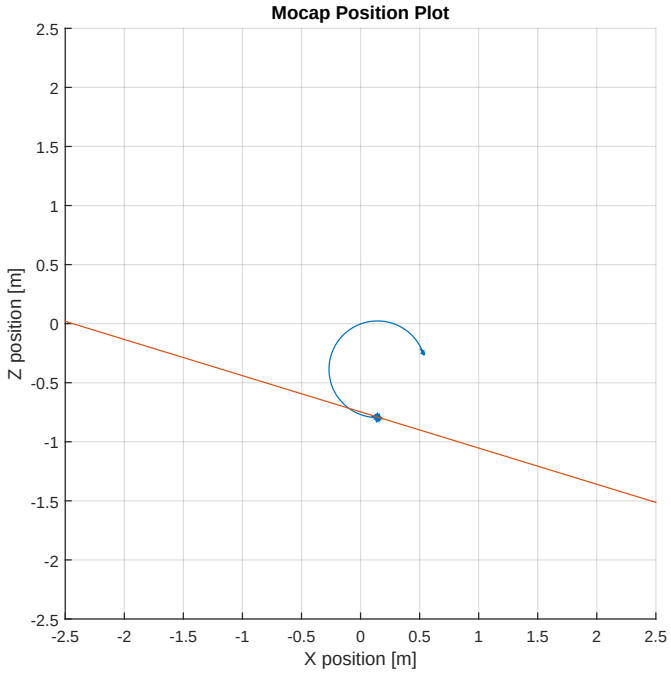


Figure 6.2: Location data plotted in a Matlab figure.

Transforming quaternions to euler angles

For the calibration of AGVs, it was important to retrieve the yaw of a vehicle. OptiTrack however outputs the orientation data in quaternions q_x, q_y, q_z and q_w to give a uniquely defined orientation. Euler angles however do not unambiguously define the orientation of an object and transformation typically limits the yaw of an object to 90 degrees. The limitation can be circumvented by using Equation 6.1 and allows for a full 360 degrees rotation.

$$\beta = \arctan \frac{2q_y q_w - 2q_x q_z}{1 - 2q_y^2 - 2q_z^2} \tag{6.1}$$

There are two exceptions to Equation 6.1, which can be found with Equation 6.2. When the value *pole* is 0.5, the yaw is directed to the north pole. However when *pole* is -0.5, the yaw is directed to the south pole. To prevent the Matlab from giving an error, an if statement is used to execute Equation 6.3 in the case of a north pole and Equation 6.4 in the case of a south pole.

$$pole = q_x q_y + q_z q_w \tag{6.2}$$

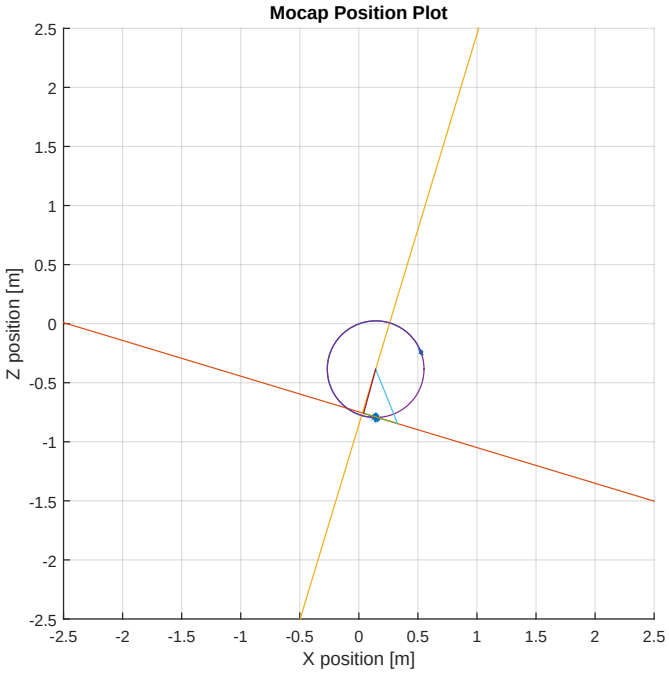


Figure 6.3: Matlab figure after performing all steps for determining the steering angles of the front and rear axles.

$$\beta = 2 \arctan \frac{q_x}{q_w} \tag{6.3}$$

$$\beta = -2 \arctan \frac{q_x}{q_w} \tag{6.4}$$

Circle fitting to data

In order to find the center of rotation and radius of a trajectory, a circle is fitted on the stored location data of an AGV. On the Matlab website (Mathworks), several circle fitting algorithms exist. An updated version of a fast approach for geometric circle fitting is created by Sumith, which follows the Landau-Smith method presented in Tomas and Chan (1989).

A circular fit following this method is depicted in Figure 6.3. Herein, the trajectory depicted in Figure 6.2 is fitted. As the data shows, fitting with the Landau-Smith method allows for data fitting of partial circles, which is desired as a trajectory will not always circumscribe a full circle.

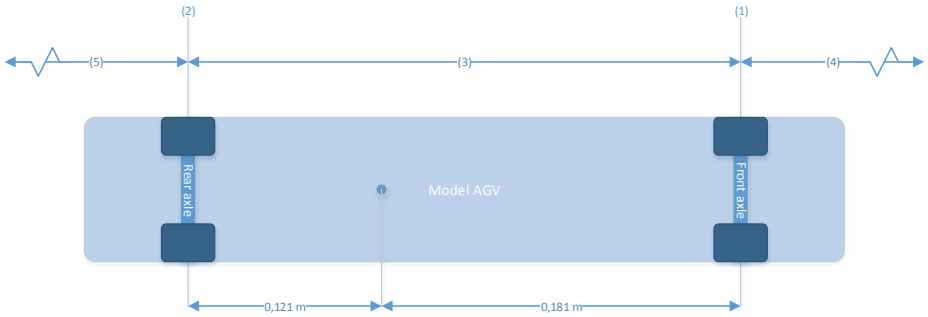


Figure 6.4: [Determining which scenario is the case for the center point of the circle fitted to the location data.

Determining which case

In chapter 5 is explained that three cases exist for model AGVs to circumscribe a circle. For calculation purposes, it is necessary to find which of this cases holds for a performed trajectory. This can be achieved by finding the projection of the circle center on the AGV's center line. The following positions of the circle center projection are possible and are depicted in Figure 6.4:

1. On the front axle
2. On the rear axle
3. Between the front and rear axle
4. In front of the front axle
5. Behind the rear axle

The AGV's center line can be calculated with Equation 6.5. This requires the yaw of the vehicle (β). In the equation $m = \tan\beta$ and the n value can be determined with the location data of the AGV's center.

$$y_1 = mx_1 + n \tag{6.5}$$

The shortest distance between line y_1 and the circle center is determined by the intersection of lines y_1 and y_2 (see Equation 6.6). For line y_2, $p = -1/m$ and q can be determined by substituting the x and y values for the circle center.

$$y_2 = px_2 + q \tag{6.6}$$

The calculated locations can then be used to determine the distances required in Equations 5.1, 5.2, 5.3, 5.4 and 5.5 according to the five cases listed above. After performing all these steps, Matlab plots the lines and distances in a figure, as is depicted in Figure 6.3. The angle of the axles are then presented in degrees.

Determining the AGV's velocity

For the comparison of the AGV's velocity input and output, the AGV is driven at a constant velocity. By taking the median of all the velocity data, a good estimation can be obtained. OptiTrack however does not register the velocity of objects and it is therefore required to calculate the velocity of an object based on two locations and the passed time. This is performed with Equation 6.7.

$$v = \sqrt{(x_{i+1} - x_i)^2 + (z_{i+1} - z_i)^2} \tag{6.7}$$

The velocity is only calculated after finishing the program, as it is not necessary to calculate it during each loop. This saves processing power for more important tasks.

6.3 Arduino program

A generic program for AGV control is available in the laboratory. The workings of the program are described in van Blijswijk (2015). After calibration however, the program needs to be adapted to relate the calibrated input to the output. During tests it was also noticed that the method developed in van Blijswijk (2015) had an accuracy of 1 degree. To circumvent this problem, an extra library is included in the Arduino program that allows for servo control with the pulse width of the signal (τ_v, τ_f and τ_r).

In Appendix C, the data is given for the servos used in the model AGVs. The data is retrieved from Database and shows that the used Hitec HS-700BB has a pulse width of $900 - 1200\mu s$. Comparing this with the range of 45 to 135 degrees described in van Blijswijk (2015), gives a $13,3$ times higher resolution when using pulse width. Additionally, with the old commands, it was not possible to bring the model AGV to a complete halt.

If these commands would have been used during a simulation, it would have resulted in numerous errors, with vehicles randomly running their course. Reluctantly, controlling the servo with its pulse width results in a better control of the AGV. With a pulse width set to 1500, a servo completely stops operating. An extra effect of this is that the AGVs do not produce sound when they halt. The Arduino program from van Blijswijk (2015) is thus adapted to incorporate these additional features.

Table 6.1: Angle of the front and rear axle when the Arduino is given a certain command for the axle steering angle. The presented data is an average of three measurements.

Vehicle command [μs]	900	1100	1300	1500	1700	1900	2100
Vehicle output front [degrees]	39,8	27,2	14,1	1,87	-11,0	-24,7	-38,4
Vehicle output rear [degrees]	34,4	24,0	11,8	-0,10	-13,0	-25,6	-36,1

Table 6.2: Velocity of an AGV at two different inputs. The presemted data is an average of three measurements.

Vehicle command	900	1500	2100
Vehicle output [m/s]	0,32	0,00	-0,32

6.4 Verification and validation

An important part of assessing the usefulness of models and computer programs is the verification and validation. In chapter 4, among others, the definition of the terms verification and validation are discussed. The focus in this chapter was on models, but can also be applied to software. The verification and validation of the kinematic model will be skipped, as this is already verified and validated in Wang and Qi (2001). Additonally Gerritse (2014) has used this kinematic model for the odometry of AGVs.

6.4.1 Verification

The definition of verification states that it can be associated with asking "Is the model right?". A typical method to answer this question is to provide the system with simple inputs and verify that the outputs correspond to one's expectations. As the verification is purely intended for testing the model, the vehicle has not yet been calibrated. It can thus be expected that the desired angle and actual angle of the axles are not similar. This is the case in Table 6.1.

It can however be verified that the model is working correctly from this table. When setting the steering angle at -30 degrees instead of -45 degrees, one expects the output also to be higher. This is the case for all the tested angles in Table 6.1.

For measuring the velocity of the AGVs, there is little comparison data. In de Groot (2015) is measured that the maximum velocity of an AGV is $0,32m/s$. In the results depicted in Table 6.2 is found that the maximum velocity matches the velocity found in de Groot (2015). From the tests can be assumed that both the rotational as well as the translational properties of AGV are being measured correctly by the implemented program that is described in this chapter.

6.4.2 Validation

For the validation of models and software, one can ask a similar question as is the case with verification: "Is it the right model?". The intention of this question is to determine whether the calibration program complies with the set demands. The demand for the calibration process was that it should be automated and applicable to the AGVs in the laboratory.

The program is applicable to the laboratory AGVs and for the largest part automated. The data for each calibration however must be interpreted manually, to assess the best fitted curve. With the exception of data interpretation, the calibration method is automated. Although this is the case, the program is still considered to be compliant to the demands made and therefore validated.

Chapter 7

Performing the calibration

Now that the program code has been verified and validated, the calibration of a model AGV can be performed. The calibration will be performed for the steering servos and the drive servos. How the tests are performed is described in section 7.1 with two schematic depictions of the method. Hereafter, the results to the calibration of the steering and drive servos are mentioned in section 7.2.

These results are then interpreted in section 7.3 with the use of curve fitting to establish the relation between servo input and output. This relation can then be used in the Arduino code to minimize the deviation between the desired output of the servo and the actual output of the servo.

7.1 Calibration procedure

To ensure that every single AGV is calibrated appropriately, it is useful to have a standard calibration procedure. For this report, the calibration procedure is more extensive, as the relation between input and output is still unclear. The procedure will therefore contain more data points and a higher number of repetitions than the calibration procedure that will be used in practise.

In this research, the front axle, rear axle and drive servos will be calibrated. This results in two individual procedures, one for the steering servos and one for the drive servos. The first procedure is depicted schematically in Figure 7.1 and determines the relation between the pulse width for the steering angle (τ_f and τ_r) as input and actual steering angle as output (δ_f and δ_r).

The procedure starts with setting the front axle as active subject to be varied. It sets the pulse width of all three servos to $1500\mu s$, which is the center for both axles and

halt for the drive servo. The active subject (in this case the front axle) is then varied between $900\mu s$ and $2100\mu s$ with steps of $100\mu s$ and each step being measured 10 times. After completion of all steps, the rear axle is set as active and all servos are set to their mid values. Again the pulse width will be varied between $900\mu s$ and $2100\mu s$, every step being $100\mu s$. The measurements are stored whereafter the last part of the procedure is performed, opposite steering axles. Originally it was planned to measure these with steps of $100\mu s$, but due to time constraints the measurement was performed with $300\mu s$ steps. This concludes the steering angle calibration.

In the second procedure, the relation between the pulse width for the driving servo (τ_v) and the vehicle velocity (v) for different steering angles and configurations is determined. A flowchart schematic is depicted in Figure 7.2. For this procedure, the test does not have to be repeated multiple times, as the velocity is calculated between data points in a vector. The presented velocity then is the median of the velocity vector.

The procedure starts with setting the pulse width of the drive servo to $900\mu s$. A loop is started which sets the AGV to front axle single steer and varies the steering servo input from $900\mu s$ to $2100\mu s$ with steps of $200\mu s$. This is then repeated for both opposite steer and crab steer. The loop is repeated for a drive servo pulse width of $900\mu s$ to $2100\mu s$ in steps of $50\mu s$. Reaching $2100\mu s$ concludes the measurements.

7.2 Calibration results

From the tests described in section 7.1, a large amount of data was retrieved. The data in Table 7.1 depicts the statistics of the results for front wheel steering with most importantly the median, minimum and maximum. The maximum spread between median and absolute extreme value of the front axle is observed to be $0,46$ degrees. The same test performed on the rear axle shows the absolute spread for the rear axle of the tested AGV to be $0,94$ degrees, seen in Table 7.2.

For Tables 7.1 and 7.2, only one axle is rotated whilst the other is kept approximately straight. Rotating both axles in opposite direction yields the results from Table 7.3. Comparing the results with Tables 7.1 and 7.2, shows that a similar input for both single and opposite steering results in a similar steering angle observed by the calibration procedure, thus further assuring the correctness of the method.

After performing the measurements for steering angles, the velocity of the AGV is measured. The median of the velocity vector is presented in Table 7.4. This table gives a clear overview.. It was however noticed that the AGVs tend to drive slower when taking corners. This information is presented in Appendix B. The velocities are measured for 16 different speed inputs, with 7 different angles in 3 configurations, resulting in 336 median velocities.

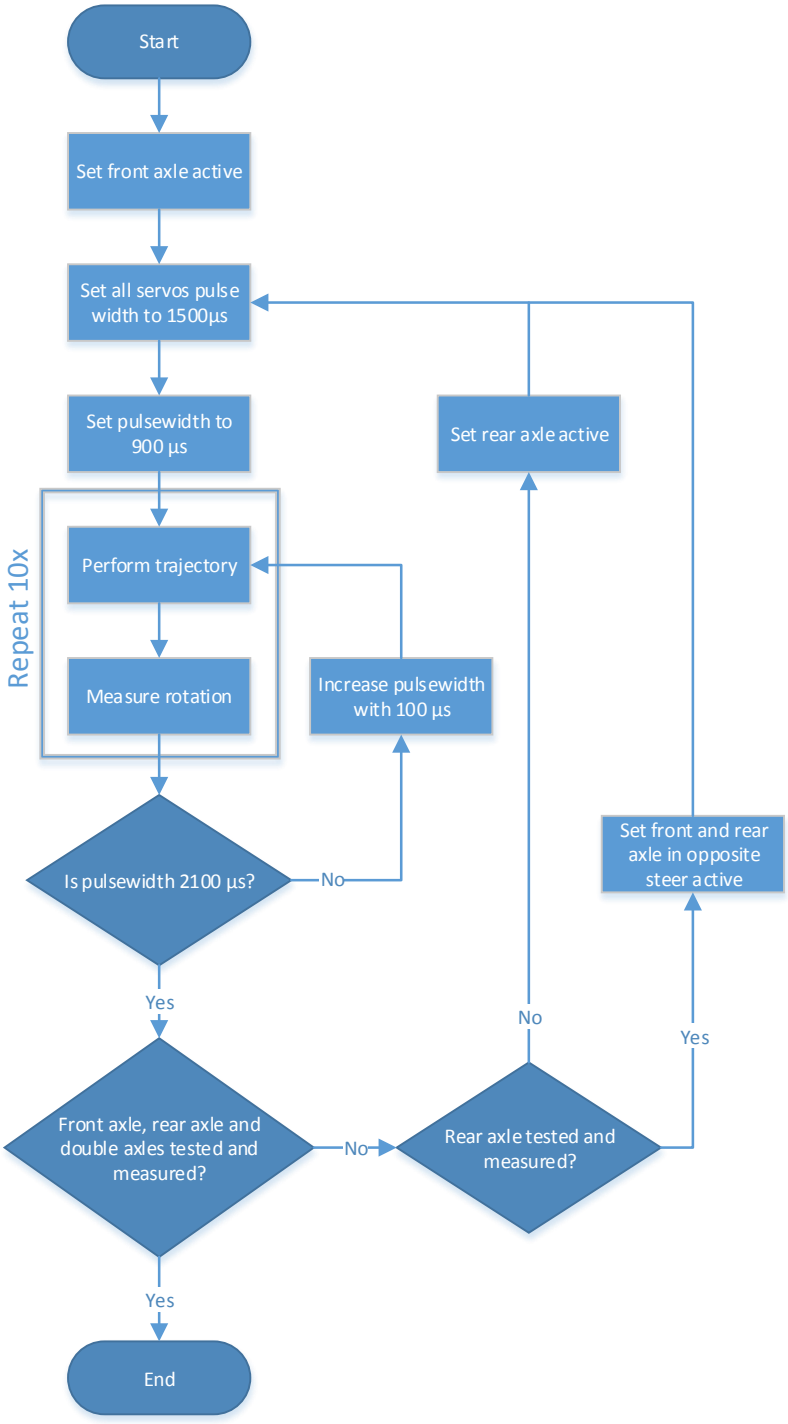


Figure 7.1: A flowchart depicting the procedure that will be used to calibrate the rotation of model AGVs.

