Joseph Man-Kit Chung

Formation Control for Multiple Robots

Master thesis based on the examination and study regulations for the Master of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Holzhüter
Second examiner: Prof. Dr. Claussen

Day of delivery November 1st 2006
Joseph Man-Kit Chung

Title of the Master Thesis
Formation Control for Multiple Robots

Keywords
Formation, S-functions, M-files, MATLAB/SIMULINK, Control System, Feedback Control, Integral Control, State-Space Control, Pole Placement, Camera Calibration, Bi-Linear Interpolation, Image Reduction, Position Estimation, Stability, Overlapping Area, Merging Algorithm, Distance to Leading Robot, Steady State, Equilibrium State, Timing, Rest Time, Control-Step Time, Curvature Error, Reference Track

Abstract
In this master thesis, a navigation system for multiple mobile robots has been developed. Two robots have been controlled to follow a static track, which lies in an area observed by two cameras. Image processing, image reduction and camera calibration techniques are applied to obtain the robot states from the image. Controllers were determined from the state base description of the robots. In order to compensate the system disturbances and improve accuracy, different controllers are designed. Timing was improved through image reduction by estimation of robot’s position. The system was developed on the platform MATLAB/SIMULINK. The results were very satisfying with certain limitation of speed of the mobile robots.

Joseph Man-Kit Chung

Thema der Masterarbeit
Formation Regelung für mehrere Roboter

Stichworte
Formation, S-Funktion, M-Datei, MATLAB/SIMULINK, Regelungssystem, Rückkoppelungsregler, Integralregler, Regler im Zustandsraum, Polvorgabe, Kamera-Kalibrierung, Bi-lineare Interpolation, Bildbereichsreduktion, Positionsschätzung, Stabilität, Überlappungsbereich, Algorithmus für Verschmelzung, Distanz zum führenden Roboter, Stetigkeitszustand, Gleichgewichts-Zustand, Timing, Restzeit, Regelschrittdauer, Krümmungsfehler, Referenzstrecke

Kurzfassung
Contents

1. INTRODUCTION ...................................................................................................................... 1
   1.1. PROJECT OVERVIEW ........................................................................................................... 1
   1.2. THESIS OUTLINE ............................................................................................................... 2
   1.3. DEVELOPMENT ENVIRONMENT .......................................................................................... 2

2. SYSTEM OVERVIEW ................................................................................................................. 2
   2.1. ROBOT ............................................................................................................................... 2
       2.1.1. Motors and Position Encoders ....................................................................................... 2
       2.1.2. Robot’s radio communication ....................................................................................... 2
       2.1.3. Robot software interface ............................................................................................. 2
   2.2. CAMERA SYSTEM ............................................................................................................... 2
   2.3. OPERATION FIELD ............................................................................................................. 2
   2.4. CONTROL SOFTWARE ....................................................................................................... 2
       2.4.1. MATLAB as computing platform ................................................................................. 2
       2.4.2. SIMULINK as modelling environment ........................................................................ 2

3. STATE-SPACE CONTROL THEORY .......................................................................................... 2
   3.1. TRANSFER FUNCTION AND SYSTEM STABILITY .............................................................. 2
   3.2. STATE-SPACE DESCRIPTION ........................................................................................... 2
   3.3. SYSTEM STABILITY IN STATE-SPACE ............................................................................ 2
   3.4. FEEDBACK STATE SPACE CONTROL ............................................................................... 2
   3.5. INTEGRAL CONTROL ......................................................................................................... 2
   3.6. OBSERVER CONTROL ........................................................................................................ 2
   3.7. CONTROLLABILITY ............................................................................................................. 2
   3.8. OBSERVABILITY .................................................................................................................. 2

4. STATE SPACE CONTROL MODELS OF THE ROBOTS .............................................................. 2
   4.1. CONTROL MODEL OF THE LEADING ROBOT .................................................................... 2
       4.1.1. Differential equations and state space description ...................................................... 2
       4.1.2. Linearization ................................................................................................................ 2
   4.2. CONTROL MODEL OF THE TRACING ROBOT ................................................................. 2

5. CAMERA CALIBRATION ............................................................................................................. 2
   5.1. CAMERA DISTORTION ....................................................................................................... 2
   5.2. DIRECT CALIBRATION APPROACH .................................................................................. 2
       5.2.1. Bilinear interpolation .................................................................................................... 2
       5.2.2. Procedure of the Direct Calibration approach ............................................................ 2
   5.3. EXTENSION TO TWO CAMERAS ....................................................................................... 2
   5.4. ALIGNMENT OF TWO CAMERAS ...................................................................................... 2
       5.4.1. Geometrical Transformation ....................................................................................... 2
       5.4.2. Calibration of second camera .................................................................................... 2
   5.5. PROJECTION TRANSFORMATION WITH TWO CAMERAS ........................................... 2
   5.6. ACCURACY CHECK OF OVERLAPPING AREA ................................................................... 2