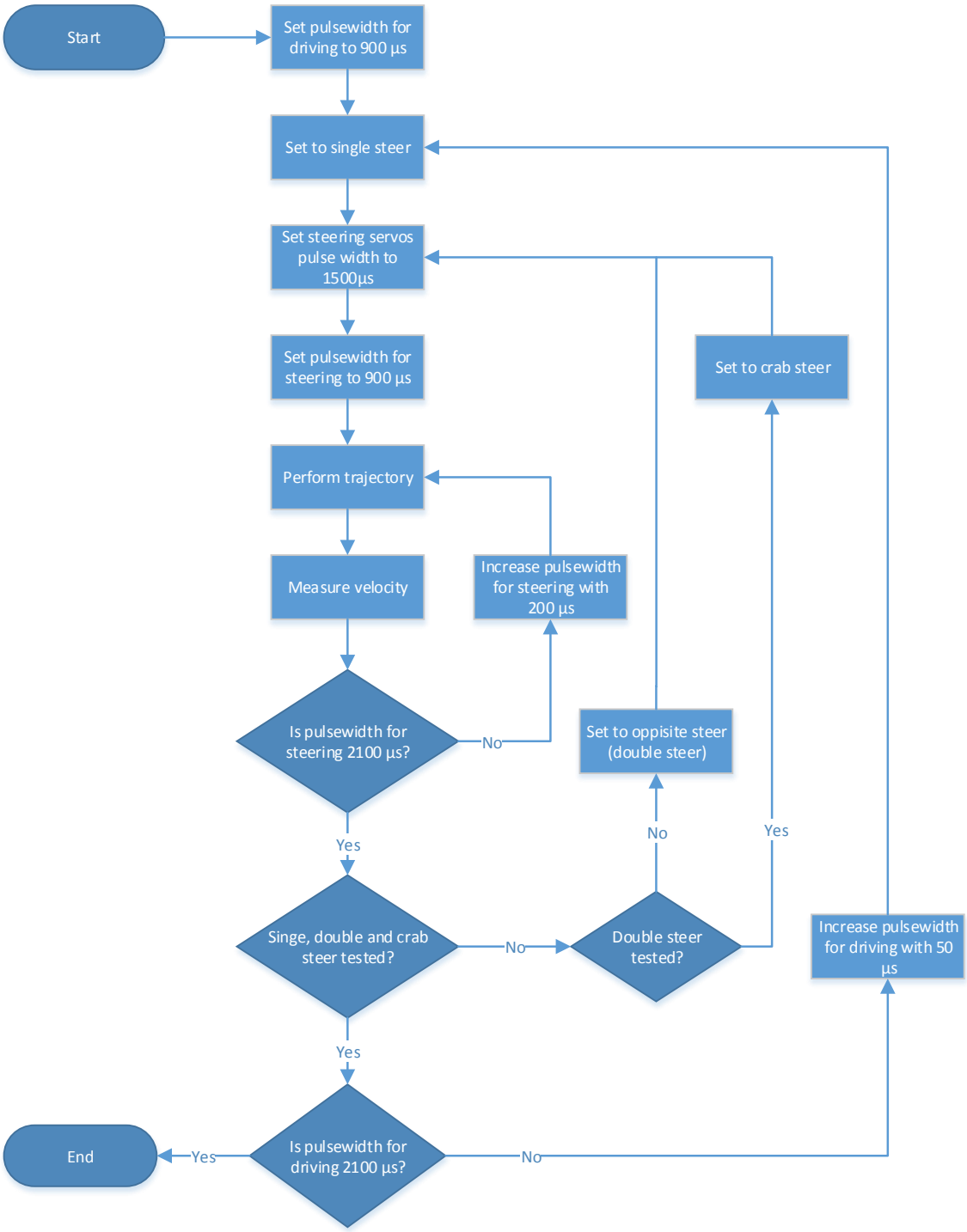


Figure 7.2: A flowchart depicting the procedure thath will be used to calibrate the velocity of model AGVs.

Table 7.1: Minimum, maximum, median, 1st quartile and 3rd quartile of the tests with changing front servo in degrees.

Input [μs]	Front axle [degrees]					Rear axle [degrees]				
	MIN	Q1	MED	Q3	MAX	MIN	Q1	MED	Q3	MAX
900	40,08	40,14	40,21	40,23	40,26	-3,12	-3,07	-2,96	-2,38	-2,23
1000	40,06	40,07	40,17	40,22	40,30	-2,89	-2,79	-2,69	-2,64	-2,55
1100	33,40	33,51	33,65	33,65	33,87	-2,48	-2,41	-2,27	-2,24	-2,12
1200	24,69	24,70	24,75	24,78	24,89	-2,63	-2,58	-2,54	-2,52	-2,41
1300	16,11	16,12	16,17	16,30	16,30	-2,19	-2,15	-2,13	-1,97	-1,94
1400	7,42	7,43	7,57	7,61	7,87	-2,49	-2,47	-2,35	-2,33	-2,32
1500	-1,71	-1,63	-1,59	-1,54	-1,19	-2,29	-2,09	-2,04	-1,96	-1,82
1600	-10,18	-9,80	-9,72	-9,72	-9,60	-1,75	-1,70	-1,69	-1,63	-1,58
1700	-18,45	-18,35	-18,28	-18,23	-18,19	-1,63	-1,49	-1,47	-1,38	-1,35
1800	-26,68	-26,64	-26,61	-26,50	-26,42	-1,49	-1,41	-1,40	-1,21	-1,18
1900	-35,55	-35,52	-35,47	-35,47	-35,45	-1,25	-1,19	-1,15	-1,11	-1,10
2000	-41,30	-41,24	-41,22	-41,14	-41,14	-1,30	-1,22	-1,13	-1,02	-1,00
2100	-42,13	-42,10	-42,06	-42,01	-41,93	-1,67	-1,60	-1,50	-1,43	-1,34

Table 7.2: Minimum, maximum, median, 1st quartile and 3rd quartile of the tests with changing rear servo in degrees.

Input [μs]	Front axle [degrees]					Rear axle [degrees]				
	MIN	Q1	MED	Q3	MAX	MIN	Q1	MED	Q3	MAX
900	-0,43	-0,16	-0,10	0,05	0,08	40,13	40,38	40,39	40,42	40,42
1000	-0,64	-0,62	-0,54	-0,45	-0,41	39,13	39,22	39,23	39,25	39,40
1100	-1,05	-0,70	-0,57	-0,47	-0,46	32,14	32,45	32,50	32,51	32,68
1200	-0,80	-0,67	-0,67	-0,64	-0,61	23,90	23,98	24,01	24,05	24,08
1300	-1,18	-1,11	-1,05	-0,94	-0,84	15,14	15,22	15,26	15,36	15,48
1400	-1,71	-1,22	-1,20	-1,11	-1,11	6,20	6,49	7,14	7,15	7,17
1500	-1,51	-1,47	-1,44	-1,36	-1,18	-2,45	-2,31	-2,27	-2,27	-2,16
1600	-1,34	-1,26	-1,08	-1,01	-0,97	-11,34	-11,31	-11,30	-11,25	-10,60
1700	-1,53	-1,49	-1,35	-1,24	-1,22	-20,26	-20,25	-20,13	-20,00	-19,97
1800	-1,68	-1,55	-1,29	-1,17	-1,16	-29,33	-29,16	-28,99	-28,84	-28,83
1900	-1,73	-1,59	-1,46	-1,43	-1,41	-37,58	-37,48	-37,48	-37,37	-37,32
2000	-1,75	-1,36	-1,31	-1,28	-1,25	-39,54	-39,22	-39,18	-39,17	-39,10
2100	-1,52	-1,49	-1,25	-1,21	-1,05	-39,97	-39,95	-39,76	-39,74	-39,69

Table 7.3: Minimum, maximum, median, 1st quartile and 3rd quartile of the tests with changing front and rear servo in opposite steering mode in degrees.

Input [μs]	Front axle [degrees]					Rear axle [degrees]				
	MIN	Q1	MED	Q3	MAX	MIN	Q1	MED	Q3	MAX
900	40,01	40,09	40,16	40,34	40,46	40,85	41,13	41,18	41,19	41,20
1200	24,64	24,80	24,84	24,97	25,00	24,33	24,42	24,62	24,63	24,88
1500	-1,51	-1,47	-1,44	-1,36	-1,18	-2,45	-2,31	-2,27	-2,27	-2,16
1800	-27,33	-27,21	-27,19	-27,18	-27,09	-29,11	-29,09	-28,99	-28,98	-28,94
2100	-42,29	-42,12	-42,10	-41,96	-41,94	-40,64	-40,63	-40,42	-40,26	-40,23

Table 7.4: Median of the velocity vector for an AGV driving with both steering servos set at $1500\mu s$, driving straight forward.

Drive servo input [μs]	Velocity [m/s]
900	0,322
950	0,323
1000	0,323
1050	0,322
1100	0,324
1150	0,322
1200	0,321
1250	0,291
1300	0,260
1350	0,221
1400	0,160
1450	0,060
1500	0,000
1550	-0,006
1600	-0,116
1650	-0,192
1700	-0,243
1750	-0,275
1800	-0,299
1850	-0,325
1900	-0,324
1950	-0,323
2000	-0,323
2050	-0,322
2100	-0,323

Table 7.5: The R^2 of the three servos for two different curve fit candidates. For the drive servo, both the full data set and a data set with partially neglected points are used.

Curve fit	Front axle steering servo	Rear axle steering servo	Drive servo (full)	Drive servo (partial)
1^{st}-order polynomial	0,9991	0,9954	0,9234	0,9837
3^{rd}-order polynomial	-	-	0,9917	0,9949

7.3 Curve fitting the data

To determine the relation between the input and output of a servo, curve fitting is used. The curve can vary from n^{th}-order polynomial to exponential function or anything fitting. This mathematical relation will then be used to send the right commands to the servos. It should be noted that the relation only has to be valid for the input range from -40 to 40 degrees, as this is the limit of the servos.

For the steering servo data obtained in Table 7.1 and Table 7.2, as well as the drive servo data from Table 7.4, there are two possible candidates for the curve fit. Other possibilities are not considered since the data does not indicate the existence of such curves. The two possible candidates are:

- A 1^{st}-order polynomial (simple line)
- A 3^{rd}-order polynomial (higher order function)

The R^2 of all data curve fits is presented in Table 7.5. It appeared that during for the 3^{rd}-order polynomial curve fit of the steering servo data, insufficient data point were available to reliably calculate the curve. Since the linear fit already gives an accuracy of at least $99,5\%$ on the steering servos, the 1^{st}-order polynomial is used for the calibration thereof. An illustration of the front axle and rear axle servo curve fits is depicted in Figure 7.3 and Figure 7.4 respectively. It should be noted that since for both servos, the two most outer data points are omitted. Equation 7.1 and Equation 7.2 are therefore only applicable in the range of -40 to 40 degrees.

$$\delta_f = -0,0840 pw_B + 125,00 \tag{7.1}$$

$$\delta_r = -0,0837 pw_C + 123,55 \tag{7.2}$$

For the drive servo, Table 7.5 shows that a better accuracy can be gained by using the 3^{rd}-order polynomial curve fit on the partial data set from section 7.2. A depiction of this is given in Figure 7.5. Here it also should be noted that since several data points are neglected, the equation for the curve, Equation 7.3, is only applicable for the range from $-0,32$ to $0,32$ m/s.

$$v = 3,74x10^{-9} pw_A^3 - 1,70x10^{-5} pw_A^2 + 0,024 pw - 10,86 \tag{7.3}$$

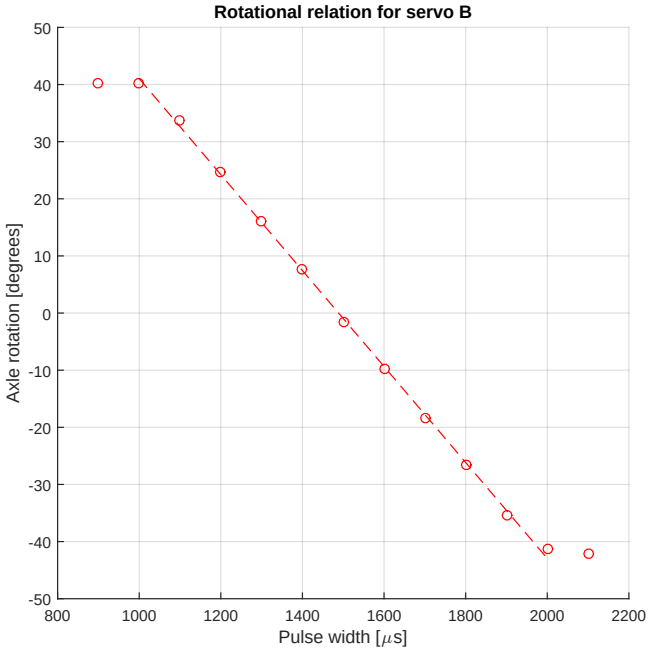


Figure 7.3: Linear fit of the steering angle data for the front axle.

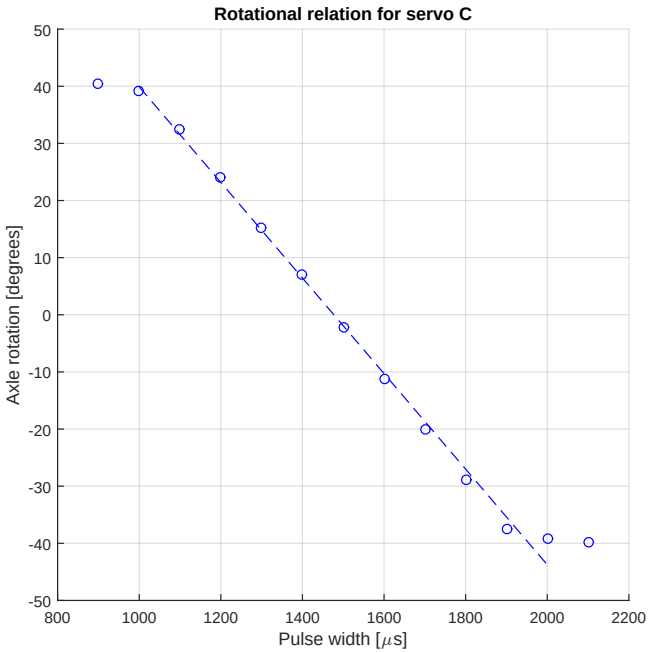


Figure 7.4: Linear fit of the steering angle data for the rear axle.

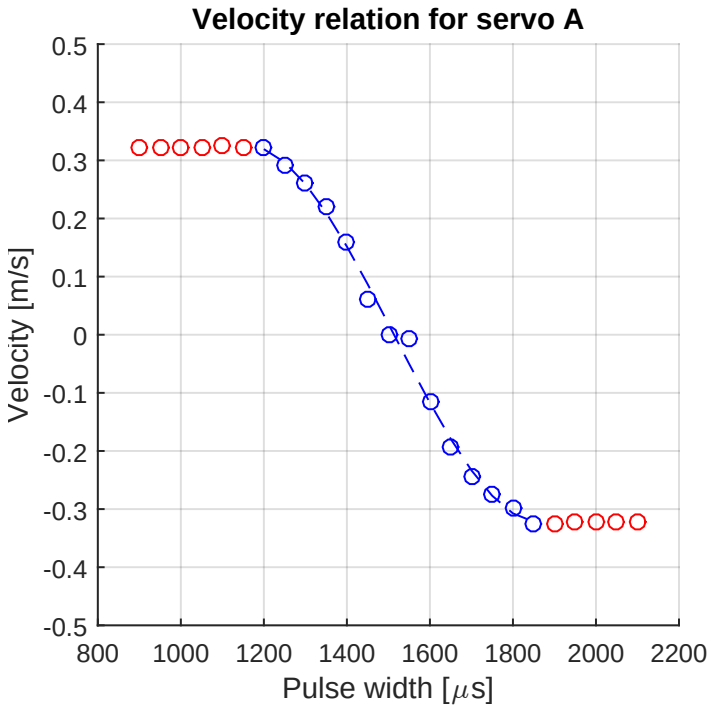


Figure 7.5: A 3rd-order polynomial fit to the velocity data. The red data points are neglected in the fit to improve the overall fit for the specific range of $-0,32$ to $0,32$ m/s.

The equations for the calibration of the steering servo are now implemented in the Arduino program code. The drive servo curve fit is then used to calculate the time interval between starting and stopping the vehicle.

As an extra note, in section 7.2 is mentioned that the velocity also is tested for different steering angles and types of steering. The data can be found in Appendix B. In order to give an impression, a plot and curve fit is generated for one of the inputs, namely $1200\mu s$ on the drive servo, with differing steering angles ranging from $900\mu s$ to $2100\mu s$. This plot is depicted in Figure 7.6. Future research may focus on finding an equation which predicts the velocity based on the servo inputs for the drive and steering servos.

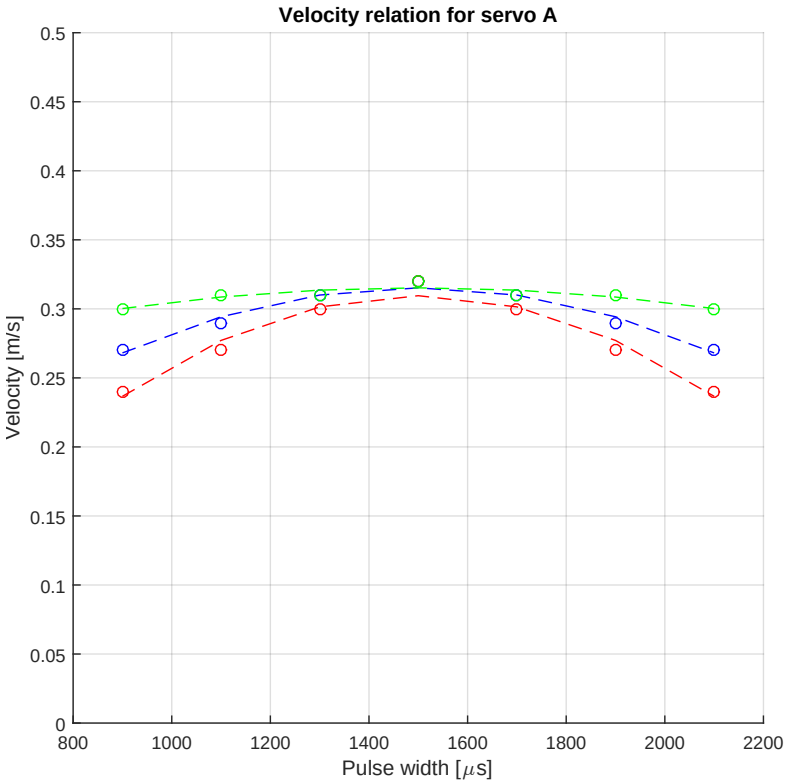


Figure 7.6: A plot and simple curve fit to give an impression of the relation between velocity and steering angle for different types of steering. The drive servo is set at 1200μs and the steering angle input ranges from 900μs to 2100μs with steps of 200μs. The green curve, blue curve and red curve represent the crab steer, single steer and double steer respectively.

Chapter 8

Results

This chapter presents the results of the performed tests. There are two main parts of this chapter and the report that require testing. In the first part, the assumption is made that the battery drainage can be neglected when regarding the performance. This is substantiated with evidence presented in section 8.1. The section also discusses the effect of acceleration on the precision of the results.

The second part of this chapter depicts the data that will be used to answer the main research question in chapter 9. The data is obtained by performing the approach described in chapter 2. Herein, a calibrated and an uncalibrated AGV are compared for start and end position.

8.1 Performance of a model AGV

A Matlab script was created to determine the vehicles velocity and log this in a file. In Figure 8.1 the velocity is constructed for a period of 14717 seconds. Several peaks occur, which are caused by the motion capture system losing track of the subject for several frames. Between approximately 10000 and 14717 seconds, these peaks occur regularly due to the model AGV leaving the captured workspace and re-entering it. These data points will be neglected in the results.

The total endurance test lasted for 29877 seconds, about 8 hours and 18 minutes. The position is measured 10 times each second, giving 298770 data points. Table 8.1 shows the percentiles for the velocity of all data points. This shows that the median velocity is 0,305 m/s. Table 8.2 shows the velocity of the model AGV for several intervals. Each velocity consists of an average of 200 data points, with each interval spaced 3000 seconds apart. This provides a rough overview of the dependency on battery power by the forward actuating motor.

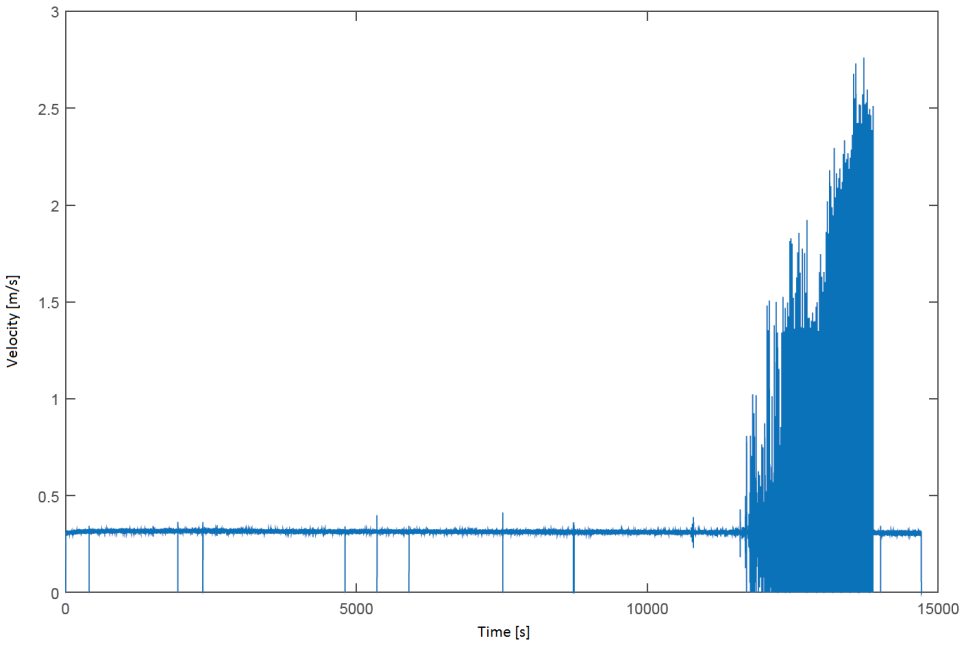


Figure 8.1: Four hour part of the model AGV endurance test.

Table 8.1: Percentiles for all endurance test data points.

Percentile	5	25	50	75	95
Velocity [m/s]	0.287	0.297	0.305	0.313	0.319

Table 8.2: Velocities in [m/s] for several intervals in the endurance test.

Interval	1	2	3	4	5	6	7	8	9	10	11
Velocity [m/s]	0.311	0.315	0.314	0.311	0.310	0.303	0.303	0.300	0.296	0.292	0.291

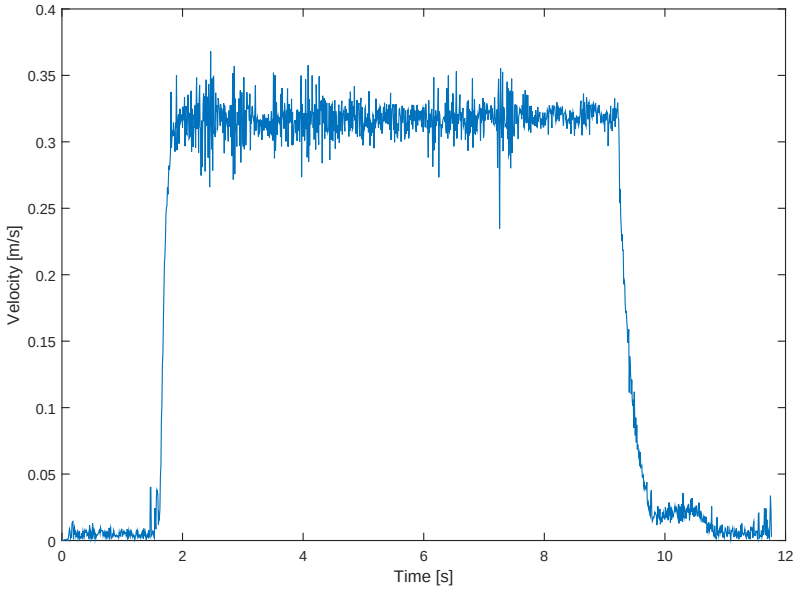


Figure 8.2: The velocity-time diagram of a model AGV derived from the location data.

The table clearly shows that after 8 hours and 18 minutes of operation, the final velocity is 7.6% lower than the maximum achieved velocity. It should be noted that the peaks from Figure 8.1 are disregarded in these results.

To assess the effect of acceleration and deceleration, a short test run is performed. The velocity-time graph is depicted in Figure 8.2. The graph shows a near vertical line where acceleration and deceleration occurs. After the deceleration however, a small increase in velocity is noticed. The hypothesis is that this small increase is present due to the AGV driving backward as a result of heavy braking and the fact that the calculated velocity is always positive.

From the data of the graph can be retrieved that the acceleration from 0,0 m/s to 0,3 m/s occurs in 0,294 sec. The deceleration is from 0,32 m/s to 0,02 m/s in 0,588 sec. This means that during acceleration a total of 0,043 m will have passed and during deceleration a total of 0,099 m will have passed. On a 10 second drive, as is seen in Figure 8.2, this would be a small deviation of about $4,7\%$. The percentage will decrease as the length of the trajectory increases. It is therefore assumed that acceleration and deceleration effects can be neglected for calibration.

8.2 Comparison of a calibrated and an uncalibrated AGV

In this section is investigated whether calibration has a positive effect on the final position and rotation of AGVs. The approach for performing these tests is explained in

chapter 2. For the tests, several trajectories are proposed. These trajectories are however limited by the presence of only one uncalibrated velocity, which is the maximum velocity. The uncalibrated AGV is thus only tested with maximum velocities. When taking this into account, the following basic trajectories are proposed for testing and depicted in Figure 8.3:

1. A straight trajectory of 2 meters in forward direction at maximum velocity
2. A straight trajectory of 2 meters in backward direction at maximum velocity
3. A straight trajectory of 2 meters in forward direction at half the maximum velocity
4. A straight trajectory of 2 meters in backward direction at half the maximum velocity
5. A turn of 90 degrees with the end location 1 meter to the left and 1 meter in front of the start at maximum velocity.
6. A turn of 90 degrees with the end location 1 meter to the right and 1 meter in front of the start at maximum velocity.
7. A turn of 90 degrees with the end location 1 meter to the left and 1 meter behind the start at maximum velocity.
8. A turn of 90 degrees with the end location 1 meter to the right and 1 meter behind the start at maximum velocity.

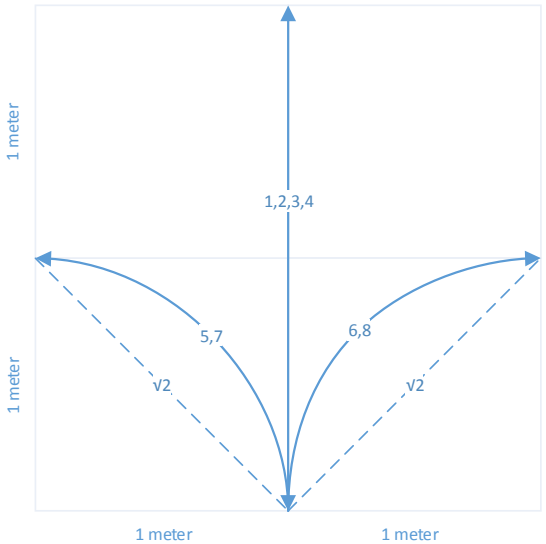


Figure 8.3: The test trajectories for the comparison of calibrated and uncalibrated AGVs. The distances are denoted, as well as the numbers of each trajectory.

Table 8.3: Test results for trajectory case 1, driving forward in a straight line at maximum velocity. In this table, the values for Δs are denoted in millimetres [mm] and the values for $\Delta\delta$ in [degrees].

	MIN	Q1	MED	Q3	MAX
$\Delta s_{uncalibrated}$	72,46	77,06	83,03	87,46	119,80
$\Delta s_{calibrated}$	1,20	3,04	5,90	13,03	16,30
$\Delta\delta_{uncalibrated}$	0,60	2,85	5,10	5,875	8,70
$\Delta\delta_{calibrated}$	0,20	1,33	1,90	2,30	2,30

Table 8.4: Test results for trajectory case 2, driving backward in a straight line at maximum velocity. In this table, the values for Δs are denoted in millimetres [mm] and the values for $\Delta\delta$ in [degrees].

	MIN	Q1	MED	Q3	MAX
$\Delta s_{uncalibrated}$	7,40	10,73	18,34	22,20	56,44
$\Delta s_{calibrated}$	3,50	12,25	14,60	18,72	70,10
$\Delta\delta_{uncalibrated}$	1,30	2,33	3,4	3,95	5,70
$\Delta\delta_{calibrated}$	0,10	1,93	2,3	3,35	3,60

For the proposed trajectories it, the starting position and rotation is measured as well as the end position and location. To simplify the results, taking into account the possibility of cornering by the vehicle, the position data is transformed into a distance between start and end position. For the straight trajectories, these distance needs to be 2 meters, whereas the turning trajectories require a distance of 1,41 meters. The observed rotation of the vehicle must be 0 degrees in the case of a straight trajectory and 90 degrees for a turn.

Table 8.3 and Table 8.4 show the results for cases 1 and 2. It is observed that for both the forward and the backward case, the median of the distance between start and end is improved. However, the forward case shows more improvement than the backward case. A similar result is observed for the rotation of the vehicle in both cases.

Tables 8.5 and 8.6 show the results of a calibrated AGV performing a straight 2 meter trajectory at half the maximum speed. No uncalibrated data is available since the only known velocity of an uncalibrated AGV is its maximum velocity. The results are consistent with cases 1 and 2, confirming the potential of calibration on model AGVs.

Table 8.7 through Table 8.10 depict the results for cases 5 through 8 respectively. In all cases, the median of the rotation shows improvement with the calibration. When looking at the distances, the cases with forward movement show improvement for the calibrated test AGV. The backward movement in case 6 however shows worse accuracy after calibration. An interesting results is also that the spread also seems to have been reduced by the calibration. This is unexpected and an explanation still remains.

Table 8.5: Test results for trajectory case 3. In this table, the values for Δs are denoted in millimetres [mm] and the values for $\Delta\delta$ in [degrees].

	MIN	Q1	MED	Q3	MAX
$\Delta s_{calibrated}$	1,90	4,26	10,30	31,90	46,70
$\Delta\delta_{calibrated}$	0,50	2,03	2,20	2,30	2,30

Table 8.6: Test results for trajectory case 4. In this table, the values for Δs are denoted in millimetres [mm] and the values for $\Delta\delta$ in [degrees].

	MIN	Q1	MED	Q3	MAX
$\Delta s_{calibrated}$	0,10	5,80	10,10	18,78	34,90
$\Delta\delta_{calibrated}$	0,70	2,55	3,20	3,30	3,70

Table 8.7: Test results for trajectory case 5. In this table, the values for Δs are denoted in millimetres [mm] and the values for $\Delta\delta$ in [degrees].