6. IMAGE PROCESSING AND ANALYSING .................................................................................. 2
   6.1. OVERALL PROCEDURE ....................................................................................................... 2
   6.2. REDUCTION OF IMAGE SIZE ........................................................................................... 2
       6.2.1. Methods to estimate position ....................................................................................... 2
       6.2.2. Size of reduced image ................................................................................................. 2
6.3. OBJECT DETECTION .................................................................................................. 2
   6.3.1. Colour conversion ............................................................................................. 2
   6.3.2. Boundary detection .......................................................................................... 2
   6.3.3. Least Squares Fitting ...................................................................................... 2

6.4. MERGING OF OBJECTS ....................................................................................... 2

7. IMPLEMENTATION ISSUES .................................................................................... 2
   7.1. HARDWARE FEASIBILITY ................................................................................... 2
      7.1.1. Multiple cameras .......................................................................................... 2
      7.1.2. Multiple robots ............................................................................................ 2
   7.2. SIMULATION MODELS ....................................................................................... 2
      7.2.1. Model of leading robot ................................................................................... 2
      7.2.2. Model of tracing robot .................................................................................. 2
      7.2.3. Define reference Track and transfer Coordinate ............................................. 2
   7.3. CURVATURE ERROR ........................................................................................... 2
   7.4. SIMULATION RESULT ....................................................................................... 2
      7.4.1. Basic control vs. integral control ...................................................................... 2
      7.4.2. Simulation result of oval track ........................................................................ 2
      7.4.3. Improvement through pole placement ............................................................ 2
      7.4.4. Improvement through prediction .................................................................... 2
   7.5. WIDTH OF OVERLAPPING AREA .......................................................................... 2
   7.6. SATURATION VS. OVERDRIVE ........................................................................ 2
   7.7. REAL-TIME MODELS ........................................................................................ 2
   7.8. TIMING CONSIDERATION .................................................................................. 2
      7.8.1. Timing in real time system .............................................................................. 2
      7.8.2. Timing challenges ........................................................................................ 2
   7.9. ANALYSE CONTROLLER MATRIX ...................................................................... 2

8. TRACK FOLLOWING TESTS .................................................................................... 2
   8.1. LOW SPEED 200 MM/S RESULTS ...................................................................... 2
      8.1.1. Feed back control .......................................................................................... 2
      8.1.2. Integral control ............................................................................................. 2
   8.2. HIGH SPEED RESULTS ...................................................................................... 2
      8.2.1. Feedback control ........................................................................................... 2
      8.2.2. Integral control ............................................................................................ 2

9. CONCLUSION ............................................................................................................ 2

ACKNOWLEDGEMENT ................................................................................................. 2

REFERENCES ............................................................................................................... 2

FIGURE-INDEX ............................................................................................................. 2

APPENDIX A: ALPHABETICAL FUNCTION LIST ........................................................... 2

APPENDIX B: SCRIPT TO COMPUTE CONTROLLER MATRICES ................................. 2

DECLARATION ................................................................................................................. 2
1. Introduction

Mobile robots are compared to wired or fixed robot especially flexible. That is certainly one important reason that the interest of research on mobile robot control has been continuously increased in the past decades. It is an interdisciplinary field, which covers several areas, such as control technology, mechanics, wireless communication, optics, acoustics, image processing, digital signal processing, data computing etc. The progresses have been made in these areas in the last years enable affordable, reasonable-size and stable systems of mobile robot control. In the last decade the most significant improvement has been carried out in the wireless communication and computing power for data processing.

More and more practical applications of mobile robot control have been realized and adopted. Practical mobile robot control in real life can be found e.g. in modern factory, hospital, space-expedition, deep-sea expedition, warfare and even in household. Formation control of multiple robots is expected to be especially promising, due to the opportunity to execute more complex tasks by several coordinated agents.

Several approaches of control have been formulated and implemented. Few examples are fuzzy logic control, procedure control, servo control and state-space control. The control method used in this project is state space control, which has become more and more popular in the last decades.