	MIN	Q1	MED	Q3	MAX
$\Delta s_{uncalibrated}$	5,34	9,59	13,22	15,79	38,08
$\Delta s_{calibrated}$	0,10	5,89	8,00	9,49	13,40
$\Delta\delta_{uncalibrated}$	101,10	102,05	103,10	104,50	106,10
$\Delta\delta_{calibrated}$	87,90	88,48	89,70	89,80	90,50

Table 8.8: Test results for trajectory case 6. In this table, the values for Δs are denoted in millimetres [mm] and the values for $\Delta\delta$ in [degrees].

	MIN	Q1	MED	Q3	MAX
$\Delta s_{uncalibrated}$	32,72	41,34	46,17	58,44	100,44
$\Delta s_{calibrated}$	47,50	53,20	55,50	58,41	180,70
$\Delta\delta_{uncalibrated}$	88,50	97,43	98,10	98,40	100,00
$\Delta\delta_{calibrated}$	90,10	92,03	93,20	93,47	94,40

Table 8.9: Test results for trajectory case 7. In this table, the values for Δs are denoted in millimetres [mm] and the values for $\Delta\delta$ in [degrees].

	MIN	Q1	MED	Q3	MAX
$\Delta s_{uncalibrated}$	64,29	123,97	136,15	142,08	178,63
$\Delta s_{calibrated}$	3,50	5,45	9,60	14,32	18,30
$\Delta\delta_{uncalibrated}$	78,50	78,68	80,50	81,95	82,30
$\Delta\delta_{calibrated}$	82,60	84,63	85,80	88,23	91,20

Table 8.10: Test results for trajectory case 8. In this table, the values for Δs are denoted in millimetres [mm] and the values for $\Delta\delta$ in [degrees].

	MIN	Q1	MED	Q3	MAX
$\Delta s_{uncalibrated}$	9,31	64,74	68,29	74,65	123,30
$\Delta s_{calibrated}$	57,70	59,49	62,90	66,93	77,50
$\Delta\delta_{uncalibrated}$	80,50	82,03	82,80	83,30	83,90
$\Delta\delta_{calibrated}$	84,60	85,35	86,90	88,33	90,90

Chapter 9

Concluding remarks

In this chapter, the results of the study into the calibration of AGVs are interpreted and a conclusion is drawn. This is elaborated in section 9.1. The conclusion could only be reached by researching the sub-questions that are drafted in the introduction of this report. During the study, several interesting other topics and improvements were encountered. These topics are discussed in section 9.2.

9.1 Discussion

To improve the odometry of model AGVs in the laboratory, it is requested to investigate the deviation between the perceived servo inputs and outputs. This research has done so with the main research question being:

> *Can the accuracy of model AGVs be improved with the calibration of actuators?*

In chapter 8, the results of a simple comparison between a calibrated and an uncalibrated AGV are given. From the data can be concluded that all cases, the rotation of the model AGV benefits from being calibrated. This conclusion is based on the fact that the actual rotation of a model AGV is closer to the desired location for calibrated AGVs, as opposed to uncalibrated AGVs. The results also show that the deviation in distance is reduced in most cases, with case 6 being an exception to this. Having such a result may indicate that insufficient tests were performed.

Previously it was also mentioned that the spread unexpectedly has been reduced. This can be explained by the fact that the curve fit for the steering servo contains a linear

component. This can cause the spread in results to decrease, as occurs in all the cases. For the vehicle's orientation, the spread is typically between 2 and 4 degrees. The distance typically shows a spread between 1 and 90 millimetres.

In all cases with the exception of one the deviation between the desired end-position and the actual end-position of the calibrated AGV as well as the deviation between the desired end-orientation and the actual end-orientation, has been reduced. This leads to the conclusion that calibration of actuators can indeed improve the accuracy of model AGVs. This conclusion could only be drawn when looking at the following sub-question:

- What is the laboratory set-up and equipment?
- Which method is suited for the calibration of mobile robots?
- How will the calibration method be implemented for the calibration of model AGVs?
- What are the results for the model AGV's calibration?
- Is a calibrated AGV more accurate than an uncalibrated AGV?

The investigation into these questions has occasionally resulted in additional topics and improvements. These additional topics and improvements are distilled and presented in section 9.2

9.2 Recommendations

Following this study into the calibration of AGVs, several interesting topics and improvements were encountered. During the calibration and the tests it was noticed that there still is a significant deviation between similar trajectories. The cause of these deviations are not exactly known, but it is highly likely that insufficiently tightened parts and delays in the Arduino processor are contributors to these deviations. When performing trajectories, it is desired to have a small spread, since uncertainties result in a less predictable trajectory. It is recommended to improve the predictability of trajectories by decreasing the spread.

In Wang and Qi (2001), from which the kinematic model is adopted, is assumed that the slip and difference in wheel diameter can be neglected. Similarly, it was assumed that the acceleration and deceleration would have little effect on the model AGVs trajectory, since the maximum velocity is only small and the acceleration is high. Two additional assumptions that are made in this research are that the axles rotate instantaneous and that the front axle drive servo and rear axle drive servo have similar speeds. It could be investigated whether one or more of these assumptions still hold, when trying to improve the accuracy of calibration further.

Finally, it is hypothesized that a variable delay between sending a command to the Arduino controller and the Arduino controller setting the servo could have an impact on the predictability of trajectory. If the path following algorithm would be aware of this, the accuracy of the model AGVs could be improved.

Bibliography

Arduino. Arduino. URL http://www.arduino.cc/. First accessed March 6, 2015.

Martin John Baker. Maths - rotations, a. URL http://www.euclideanspace.com/maths/geometry/rotations/. First accessed April 16, 2015.

Martin John Baker. Maths - quaternions, b. URL http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/. First accessed April 16, 2015.

Joaquim A. Batlle, Josep M. Font-Llagunes, and Ana Barjau. Calibration for mobile robots with an invariant jacobian. *Robotic and Autonomous Systems*, 58:10–15, 2010. ISSN 0921-8890. doi: 10.1016/j.robot.2009.09.002.

Huiying Chen, Kohsei Matsumoto, Jun Ota, and Tamio Arai. Self-calibration of environmental camera for mobile robot navigation. *Robotic and Autonomous Systems*, 55: 177–190, 2007. ISSN 0921-8890. doi: 10.1016/j.robot.2006.09.003.

Servo Database. Hitec hs-700bb - standard 1/4 scale servo. URL http://www.servodatabase.com/servo/hitec/hs-700bb. First accessed May 18, 2015.

Kaj de Groot. *Route calculation and testing for AGVs*. PhD thesis, Technische Universiteit Delft, Delft, The Netherlands, 2015.

G. Duelen and K. Schröer. Robot calibration - method and results. *Robotics & Computer-Integrated Manufacturing*, 8(4):223–231, 1991. ISSN 0736-5845.

Erik J. Gerritse. *Improving the position estimation of AGVs using an accelerometer*. PhD thesis, Technische Universiteit Delft, Delft, The Netherlands, 2014.

Reza N. Jazar. *Vehicle Dynamics: Theory and Application*. Springer US, 2008. ISBN 978-0-387-74243-4. doi: 10.1007/978-0-387-74244-1_7.

W. Khalil and E. Dombre. Geometric calibration of robots. In *Modeling, Identification and Control of Robots*, volume , chapter 11, pages 257–289. Springer Netherlands, 2002.

Kooktae Lee, Woojin Chung, and Kwanghyun Yoo. Kinematic parameter calibration of a car-like mobile robot to improve odometry accuracy. *Mechatronics*, 20:582–595, 2010. ISSN 0957-4158. doi: 10.1016/j.mechatronics.2010.06.002.

Mathworks. Matlab central - file exchange. URL http://www.mathworks.com/matlabcentral/fileexchange/. First accessed April 26, 2015.

J.-P. Merlet. Calibration. In J.-P. Merlet, editor, *Parallel Robots*, volume 128 of *Solid Mechanics and Its Applications*, chapter 10, pages 289–300. Springer Netherlands, 2006.

Patrick F. Muir and Charles P. Neuman. Kinematic modelling of wheeled mobile robots. June 1986.

NatNet. Natnet software development kit. URL http://www.optitrack.com/products/natnet-sdk/. First accessed March 6, 2015.

NaturalPoint Inc. Optitrack motion capture system.

Maria Isabel Ribeiro and Pedro Lima. Kinematics models of mobile robots. April 2002.

Boris A. Rosenfeld. *A History of Non-Euclidean Geometry: Evolution of the Concept of a Geometric Space*. Springer New York, english edition edition, 1988. ISBN 9780387964584.

Edward J. Rykiel. Testing ecological models: the meaning of validation. *Ecological Modelling*, 90:229–244, 1996. ISSN 0304-3800.

Science Direct. URL http://www.sciencedirect.com/. First accessed March 25, 2015.

SpringerLink. URL http://link.springer.com/. First accessed March 26, 2015.

Y.D. Sumith. Fast circle fitting using landau method. URL http://www.mathworks.com/matlabcentral/fileexchange/44219-fast-circle-fitting-using-landau-method. First accessed May 3, 2015.

Samuel M. Tomas and Y.T. Chan. A simple approach for the estimation of circular arc center and its radius. *Computer Vision, Graphics, and Image Processing*, 45(3): 362–370, 1989. doi: 10.1016/0734-189X(89)90088-1.

TU Delft Library. URL http://library.tudelft.nl/. First accessed March 24, 2015.

Maarten van Blijswijk. *A proposal of a generic Matlab-Arduino interface for use in the AGV lab*. PhD thesis, Technische Universiteit Delft, Delft, The Netherlands, 2015.

M. Saeed Varziri and Leila Notash. Kinematic calibration of a wire-actuated parallel robot. *Mechanism and Machine Theory*, 42:960–976, 2007. ISSN 0094-114X. doi: 10.1016/j.mechmachtheory.2006.07.007.

Danwei Wang and Feng Qi. Trajectory planning for a four-wheel-steering vehicle. *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, May 2001.

Web of Science. URL https://webofknowledge.com/. First accessed March 25, 2015.

Appendix A

Matlab code

This appendix presents all the code that was written for the calibration of AGVs. The code is initiated by running TrackingMain, which then calls the subsequent functions in this appendix.

A.1 TrackingMain

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%% Matlab main file for tracking the position of AGV's in the laboratory.
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
clc

%% Define global variables
global frameRate;
global desiredFrameRate;
global vehicleID;
global totalDropped;
global totalDuplicate;
global AGV;

%% Load Parameters
Parameters

speedFig  = false; % if true, calculate the speed
```

```
storeData = false; % if true, save data in excel file
calibrate = true;  % if true, perform calibration

%% Initialization of the client
[theClient, errCode] = InitializeConnection(ClientIP, ServerIP, CastMethod);
if errCode ~= 0
    return
end

%% Retrievel of information
frameRate = GetFrameRate(theClient);
fprintf('\n[NatNet] OptiTrack Frame Rate : %d fps\n\n', frameRate)
rigidBodies = RetrieveObjectData(theClient);

%% Initialize figure and hold
locationFigure = figure('Name','AGV Position Data','NumberTitle','off');

%% Create frame ready event handler
ls = addlistener(theClient,'OnFrameReady2',@(src,event)FrameReadyCallback(src,event));
uiwait(locationFigure);

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

% Display total dropped frames and duplicate frames
fprintf('\n[NatNet] Total Dropped Frames   : %d frame(s)', totalDropped);
fprintf('\n[NatNet] Total Duplicate Frames : %d frame(s)', totalDuplicate);

% Disconnect with NatNet server. Improper termination will lead to Matlab errors!
UninitializeConnection(theClient);
clear functions


%% Process data

if calibrate
    % perform a circle fit on the data and create circle data for plotting.
    [Cx, Cz, R] = Landau_Smith(AGV(vehicleID).x, AGV(vehicleID).z);
    theta = 0 : pi/180 : 2*pi;
    xcircle = R * cos(theta) + Cx;
    zcircle = R * sin(theta) + Cz;

    % calculate the formula of the lines that represent the parallel and
    % tangent to the AGV's direction.
    arrayIndex = length(AGV(vehicleID).yaw);
    m = tan(AGV(vehicleID).yaw(arrayIndex));
    n = AGV(vehicleID).z(arrayIndex) - m * AGV(vehicleID).x(arrayIndex);
    z1 = m * AGV.rangeX + n;
    p = - 1 / m;
    q = Cz - (p * Cx);
```

```
76      z2 = p * AGV.rangeX + q;
77
78      Ix = (q - n) / (m - p);
79      Iz = m * Ix + n;
80      Fx = AGV(vehicleID).x(arrayIndex) + (a1 * cos(AGV(vehicleID).yaw(arrayIndex)));
81      Fz = AGV(vehicleID).z(arrayIndex) + (a1 * sin(AGV(vehicleID).yaw(arrayIndex)));
82      Rx = AGV(vehicleID).x(arrayIndex) - (a2 * cos(AGV(vehicleID).yaw(arrayIndex)));
83      Rz = AGV(vehicleID).z(arrayIndex) - (a2 * sin(AGV(vehicleID).yaw(arrayIndex)));
84
85      % calculate the distance between intersect point I and the front and
86      % rear axles to determine the location of point I, also calculate the
87      % shortest distance between the circle center and point I.
88      distFront  = sqrt((Fx - Ix)^2 + (Fz - Iz)^2);
89      distRear   = sqrt((Rx - Ix)^2 + (Rz - Iz)^2);
90      distCenter = sqrt((Cx - Ix)^2 + (Cz - Iz)^2);
91
92      if distFront == 0
93          % intersect point I is exactly on front axle and therefore the
94          % rotation is zero, but the rotation of the rear axle is non-zero.
95          state = 1;
96          angleF = 0;
97          angleR = atan2((a1 + a2), distCenter);
98
99      elseif distRear == 0
100         % intersect point I is exactly on rear axle and therefore the
101         % rotaton is zero, but thre rotation of the front axle is non-zero.
102         state = 2;
103         angleF = atan2((a1 + a2), distCenter);
104         angleR = 0;
105
106     elseif and(distFront < (a1 + a2), distRear < (a1 + a2));
107         % intersect point I is between the front and rear axle, both axles
108         % are now rotated with opposite signs.
109         state = 3;
110         angleF = atan2(distFront, distCenter);
111         angleR = -atan2(distRear, distCenter);
112
113     elseif and(distFront < distRear, distRear > (a1 + a2));
114         % intersect point I is in front of the front axle, both axles are
115         % now rotated in similar direction but with different angles and
116         % negative sign.
117         state = 4;
118         angleF = -atan2(distFront, distCenter);
119         angleR = -atan2(distRear,  distCenter);
120
121     elseif and(distFront > distRear, distFront > (a1 + a2));
122         % intersect point I is behind the rear axle, both axles are now
123         % rotated in a similar direction but with different angles and a
124         % positive sign.
125         state = 5;
126         angleF = atan2(distFront, distCenter);
127         angleR = atan2(distRear,  distCenter);
128
```

```
129     end
130     side = sign(((Fx - Rx) * (Cz - Rz)) - ((Fz - Rz) * (Cx - Rx)));
131     angleF = side * ((angleF / pi) * 180); % express angle in degrees with correct sign
132     angleR = side * ((angleR / pi) * 180); % express angle in degrees with correct sign
133 
134     % recreate location data figure with similar labels and dimensions.
135     locationFigure = figure('Name','AGV Position Data','NumberTitle','off');
136 
137     hold on
138 
139     title('Mocap Position Plot')
140     xlabel('X position [m]')
141     ylabel('Z position [m]')
142     set(gca,'YLim',[-2.5 2.5])
143     set(gca,'XLim',[-2.5 2.5])
144     set(gca,'XGrid','on','YGrid','on')
145     axis square
146 
147     % plot all available data into new figure.
148     plot(AGV(vehicleID).x, AGV(vehicleID).z);
149     plot(AGV.rangeX, z1);
150     plot(AGV.rangeX, z2);
151     plot(xcircle, zcircle);
152     plot([Fx Rx], [Fz Rz]);
153     plot([Fx Cx], [Fz Cz]);
154     plot([Rx Cx], [Rz Cz]);
155 
156     hold off
157 
158     fprintf('[Calibration] State of rotation : state %d\n', state)
159     fprintf('[Calibration] Front axis angle  : %4.2f degrees\n', angleF)
160     fprintf('[Calibration] Rear axis angle   : %4.2f degrees\n', angleR)
161 
162 
163     if not(isempty(AGV))
164         for i = 2 : length(AGV(vehicleID).time)
165             if AGV(vehicleID).time(i) - AGV(vehicleID).time(i-1) > 0.000001
166                 ds = sqrt((AGV(vehicleID).x(i) - AGV(vehicleID).x(i-1))^2 + (AGV(vehicle
167                 dt = AGV(vehicleID).time(i) - AGV(vehicleID).time(i-1);
168                 AGV(vehicleID).velocity(i) = ds / dt;
169             else
170                 AGV(vehicleID).velocity(i) = 0;
171             end
172         end
173         if speedFig
174             plot(AGV(vehicleID).time,AGV(vehicleID).velocity)
175         end
176     end
177     velocity = median(AGV(vehicleID).velocity);
178     fprintf('[Calibration] Velocity   : %4.2f m/s\n', velocity)
179 end
180 
181 %% Write to file
```

```
182 if storeData
183     if not(isempty(AGV))
184         xlswrite(filename, transpose(AGV(vehicleID).time),     sheet, 'A1');
185         xlswrite(filename, transpose(AGV(vehicleID).x),        sheet, 'C1');
186         xlswrite(filename, transpose(AGV(vehicleID).z),        sheet, 'D1');
187         xlswrite(filename, transpose(AGV(vehicleID).yaw),      sheet, 'E1');
188         xlswrite(filename, transpose(AGV(vehicleID).velocity), sheet, 'F1');
189     end
190 end
```

A.2 Parameters

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%
3  %%% Matlab parameters file for position tracking system
4  %%%
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6
7
8  % IP addresses of server and client. The client is the computer which
9  % receives the postion data from Motive:Tracker and the server is the
10 % computer which sends the data.
11
12
13 %% Connection parameters
14
15 ClientIP = char('145.94.185.77');
16 ServerIP = char('131.180.120.154');
17
18 CastMethod = 1; %0 = Multicast, 1 = Unicast
19
20 %% Random framerate
21
22 desiredFrameRate = 120; % in fps. Do not increase this value above 120 fps.
23
24 %% Calibration
25 vehicleID = 1;
26
27 a1 = 0.181; % distance from center of mass to front axle
28 a2 = 0.121; % distance from center of mass to rear axle
29
30 %% File properties (xls-file with position data)
31 filename = 'Calibration.xlsx';
32 sheet = 1;
33
34 %% Figure parameters
35 useFigure = true;
```

A.3 InitializeConnection

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%% Matlab function to initialize the connection between Matlab and
%%% Motive:Tracker by using NatNet SDK.
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Input is the local IP-address of the client computer and the server
% computer, the output is the NatNet Client library.

function [theClient, errCode] = InitializeConnection(ClientIP, ServerIP, Method)
    try
        % Determin the version of Matlab on Windows to match with the
        % correct library. Display commands are for debugging.
        syst = computer;
        errCode = 0;
        if strcmp(syst,'PCWIN')
            display('[NatNet] Running on 32-bit Windows.')
            PCversion = 'x86';
        elseif strcmp(syst,'PCWIN64')
            display('[NatNet] Running on 64-bit Windows.')
            PCversion = 'x64';
        else
            display('[NatNet] Unsupported Operating System.')
            errCode = 1;
            return
        end

        display('[NatNet] Creating Client.')

        curDir = pwd;
        mainDir = curDir;
        dllPath = fullfile(mainDir,'lib',PCversion,'NatNetML.dll');
        assemblyInfo = NET.addAssembly(dllPath);

        % Create an instance of a NatNet client
        theClient = NatNetML.NatNetClientML(Method); % Input = iConnectionType: 0 = Mult
        version = theClient.NatNetVersion();
        fprintf('[NatNet] Client Version : %d.%d.%d.%d\n',version(1),version(2),version(

        % Connect to an OptiTrack server (e.g. Motive)
        display('[NatNet] Connecting to OptiTrack Server.')
        flg = theClient.Initialize(ClientIP, ServerIP); % flg = returnCode: 0 = Success
        if (flg == 0)
            display('[NatNet] Initialization Succeeded')
        else
            display('[NatNet] Initialization Failed')
            errCode = 2;
            return
```

```
50          end
51
52      % Catch any errors that might occur during the execution of the
53      % function and display them.
54      catch err
55          display(err);
56      end
57  end
```

A.4 GetFrameRate

```
1   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2   %%%
3   %%% Matlab file for retrieving the frame rate of the camera system.
4   %%%
5   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6
7   % Input is the client information and output is the frame rate.
8
9   function [frameRate] = GetFrameRate(theClient)
10      [byteArray, retCode] = theClient.SendMessageAndWait('FrameRate');
11      if(retCode == 0)
12          byteArray = uint8(byteArray);
13          frameRate = typecast(byteArray, 'single');
14      end
15  end
```

A.5 RetrieveObjectData

```
1   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2   %%%
3   %%% Matlab function to retrieve data from the Motive:Tracker server.
4   %%%
5   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6
7   % Input is the client data from NatNet SDK, output is the number of rigid
8   % bodies that are known to Motive:Tracker.
9
10  function [maxBodies] = RetrieveObjectData(theClient)
11      dataDescriptions = theClient.GetDataDescriptions();
12      maxBodies = 0;
13
14      % print data of the available markers and rigid bodies
15      fprintf('[NatNet] Tracking Models : %d\n\n', dataDescriptions.Count);
16
```

```
17      for idx = 1 : dataDescriptions.Count
18          descriptor = dataDescriptions.Item(idx-1);
19          if(descriptor.type == 0)
20              fprintf('\n\tMarkerset : %s\t(%d markers)\n', char(descriptor.Name), descrip
21              markerNames = descriptor.MarkerNames;
22              for markerIndex = 1 : descriptor.nMarkers
23                  name = markerNames(markerIndex);
24                  fprintf('\t\tMarker : %-20s\t(ID=%d)\n', char(name), markerIndex);
25              end
26          elseif(descriptor.type == 1)
27              maxBodies = max(maxBodies, descriptor.ID);
28              fprintf('\n\tRigid Body : %s\t\t(ID=%d, ParentID=%d)\n', char(descriptor.Nam
29          elseif(descriptor.type == 2)
30              fprintf('\n\tSkeleton : %s\t(%d bones)\n', char(descriptor.Name), descriptor
31              fprintf('\t\tID : %d\n', descriptor.ID);
32              rigidBodies = descriptor.RigidBodies;
33              for boneIndex = 1 : descriptor.nRigidBodies
34                  rigidBody = rigidBodies(boneIndex);
35                  fprintf('\t\tBone : %-20s\t(ID=%d, ParentID=%d)\n', char(rigidBody.Name)
36              end
37          end
38      end
39  end
```

A.6 FrameReadyCallback

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%
3  %%% Matlab frame ready callback function.
4  %%%
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6
7  function FrameReadyCallback(src, event)
8      frameOfData = event.data;
9      DataProcessing(frameOfData);
10 end
```

A.7 DataProcessing

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%
3  %%% Matlab data processing.
4  %%%
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6
```

```
7  % This function is called every time a new frame from OptiTrack on a
8  % specific port is detected by Matlab. The function is not a loop, but
9  % retains its variables via the "persistent" and "global" declaration.
10 function DataProcessing(frameOfData)
11     persistent lastFrameTime;
12     persistent initialFrameTime;
13     persistent arrayIndex;
14     persistent frameCounter;
15     persistent fitCounter;
16
17     global frameRate;
18     global desiredFrameRate;
19     global vehicleID;
20     global totalDropped;
21     global totalDuplicate;
22     global AGV;
23
24     % Create the initial figure and location for first start-up. Will not
25     % be called after the first time.
26     if isempty(AGV)
27         lastFrameTime = frameOfData.fLatency;
28         initialFrameTime = frameOfData.fLatency;
29         totalDropped = 0;
30         totalDuplicate = -1; % the first frame is always duplicate, therefore compensate with
31         arrayIndex = 1;
32         frameCounter = 0;
33         fitCounter = 0;
34         AGV.rangeX = -2.5:0.1:2.5; % define axis for drawing calibration lines
35
36         % Create initial plot of the AGV location to prevent Matlab drawing
37         % a line between [0,0] and the AGV location. And create a compass
38         % figure that displays the rotation of vehicles.
39         if(frameOfData.RigidBodies.Length() > 0)
40             hold on
41             for ID_init = vehicleID : vehicleID
42                 rigidBodyData = frameOfData.RigidBodies(ID_init);
43
44                 % Determine the x and the z coordinates of the AGVs that
45                 % correspond to the horizontal plane locations.
46                 AGV(ID_init).x(1) =  rigidBodyData.x;
47                 AGV(ID_init).z(1) = -rigidBodyData.z; % mirror the location Z-axis so that the
48                 AGV(ID_init).Location = plot(AGV(ID_init).x, AGV(ID_init).z);
49
50                 % Determine yaw of the vehicle by testing for a singularity
51                 % (either north pole or south pole). The quaternion
52                 % information from OptiTrack is transformed to euler
53                 % angles.
54                 pole = rigidBodyData.qx * rigidBodyData.qy + rigidBodyData.qz * rigidBodyData.
55                 if pole > 0.499
56                     AGV(ID_init).yaw(1) = 2 * atan2(rigidBodyData.qx, rigidBodyData.qw);
57                 elseif pole < -0.499
58                     AGV(ID_init).yaw(1) = -2 * atan2(rigidBodyData.qx, rigidBodyData.qw);
59                 else
```

```
60                      AGV(ID_init).yaw(1) = atan2(2 * rigidBodyData.qy * rigidBodyData.qw
61                  end
62
63                  m = tan(AGV(ID_init).yaw(1));
64                  n = AGV(ID_init).z(1) - m * AGV(ID_init).x(1);
65                  z1 = m * AGV.rangeX + n;
66                  AGV(ID_init).Rotation = plot(AGV.rangeX, z1);
67              end
68
69              title('Mocap Position Plot')
70              xlabel('X position [m]')
71              ylabel('Z position [m]')
72              set(gca,'YLim',[-2.5 2.5])
73              set(gca,'XLim',[-2.5 2.5])
74              set(gca,'XGrid','on','YGrid','on')
75              axis square
76              hold off
77          end
78      end
79
80      % As this function is recalled and several variables are not declared
81      % as persistent, these variables need to be created before they are
82      % used. This function initially assumes that a newFrame is false, and
83      % calculates if this is indeed a new frame later on.
84      newFrame = false;
85      droppedFrames = false;
86      frameTime = frameOfData.fLatency;
87      calcFrameInc = round((frameTime - lastFrameTime) * frameRate);
88
89      % This part of the code counts the number of times this function is
90      % called and matches this with the desired framerate that can be
91      % defined in the parameters file.
92      frameCounter = frameCounter + 1;
93      if frameCounter == ceil(frameRate/desiredFrameRate)
94          arrayIndex = arrayIndex + calcFrameInc;
95          if(arrayIndex==0)
96              arrayIndex = 1;
97          end
98          frameCounter = 0;
99          newFrame = true;
100     end
101
102     % Determine whether a frame is dropped or duplicate, else it is a
103     % marked as a new frame.
104     if(calcFrameInc > 1)
105         %fprintf('\nDropped Frame(s) : %d\n\tLastTime : %.3f\n\tThisTime : %.3f\n', calc
106         droppedFrames = true;
107         totalDropped = totalDropped + 1;
108     elseif(calcFrameInc == 0)
109         %display('Duplicate Frame')
110         newFrame = false;
111         totalDuplicate = totalDuplicate + 1;
112     end
```

```
113
114     % Try for a new frame and test if rigid bodies are present in data.
115     % Then for each rigid body retrieve position and rotation data from
116     % OptiTrack. The information is then displayed in a figure. If an error
117     % occurs, it is catched and presented without stopping the program.
118     try
119         if(newFrame)
120             if(frameOfData.RigidBodies.Length() > 0)
121                 for ID = vehicleID : vehicleID
122                     % Read data of RigidBody ID.
123                     rigidBodyData = frameOfData.RigidBodies(ID);
124
125                     % If dropped frames have occured, fill in the index
126                     % with values.
127                     if(droppedFrames)
128                         % Fill the array with values when dropped frames occurs.
129                         for i = 1 : calcFrameInc
130                             fillIndex = arrayIndex - i;
131                             if(fillIndex > 1)
132                                 AGV(ID).x(fillIndex) =  rigidBodyData.x;
133                                 AGV(ID).z(fillIndex) = -rigidBodyData.z; % mirror the location
134                                 AGV(ID).time(fillIndex) = lastFrameTime - initialFrameTime; %
135
136                                 pole = rigidBodyData.qx * rigidBodyData.qy + rigidBodyData.qz
137                                 if pole > 0.499
138                                     AGV(ID).yaw(fillIndex) = 2 * atan2(rigidBodyData.qx, rigidB
139                                 elseif pole < -0.499
140                                     AGV(ID).yaw(fillIndex) = -2 * atan2(rigidBodyData.qx, rigid
141                                 else
142                                     AGV(ID).yaw(fillIndex) = atan2(2 * rigidBodyData.qy * rigid
143                                 end
144                             end
145                         end
146                     end
147                     AGV(ID).x(arrayIndex) =  rigidBodyData.x;
148                     AGV(ID).z(arrayIndex) = -rigidBodyData.z; % mirror the location Z-axis so
149                     AGV(ID).time(arrayIndex) = frameTime - initialFrameTime;
150                     set(AGV(ID).Location,'XData',AGV(ID).x,'YData',AGV(ID).z)
151
152                     % Calculate the yaw of the AGV, regarding
153                     % singularities.
154                     pole = rigidBodyData.qx * rigidBodyData.qy + rigidBodyData.qz * rigidBodyDa
155                     if pole > 0.499
156                         AGV(ID).yaw(arrayIndex) = 2 * atan2(rigidBodyData.qx, rigidBodyData.qw
157                     elseif pole < -0.499
158                         AGV(ID).yaw(arrayIndex) = -2 * atan2(rigidBodyData.qx, rigidBodyData.qw
159                     else
160                         AGV(ID).yaw(arrayIndex) = atan2(2 * rigidBodyData.qy * rigidBodyData.qw
161                     end
162
163                     m = tan(AGV(ID).yaw(arrayIndex));
164                     n = AGV(ID).z(arrayIndex) - (m * AGV(ID).x(arrayIndex));
165                     z1 = m * AGV.rangeX + n;
```

```
166                     set(AGV(ID).Rotation,'XData',AGV.rangeX,'YData',z1)
167                 end
168             end
169         end
170     catch err
171         display(err);
172     end
173 
174     lastFrameTime = frameTime;
175 end
```

A.8 UninitializeConnection

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %%%
3  %%% Matlab function to uninitialize the connection between Matlab and
4  %%% Motive:Tracker by using NatNet SDK.
5  %%%
6  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7  
8  % Input is the client data from NatNet SDK
9  
10 function UninitializeConnection(theClient)
11     theClient.Uninitialize();
12     fprintf('\n[NatNet] Connection terminated\n\n')
13 end
```

Appendix B

Detailed velocity tables

Each of the following tables has the single-, double- and crab-steer possibilities presented for 7 different input rotations. Each table also presents this for two different driving velocity inputs. The outputs are medians of the velocity in m/s. A total of 16 different velocities are tested in these tables, for 3 steering possibilities and 7 rotations. This gives 336 results.