1.1. Project Overview

The aim of the project is to navigate more than one robot driving along a predefined track. The operational field should be extendable by applying more than one camera. The positions of the robots are determined by cameras installed at the ceiling of the laboratory. A plate with two reflection marks is attached on the top of each robot. The reflection marks are referred as spots in this thesis too. The task of image processing is to detect these reflection marks. The steering of the robots is carried out by a Windows XP Operating System based work station, which sends control commands to the robots via wireless links. The values for steering are determine by a MATLAB/SIMULINK model. Figure 1-1 shows the basic setup of the project. In this project several engineering areas are involved: automatic control, image processing, wireless communication and software engineering.
Introduction

Figure 1-1 Basic setup of the project

The navigation of two robots driving on predefined tracks in an operation field observed by two cameras has been realized. The predefined tracks are of straight-, circular- and oval-form. The first robot is control to drive on the predefined track with a constant forward moving speed, while an optional (limited) offset to the predefined track is possible. The second robot is controlled to follow the first robot, also referred as the leading robot through out the thesis, whereby the second robot should all the time keep a constant distance to the leading robot. In other words, the forward moving speed of the second robot has to be variable and is an object to control too. The second robot is also called as the tracing robot in this paper.

1.2. Thesis outline

Chapter 2 describes the project system overview. The chapter itself is subdivided into four parts: Robot, Operation field, Camera system and Control software.

In chapter 3 the theoretical background of state-space control is introduced and explained. At first Transfer function and System Stability are explained. Then the description of system in State-space is shown. In particular those state-space control methods, which are used or tested in the project, are presented. They are basic control, integral control and observer control. At the end of the chapter the term Controllability and Observability are introduced.

Chapter 4 describes how the theoretical knowledge of chapter 3 is applied to control the specific robots, which are used in this project. This chapter differentiates the control of the leading robot and the control of the tracing robot.

Chapter 5 deals with the topic Camera Calibration. At First distortion through a lens is presented, and then explanation on the direct calibration approach, which is a result of a previous master thesis work in the same department, is given. Then thoughts about alignment of two adjacent cameras are illustrated. At last projection transformation is treated.

In chapter 6 Image Processing and Analysing are treated in detailed. The procedure of image processing of the project is shown. A subchapter is dedicated to Reduction of image
size. Another subchapter is the methods used for Object detection. The problem of Merging of objects is handled too.

In chapter 7, issues related to implementation are addressed. Amongst other things the Simulation models and Real-time models are presented. Another critical object to the project, Timing consideration, is treated there too.

Chapter 8 discussed the overall results of manoeuvring on an oval track with different models using different parameters.

Chapter 9 gives a conclusion and some outlook of the entire project.

1.3. Development Environment

For development of the simulation models and for the simulation itself, any PC that is able to run MATLAB 6.5.1+Service Pack 1 stable and at a decent pace would be sufficient.

When it comes to real implementation and operation, the PC needs to be fast and posses all the interfaces to the peripherals of the system. A fast PC is important since the control task is a real time task, which means each control step has only a limited time to be executed. Increments of field size and number of subjects to control (the robots) lead to tighter timing requirement.

Unlike previous thesis-projects on this subject in the department, timing has become a critical issue. Details about timing issues can be found in chapter 7.

The workstation used for development is as following:

- PC with 3.0G Hz Pentium4 HT processor and 512M DDR memory
- MATLAB 6.5.1 + Service Pack 1.
- Com-port PCI-card
- Firewire PCI-card
2. System Overview

This chapter describes the hardware and software used in the project.

2.1. Robot

The robots used in the project are called AmigoBot, which is a product of the company ActivMedia ROBOTICS. Its technical manual [1] describes: "AmigoBot is a small, 2-wheel, differential drive, intelligent mobile robot. Like its Pioneer siblings, AmigoBot is truly an off-the-shelf, “plug and play” mobile robot, containing all of the basic components for autonomous sensing and navigation in a real-world environment, including battery power, drive motors and wheels, position / speed encoders, sonar range-finding sensors, and integrated accessories, all managed via an onboard microcontroller and mobile-robot server software."

The AmigoBot drive and sensor systems are controlled and processed from a single controller, a Hitachi H8 microprocessor.

![AmigoBot Diagram](image)

**Figure 2-1 different views of AmigoBot**

From the AmigoBot’ system, the components motor, radio modem and application interface are of great importance in our project. Therefore these parts are briefly explained separately in the following subchapters. For more detailed information about the AmigoBot please refer to its technical documents [1], [2], [5] and its website [3].

---

1 This figure is take from [1]
2.1.1. **Motors and Position Encoders**

Each of its two solid 10cm rubber tires, referred as Drive wheel in Figure 2-1, is driven by a reversible 12VDC motor. The DC motors can be control separately. This feature is used to steer the direction of the robot.

Furthermore [1] says:

“AmigoBot’s drive system uses high-speed, high-torque, reversible-DC motors. Each front drive motor includes a high-resolution optical quadrature shaft encoder that provides 9,550 ticks per wheel revolution (approx. 30 ticks per millimeter) for precise position and speed sensing and advanced dead-reckoning. The tires are four inches in diameter and made of soft, but firm rubber for good traction and low compressibility.”