ANGLE	VELOCITY [m/s] for 900			VELOCITY [m/s] for 1200		
	SINGLE	DOUBLE	CRAB	SINGLE	DOUBLE	CRAB
900	0,28	0,24	0,31	0,27	0,24	0,30
1100	0,29	0,26	0,32	0,29	0,26	0,31
1300	0,31	0,31	0,32	0,31	0,30	0,31
1500	0,32	0,32	0,32	0,32	0,32	0,32
1700	0,31	0,31	0,32	0,31	0,30	0,31
1900	0,29	0,26	0,32	0,29	0,26	0,31
2100	0,28	0,24	0,31	0,27	0,24	0,30

ANGLE	VELOCITY [m/s] for 1250			VELOCITY [m/s] for 1300		
	SINGLE	DOUBLE	CRAB	SINGLE	DOUBLE	CRAB
900	0,25	0,21	0,27	0,22	0,20	0,25
1100	0,26	0,24	0,28	0,23	0,22	0,25
1300	0,28	0,27	0,28	0,26	0,25	0,26
1500	0,29	0,29	0,29	0,26	0,26	0,26
1700	0,28	0,27	0,28	0,26	0,25	0,26
1900	0,26	0,24	0,28	0,23	0,22	0,25
2100	0,25	0,21	0,27	0,22	0,20	0,25

ANGLE	VELOCITY [m/s] for 1350			VELOCITY [m/s] for 1400		
	SINGLE	DOUBLE	CRAB	SINGLE	DOUBLE	CRAB
900	0,19	0,17	0,22	0,13	0,11	0,15
1100	0,20	0,18	0,22	0,14	0,12	0,15
1300	0,22	0,21	0,22	0,16	0,15	0,16
1500	0,22	0,22	0,22	0,16	0,16	0,16
1700	0,22	0,21	0,22	0,16	0,15	0,16
1900	0,20	0,18	0,22	0,14	0,12	0,15
2100	0,19	0,17	0,22	0,13	0,11	0,15

ANGLE	VELOCITY [m/s] for 1450			VELOCITY [m/s] for 1500		
	SINGLE	DOUBLE	CRAB	SINGLE	DOUBLE	CRAB
900	0,05	0,03	0,07	0,00	0,00	0,00
1100	0,06	0,04	0,07	0,00	0,00	0,00
1300	0,06	0,05	0,07	0,00	0,00	0,00
1500	0,07	0,07	0,07	0,00	0,00	0,00
1700	0,06	0,05	0,07	0,00	0,00	0,00
1900	0,06	0,04	0,07	0,00	0,00	0,00
2100	0,05	0,03	0,07	0,00	0,00	0,00

ANGLE	VELOCITY [m/s] for 1550			VELOCITY [m/s] for 1600		
	SINGLE	DOUBLE	CRAB	SINGLE	DOUBLE	CRAB
900	0,00	0,00	0,00	0,09	0,07	0,10
1100	0,00	0,00	0,00	0,10	0,09	0,11
1300	0,00	0,00	0,00	0,12	0,11	0,12
1500	0,01	0,01	0,01	0,12	0,12	0,12
1700	0,00	0,00	0,00	0,12	0,11	0,12
1900	0,00	0,00	0,00	0,10	0,09	0,11
2100	0,00	0,00	0,00	0,09	0,07	0,10

ANGLE	VELOCITY [m/s] for 1650			VELOCITY [m/s] for 1700		
	SINGLE	DOUBLE	CRAB	SINGLE	DOUBLE	CRAB
900	0,15	0,13	0,18	0,20	0,17	0,21
1100	0,16	0,15	0,18	0,22	0,20	0,22
1300	0,18	0,18	0,19	0,21	0,23	0,23
1500	0,19	0,19	0,19	0,24	0,24	0,24
1700	0,18	0,18	0,19	0,21	0,23	0,23
1900	0,16	0,15	0,18	0,22	0,20	0,22
2100	0,15	0,13	0,18	0,20	0,17	0,21

ANGLE	VELOCITY [m/s] for 1750			VELOCITY [m/s] for 1800		
	SINGLE	DOUBLE	CRAB	SINGLE	DOUBLE	CRAB
900	0,23	0,19	0,26	0,25	0,21	0,28
1100	0,25	0,23	0,27	0,27	0,25	0,29
1300	0,27	0,26	0,28	0,29	0,29	0,29
1500	0,28	0,28	0,28	0,30	0,30	0,30
1700	0,27	0,26	0,28	0,29	0,29	0,29
1900	0,25	0,23	0,27	0,27	0,25	0,29
2100	0,23	0,19	0,27	0,25	0,21	0,28

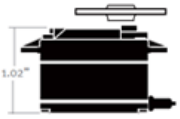
ANGLE	VELOCITY [m/s] for 1850			VELOCITY [m/s] for 2100		
	SINGLE	DOUBLE	CRAB	SINGLE	DOUBLE	CRAB
900	0,27	0,23	0,30	0,27	0,23	0,30
1100	0,30	0,27	0,31	0,30	0,27	0,31
1300	0,31	0,31	0,31	0,31	0,31	0,31
1500	0,32	0,32	0,32	0,32	0,32	0,32
1700	0,31	0,31	0,31	0,31	0,31	0,31
1900	0,30	0,27	0,31	0,30	0,27	0,31
2100	0,27	0,23	0,30	0,27	0,23	0,30

Appendix C

Servo information

ServoDatabase.com

RC Servo Specifications and Reviews



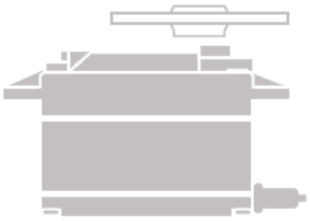
Find a servo: Google™ Custom Search Search Advanced Search Your comparison engine (0)

Servo Database › Hitec Servos › HS-700BB

Hitec HS-700BB - Standard 1/4 Scale Servo

Basic Information

Modulation:	Analog
Torque:	**4.8V:** 138.9 oz-in (10.00 kg-cm) **6.0V:** 173.6 oz-in (12.50 kg-cm)
Speed:	**4.8V:** 0.22 sec/60° **6.0V:** 0.17 sec/60°
Weight:	3.60 oz (102.1 g)
Dimensions:	Length: 2.32 in (58.9 mm) Width: 1.14 in (29.0 mm) Height: 2.04 in (51.8 mm)
Motor Type:	3-pole
Gear Type:	Plastic
Rotation/Support:	Single Bearing



Submit Photo

Brand:	HITEC
Product Number:	? (add)
Suggested Retail:	? (add)
Street Price:	25.95 USD
Compare:	add

Additional Specifications

Rotational Range:	180°
Pulse Cycle:	20 ms
Pulse Width:	900-2100 µs
Connector Type:	? (add)

User Reviews

Number of Reviews:	0
Average Rating:	n/a

User Reviews of Hitec HS-700BB (0)

Submit your review of Hitec HS-700BB »

Appendix D

Instructions

D.1 How to set up the OptiTrack system

How to set up the OptiTrack system

This document describes how to set up the OptiTrack system for use in the AGV laboratory at the 3mE faculty of the Delf University of Technology.

General information

This tutorial learns you how to set up the OptiTrack system with Flex 13 cameras (1.3 MP and 120 FPS). For documentation and references, visit the website http://www.optitrack.com/. Depending on the size of the marker, a maximum tracking distance is guaranteed by the company. This distance is denoted in the table below. The metric values are defined similarly.

Table 1 – Maximum tracking distance for each marker size, measured from the face of the camera.

Distance	28 ft.	35 ft.	39,5 ft.
Marker size	7/16 in.	5/8 in.	3/4 in.

Distance	8.5 m.	11 m.	12 m.
Marker size	11.1125 mm.	15.87500 mm.	19.05 mm.

For the system to work properly, a clean and non-reflective floor is requires to ensure that all returning infrared light (850 nm) is solely from the tracking markers, however it is impossible to fully eliminate all infrared sources. It could prove useful to eliminate outdoor light by blocking the windows.

Figure 1 – Clean and non-reflective environment.

The OptiTrack systems can be set up in multiple ways with different configurations. Typical setups are denoted in the table below with the associated capture volume.

Table 2 – Different configurations for the OptiTrack system setup, depending on the amount of cameras.

Number of cameras	**4**	**6**	**8**	**12**
Setup	Circular	Circular	Rectangular	Rectangular
Capture volume [ft.]	12 Ø x 7+	16 Ø x 7+	24 x 24 x 8+	24 x 24 x 8+
Setup area [ft.]	20 Ø	20 Ø	24 x 24	24 x 24
Capture volume [m]	3,66 Ø x 2,14+	4,88 Ø x 2,14+	7,3 x 7,3 x 2,4+	7,3 x 7,3 x 2,4+
Setup area [m]	6,1 Ø	6,1 Ø	7,3 x 7,3	7,3 x 7,3
Trackable bodies	4	5	6	8

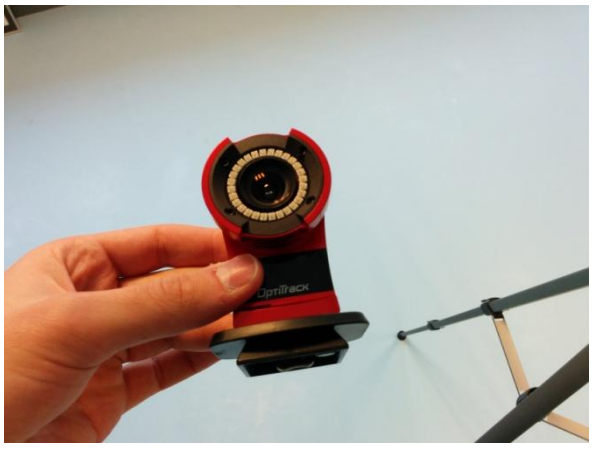
Setup of a specific configuration

The following part is intended for the step by step set up of the OptiTrack Flex 13 system. To ensure proper installation, this guide is illustrated.

Step 1 – Unpack the tripod and set it to its maximum height.

Step 2 – Remove the socket from the tripod head.

Step 3 – Use the thread to mount the camera on the socket and then mount the socket on the tripod.

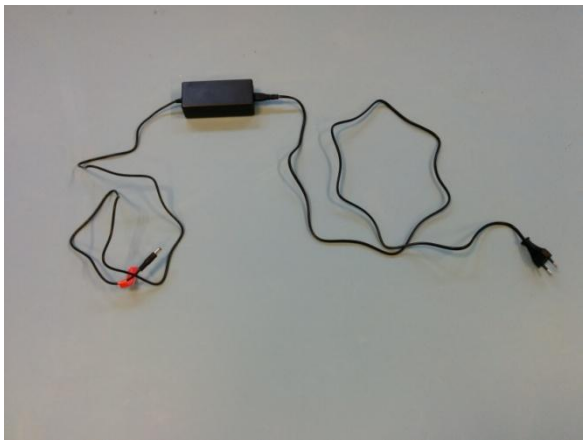


Step 4 – Place the cameras at the desired locations.

Step 5 – For each camera, connect the mini USB cable to the camera and make sure that cable strain at the connector is relieved by tying the Velcro tightly to the tripod (see picture).

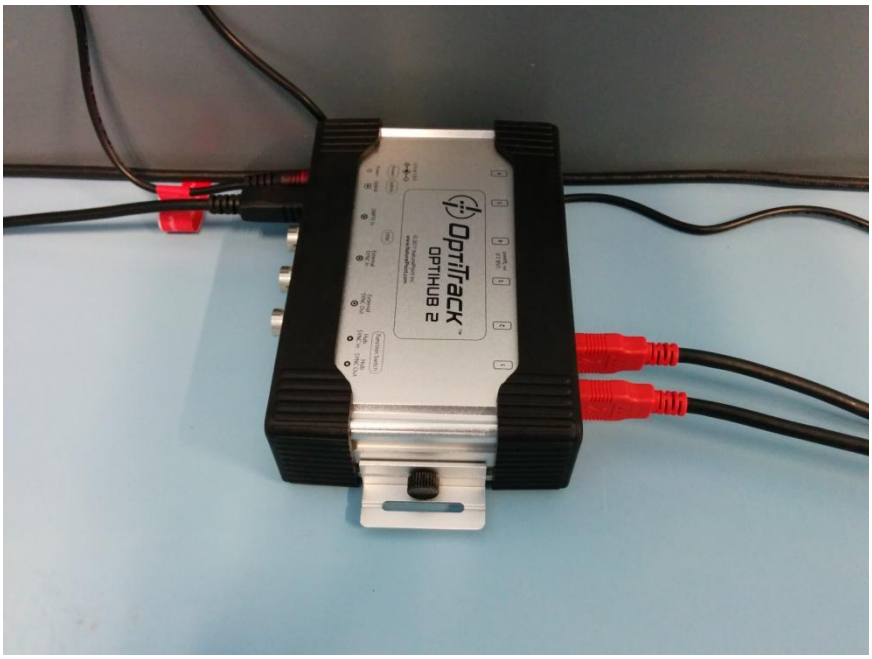
Step 6 – Connect each red USB connector to the OptiTrack OptiHub starting from 1 to 6. This cable may not exceed the total length of 5 meter, as the signal will be lost. If the OptiHub is too far from the camera, consider placing another OptiHub and different camera grouping.

Step 7 – Connect the power supply (left) to the correct input of the OptiHub.



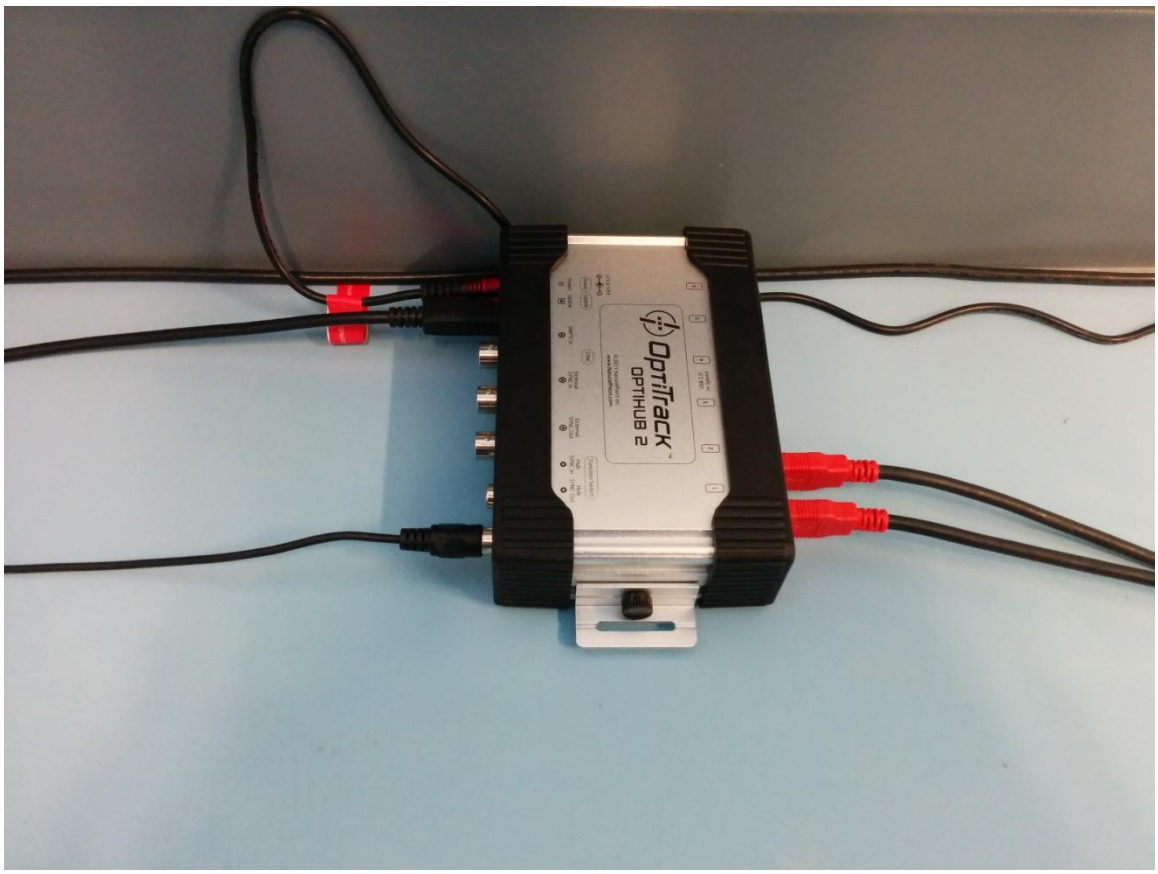
Step 8 – Connect the data cable (left) Type B USB connector to the correct input of the OptiHub. The other end, regular Type A USB cable, can be inserted into the computer USB port.

A single OptiHub with 2 connected cameras, a power cable and data cable will look like the figure below.



When multiple OptiHubs are used, synchronization has to be performed with the synchronization cable (left). The cable has to be connected between the master device output (red) and the slave device input (blue). When synchronizing three or more OptiHubs, form a serial connection in which the output of a slave device is the input of the next device and so on. Do not connect the last slave device to the master device.

A fully connected master device should look like the OptiHub depicted in the figure below.



When using multiple OptiHubs, connect each individual OptiHub directly to the computer using the data cable. If this cable proves to be insufficient in length (standard length is 5 meter), attach a USB 2.0 extension cable (figure below) to the cable. Each extension cable is 5 meter and the total length

between OptiHub and computer may not exceed 15 meter. Keep in mind that the USB port then has to be connected directly to the computers motherboard (without internal cables) to prevent data loss.

Connect multiple OptiHubs to the computer or laptop by running the data cable of each individual OptiHub to the computer USB port. Note that you should not connect the data cable of an OptiHub to the camera port of a different OptiHub.

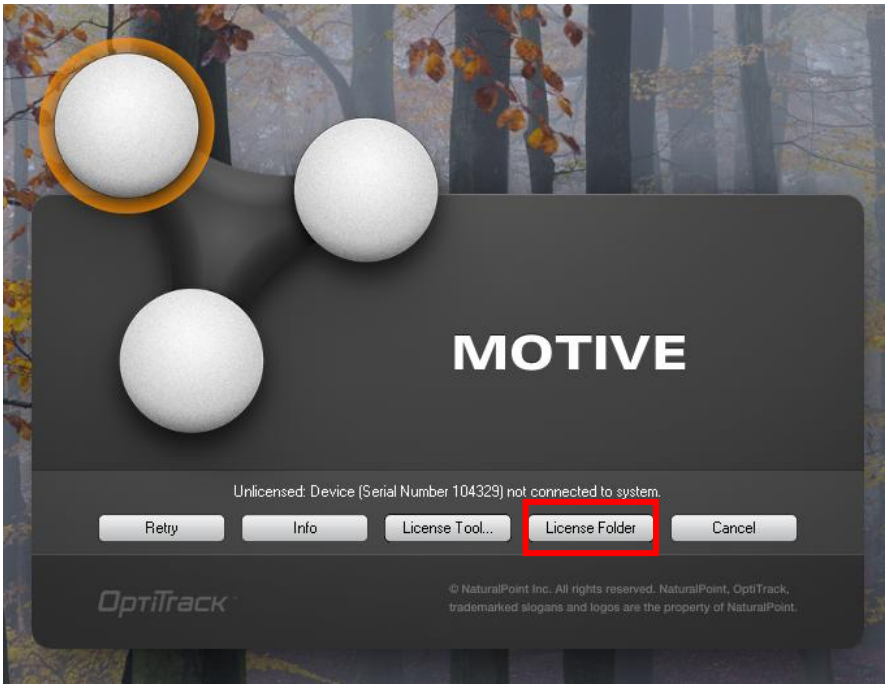
D.2 How to calibrate the OptiTrack system

How to calibrate the OptiTrack system

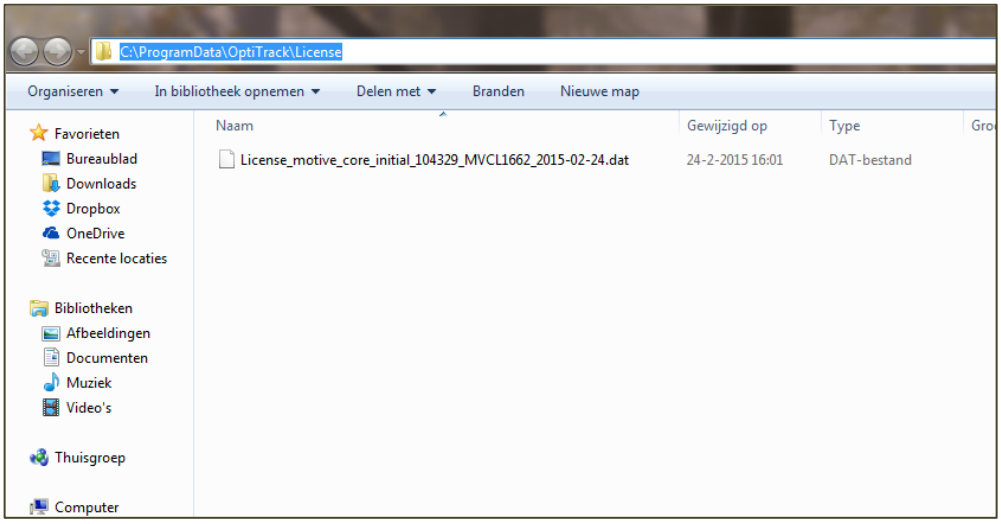
Continuing on from the previous guide, where setting up the OptiTrack system is explained, this document shows how to calibrate the system.

General information

For the system calibration, it is required to download the Motive:Tracker software from the website http://www.optitrack.com/. When the download is complete, install the software. When the installation is finished, open the program and click the “License folder” button.



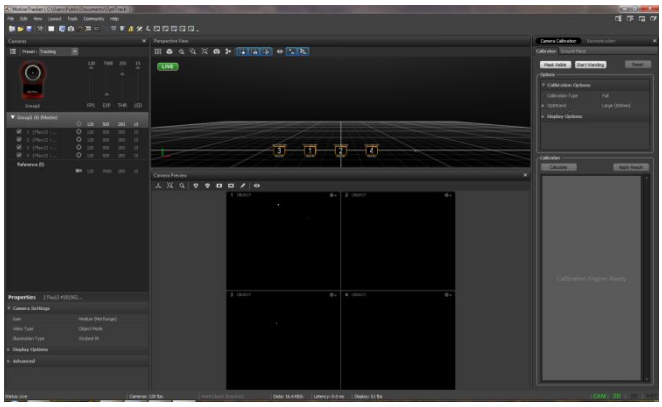
Copy and paste the license file (.dat extension) from the Laboratory Dropbox into this folder.



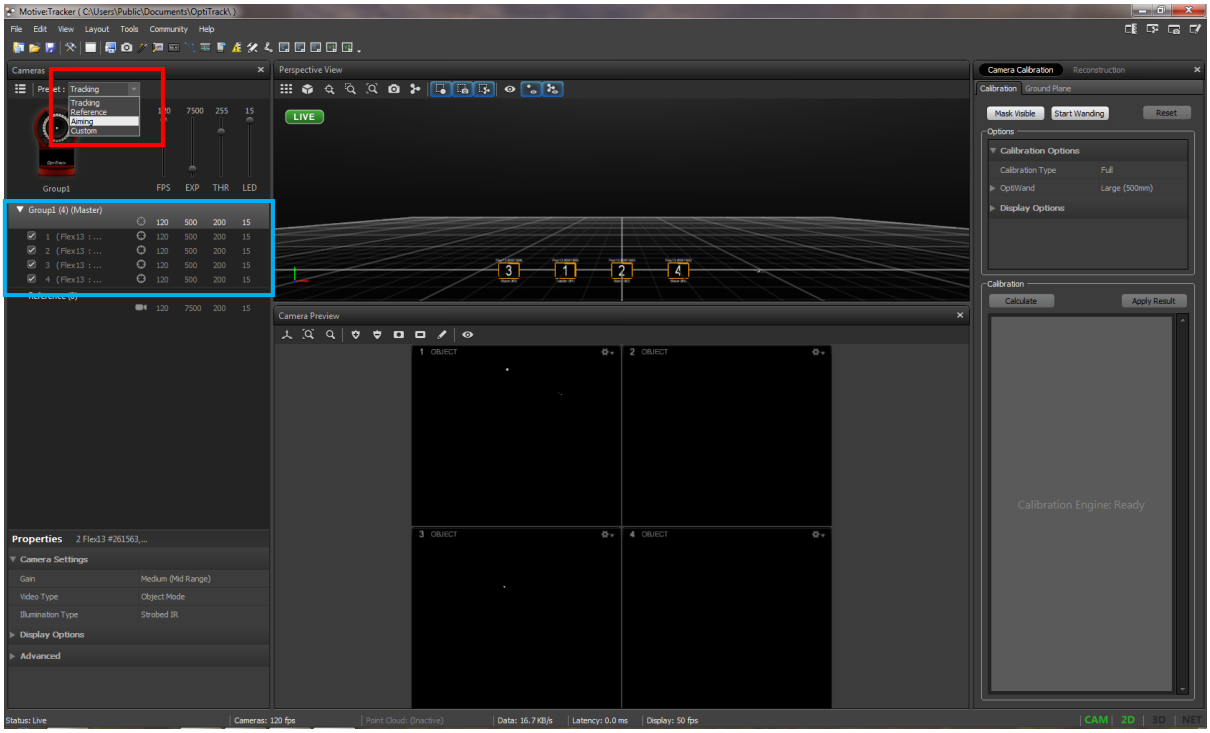
Starting Motive:Tracker

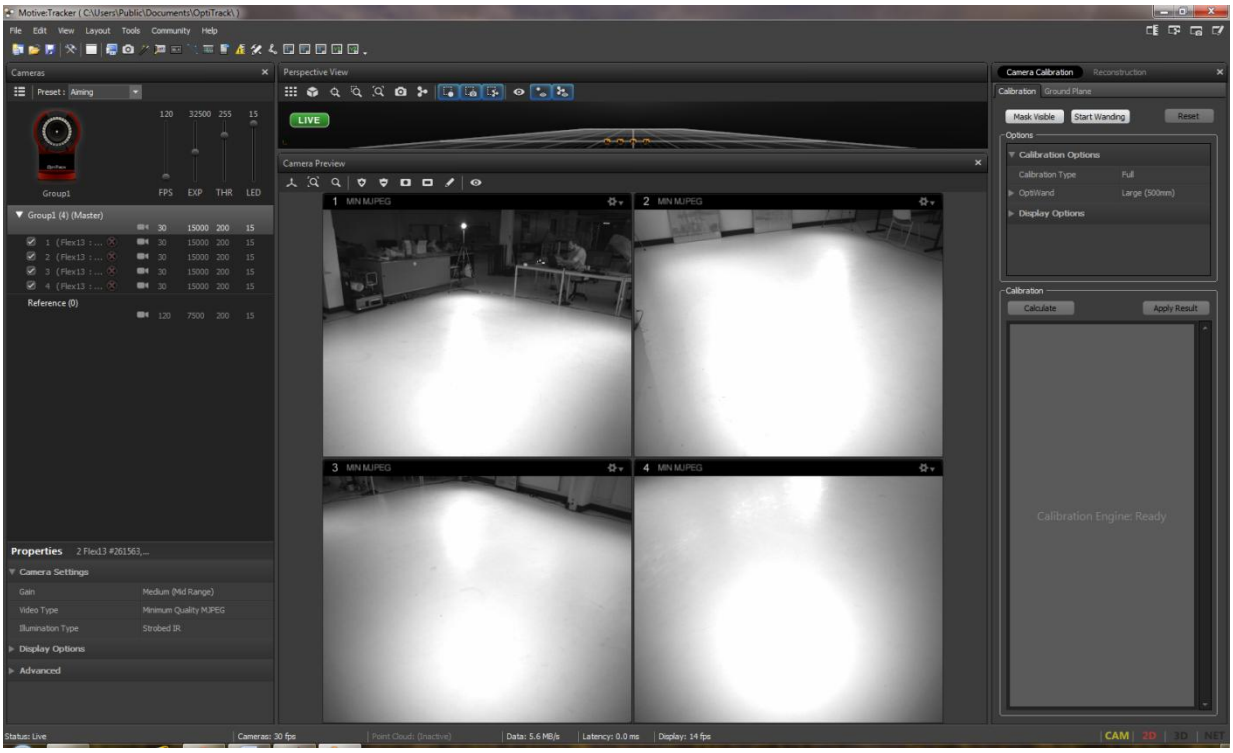
To start the Motive:Tracker software, perform the following steps.

Step 1 – Insert the hardware key that is attached to the envelope (left figure) into the computer USB port and start the program (right figure) by opening the shortcut or clicking retry in the license interface.



Step 2 – Select the camera group containing all cameras (blue) and set the mode (red) to “Aiming”, which helps with adjusting the camera position such that full coverage of the field is achieved.



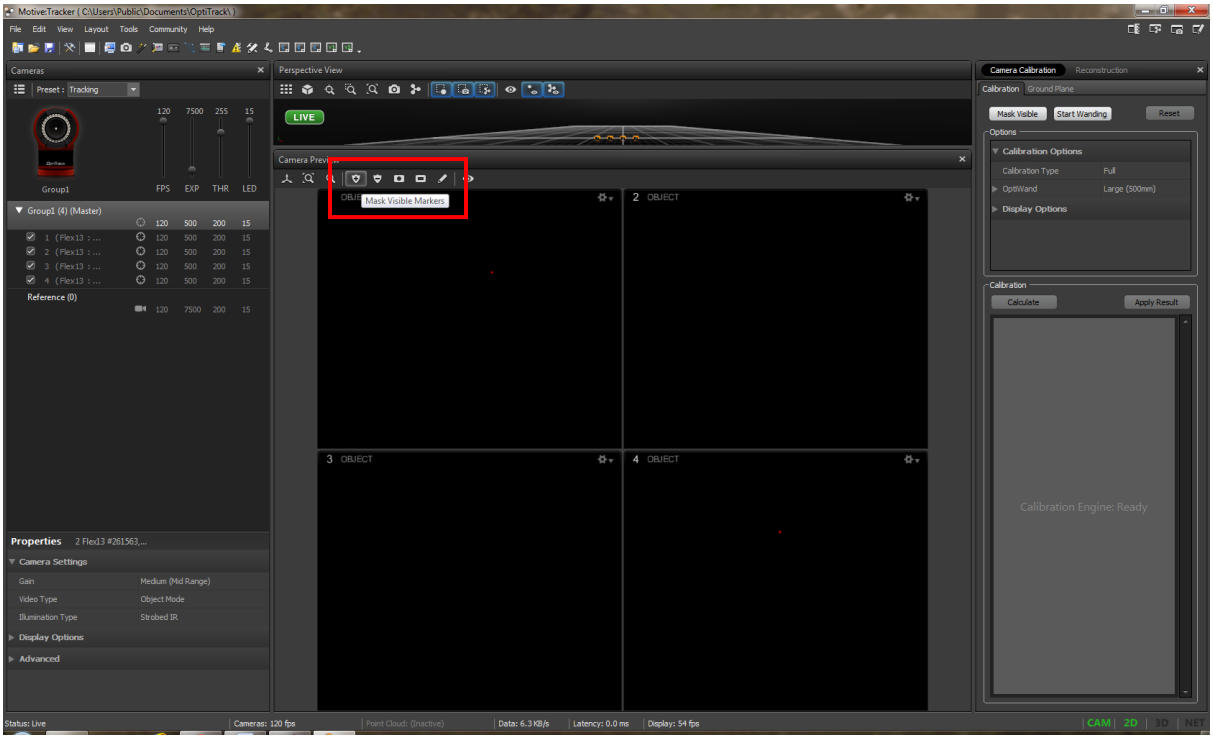


Step 3 – View the computer screen while aiming the cameras. Pressing the aim button (blue in the figure below) on the back of the camera, brings up a larger display. Exit aiming mode for the camera by pressing the button again and move to the next camera until a satisfactory setup is achieved. Now switch back to "Tracking mode" in the Motive:Tracker interface.

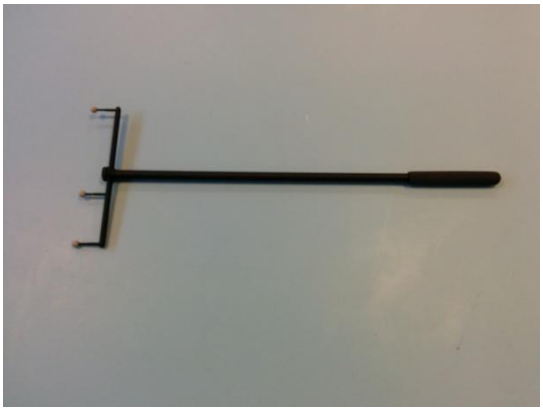
System calibration

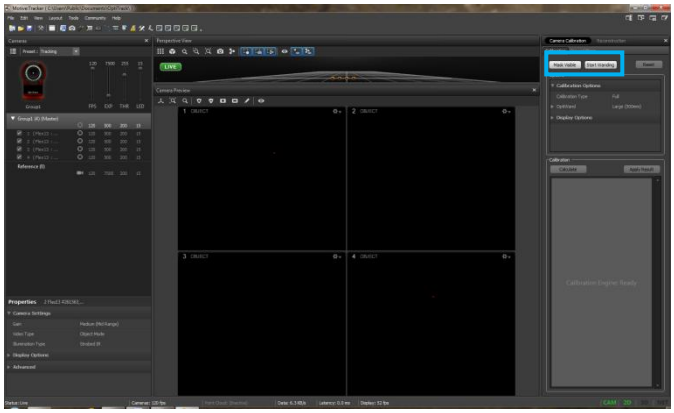
After starting the software and aiming the cameras, the system needs to be calibrated.

Step 1 – Set the cameras to tracking mode and remove all infrared reflecting objects from the subject area. It may happen that some unknown objects still reflect light or that opposing camera's see each other's infrared LED lights. To circumvent this problem, use the "Mark visible markers" option (red) or block them manually.

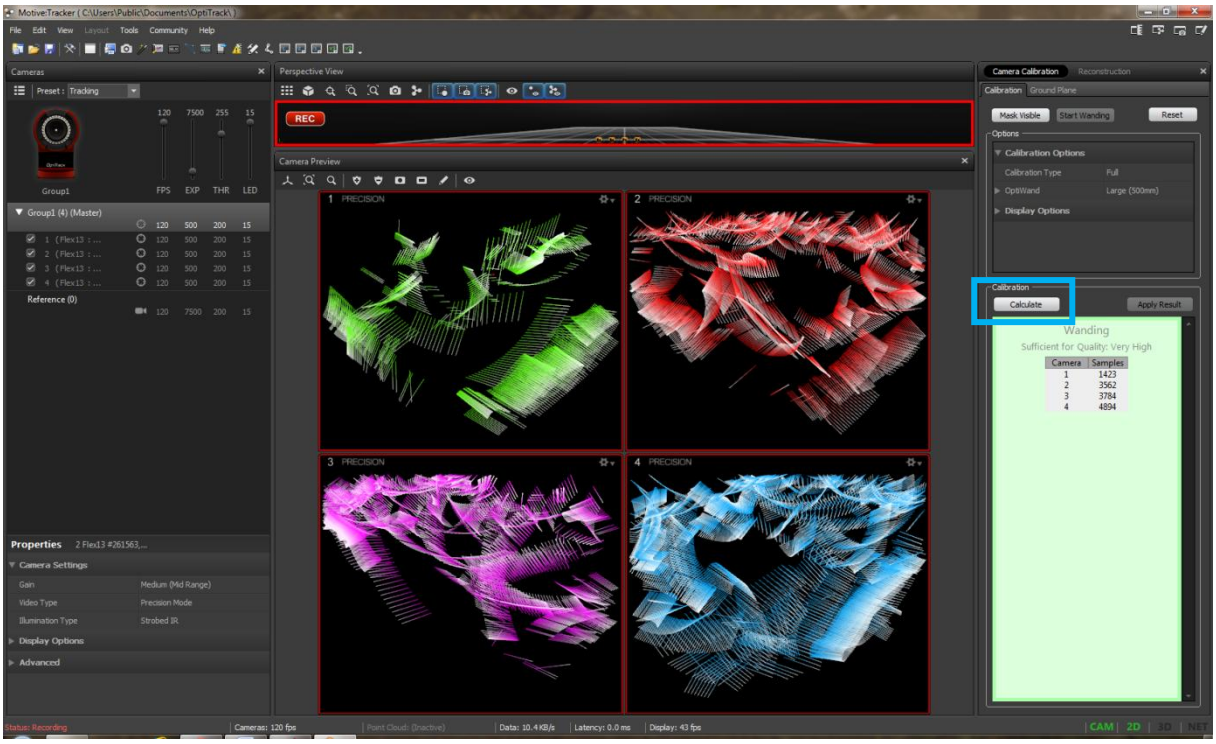


Step 2 – Pick up the "wand" (left) and select the "start wanding" option (blue) in the menu to calibrate the system.

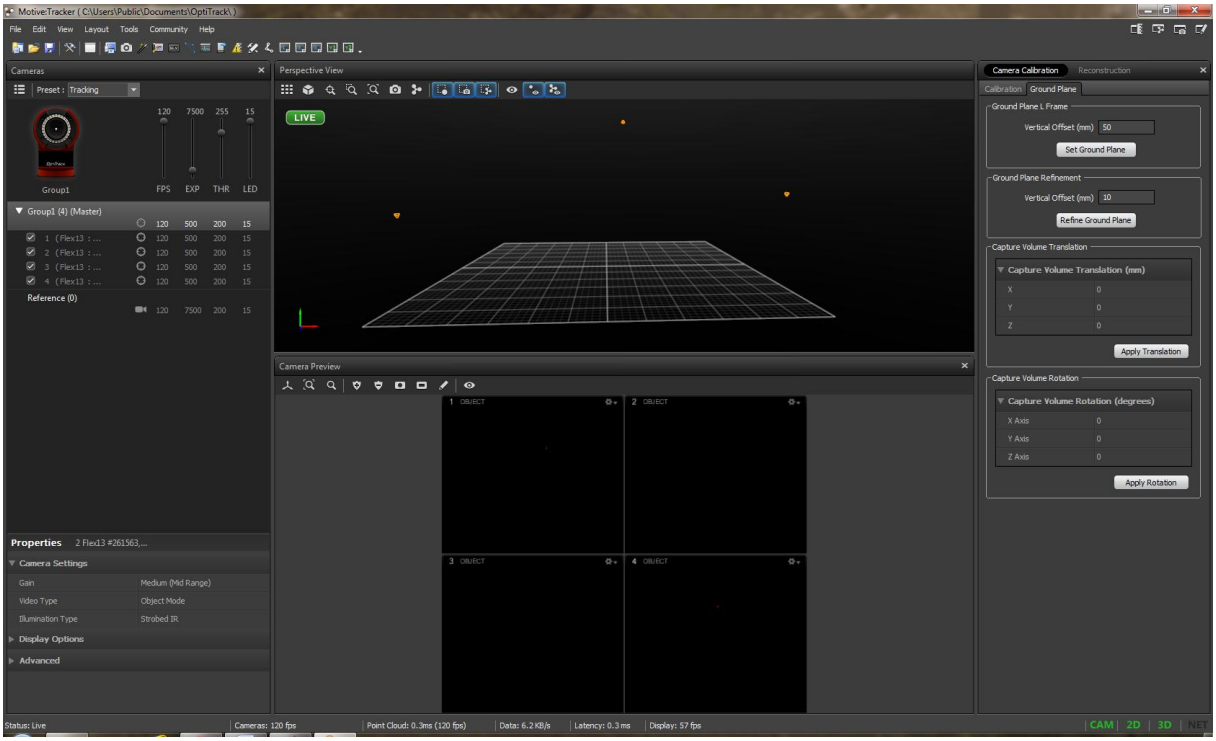




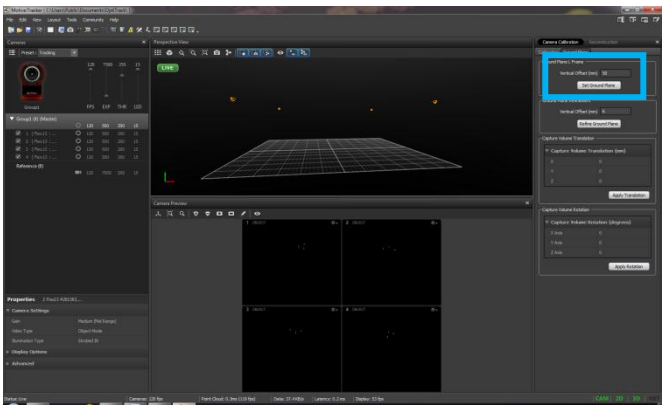
Step 3 – Wave the “wand” in the subject area and make sure the full area is covered for high quality results. When this is achieved, it is displayed in the right menu with a green background and the “calculate” button (blue) can be pressed. A screen is displayed with the results and these can be accepted.



Step 4 – The system is now calibrated and the camera positions are displayed in the 3D view. One thing that remains is the leveling of the ground plane.



Step 5 – To level the ground plane, place the "calibration square" (left figure) that is provided in the subject area and click the "set ground plane" button (blue). Before pressing the button, make sure that the calibration square is level and that the Z is positioned correctly. As the markers on the calibration square are above the ground plane, apply the correct offset to the plane.



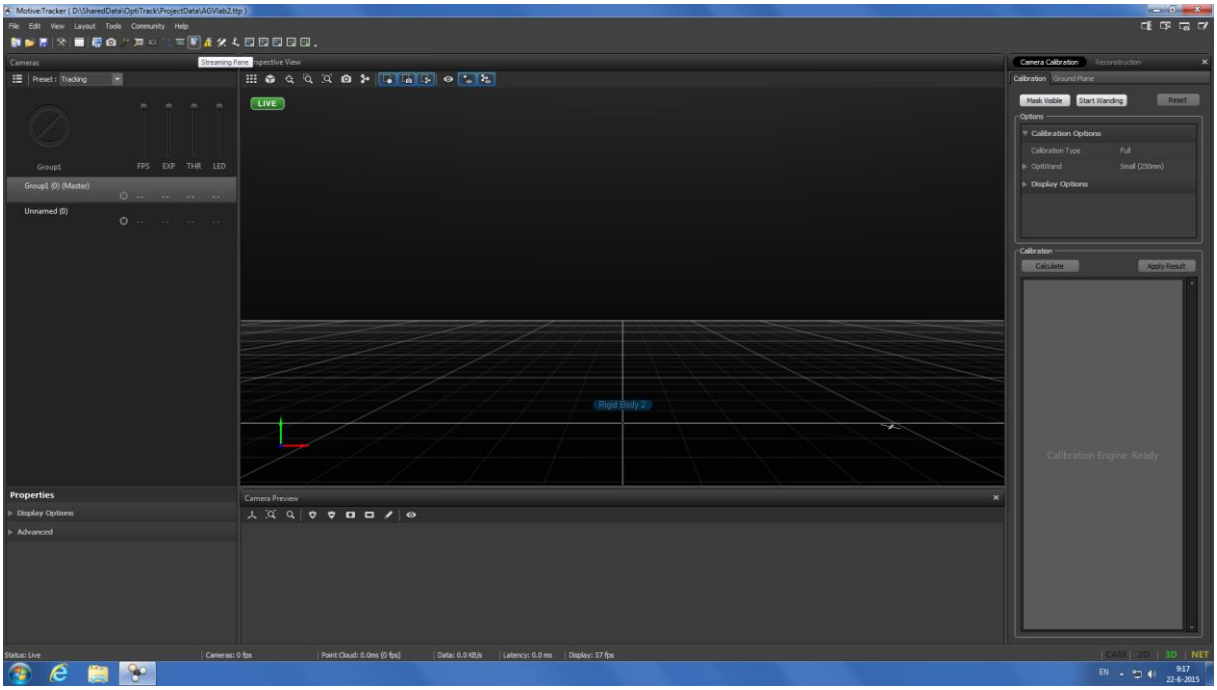
If all steps are performed correctly, the system is now calibrated and ready for use.

D.3 How to connect Matlab to the OptiTrack system

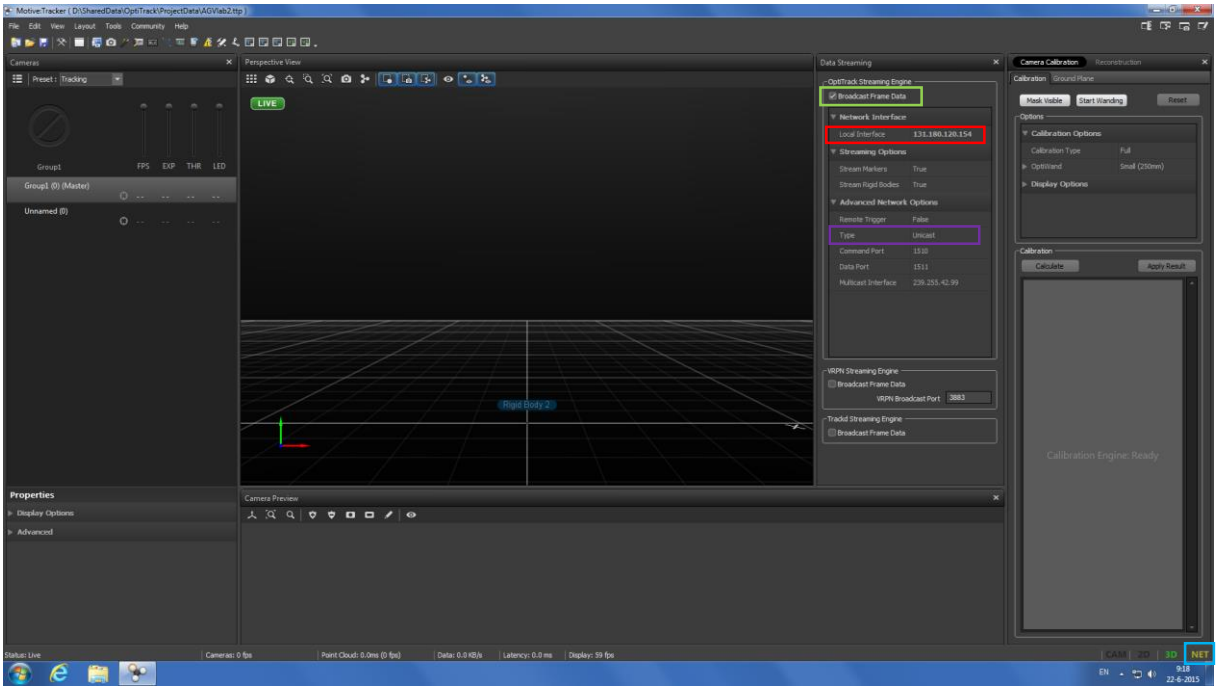
How to connect Matlab with the OptiTrack system

Following the other two instruction on how to set up and calibrate the OptiTrack system, this instruction helps with connecting OptiTrack and Matlab via the eduroam network. The network is part of the TU Delft environment and can be connected to by students and staff with either a cable or wireless connection. OptiTrack requires a computer which is allowed to broadcast on ports 1510 and 1511. Ask the IT department for more information on this. Once the computer is given access to broadcast on eduroam, take the following steps to connect Matlab:

Step 1 – Open OptiTrack's Motive:Tracker software and click on the "Streaming Pane" icon visible below.

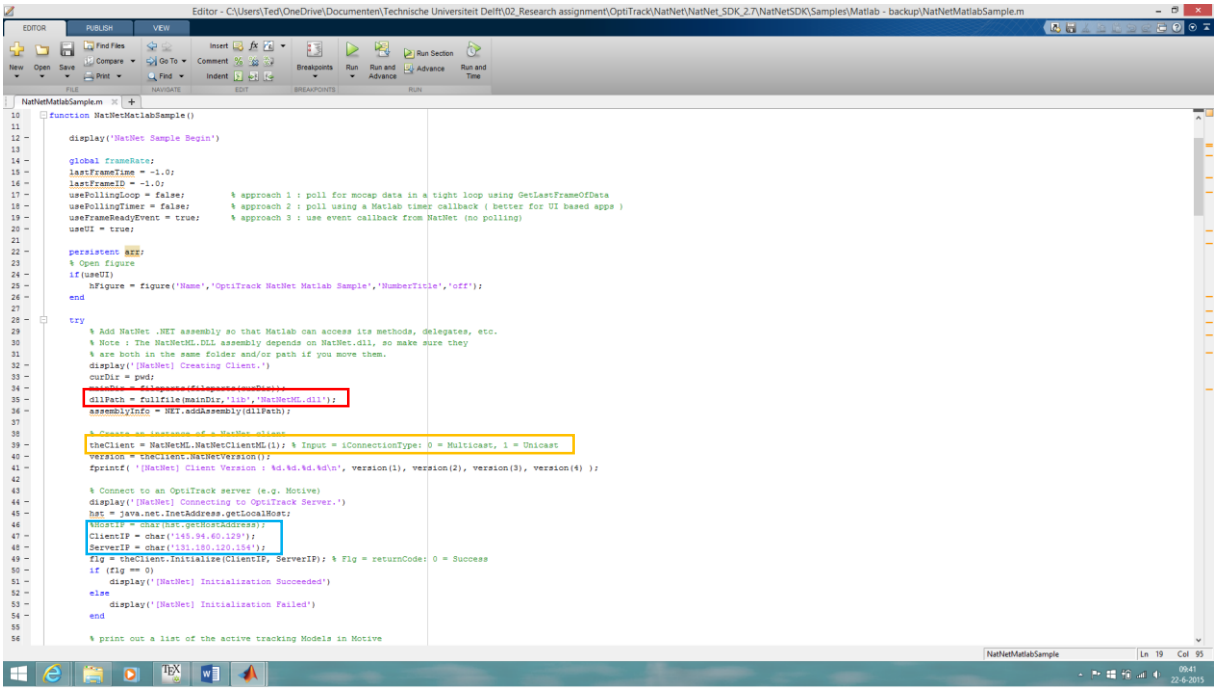


Step 2 – Clicking the button will open the Data streaming window. Make sure to note the "Local Interface" IP-address, note which type of streaming you use (Unicast or Multicast) and check the box "Broadcast frame Data". The "NET" icon in the bottom right will turn orange when the box is checked and turn green once a program has connected with Motive:Tracker.



Step 3 – Now that OptiTrack is set-up, Matlab can connect to it. An example program is provided in the NatNet SDK that can be downloaded from the website. The version that is used is NatNet SDK 2.7 and contains a folder with sample code, of which one is concerned with Matlab. A few things are important to set properly, namely;

- The code uses a library which allows for the connection to OptiTrack. The library is dependent on which version of Windows you are using. It assumes that you are using a 64-bit OS, but when using a 32-bit OS, the marked portion of needs to be replaced by: "dllPath = fullfile(mainDir,'lib','x86','NatNetML.dll'). Make sure that you do not change the hierarchy of the NatNet SDK folder, Matlab then cannot find the library.
- The orange marked portion of the code needs to be adjusted for the type of connection you make. OptiTrack can either make a Multicast and Unicast connection. Experience learned that it does not matter which type is used, but both Matlab and OptiTrack need to be set-up with the same type.
- The final marked portion of code needs to be adjusted according to the OptiTrack server IP-address (ServerIP) and your computers personal IP-address (ClientIP). The server's IP-address is found at the "Local Interface" from the figure above. The client's IP-address can be obtained by opening windows command and typing "ipconfig".



Step 4 – Now that everything is done, you can run the Matlab code. If all goes well, you will see a graph that shows the Euler angles of the object in Motive:Tracker.

Note – This is an instruction for the basic NatNet SDK code and has been written in the case that all information is lost. An altered version of the program, for the purpose of vehicle tracking, is available on the AGV lab Dropbox. Using that code is quite similar. The IP-addresses and connection type can be adjusted in the "Parameters" file. Also, in this code the selection of 32-bit and 64-bit is automated and does not require altering.