2.1.2. **Robot’s radio communication**

The internal radio modem of AmigoBot enables wireless communication between a workstation with the AmigoBot’s controller, consequently also wireless manoeuvring of it.

The setup of the radio communication between a workstation and an AmigoBot is show in Figure 2-2. The radio modems are described in [1] as following:

“AmigoBot supports an optional radio modem pair (900 MHz) for wireless operation of the robot: One modem gets attached to the robot and the other to your basestation computer. The robot’s modem is mounted on the underside and gets power (5 VDC) and signal (Control serial) via a 9-pin D SUB connector and 2.1mm power plug that come with the robot. The radio’s antenna fits up through the body; it’s top flexible section unscrews from the main body.”

![Figure 2-2 radio communication through radio modems](image)

2.1.3. **Robot software interface**

Instead of communicating directly with the on-board microprocessor H8, it is much more convenient to operate the AmigoBot through its application interface ARIA.

The AmigoBot user’s guide [2] claims:

“The ActivMedia Robotics Interface for Applications (ARIA) is a C++-based open-source development environment that provides a robust client-side interface to a variety of intelligent robotics systems, including your ActivMedia robot’s controller and accessory systems... it neatly handles the lowest-level details of client-server interactions, including serial communications, command and server-information...”

---

2 This figure is taken from [4]
packet processing, cycle timing, and multithreading, as well as a variety of accessory controls, such as motion gyros, among many others.”

Figure 2-3 shows the architecture of ARIA.

![Figure 2-3 ARIA’s architecture](image)

### 2.2. Camera system

Two monochrome CCD (Charge Coupled Device) cameras of the model DMK21F04, from Imaging Source Corporation were used in the project. The cameras are equipped with a ring of four parallel connected LED (Light-Emitting Diode) lights and some resistances. These LED’s are used as light source for the reflexion marks to reflect light back into the camera [6]. Figure 2-4 shows this installation. DMK21F04 possesses a Firewire interface 1394. The FireWire interface offers a multitude connections and power supplies possibilities. The workstation used in the project has a Firewire-PCI card installed itself. In the order to use multiple DMK21F04 at the same time, it is possible either daisy chained them or uses a firewire-hub. The last option is occupied in this project. For further information about the cameras, please referee to its technical documents [1212].

![Figure 2-4 CCD camera on the ceiling](image)

---

3 This picture is taken from [2]
4 This picture is taken from [6]
2.3. Operation field

The operational field of the robots is constrained by the visible and calibrated area of the cameras. The Figure 2-5 shows the dimension of the field covered by a single camera. The outer green rectangle is the total visible field by the camera. The inner green rectangle is the calibrated area. That is the region where the robot is allowed to operate. This area is $2000\text{mm} \times 2800\text{mm}$. The discrepancy between the visible and utilizable area results from the calibration method used. There a patterned sheet is used for calibration. This sheet must lie completely in the visible field; therefore a distance is required as buffer to the most outside border. The dashed lines show the size of a calibration sheet. The calibration method, Direct Calibration approach, is explained in 5.2.

As mentioned in the Project Overview 1.1, the operational area should be enlarged by utilizing more than one camera. In the project two cameras were used to cover the operation field. Figure 2-6 shows the dimension of the complete operation field. It is basically two the single fields placed side by side, and shifted into each other, so that an overlapping area of $800\text{mm}$ is created. That means the two cameras are $2000\text{mm}$ away.
from each other. The crosses in the middle of the rectangles symbolized the position of the cameras over the field. The dimension of the complete field is $2000\text{mm} \times 4800\text{mm}$.

### 2.4. Control software

The entire control system was realized on the software platform MATLAB/SIMULINK from *MathWorks, Inc.*

#### 2.4.1. MATLAB as computing platform

MATLAB was used for to compute of control, process the data.

“...The MATLAB system consists of five main parts:

**Development Environment.** This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.

**The MATLAB Mathematical Function Library.** This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

**The MATLAB Language.** This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

**Graphics.** MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

**The MATLAB Application Program Interface (API).** This is a library that allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.” [7]

Typical uses include:

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
• Data analysis, exploration, and visualization
• Scientific and engineering graphics
• Application development, including graphical user interface building

2.4.2. **SIMULINK as modelling environment**

SIMULINK is an integrated component of MATLAB for the purpose of modelling dynamic systems. The simulation models and control model of the project were realized in SIMULINK and executed from it.

“SIMULINK is a software package that enables you to model, simulate, and analyze dynamic systems, that is, systems whose outputs and states change with time. SIMULINK can be used to explore the behaviour of a wide range of real-world dynamic systems making SIMULINK ideal for control system design, DSP design, communication system design, and other simulation applications.

Simulating a dynamic system is a two-step process with SIMULINK. First, you create a graphical model of the system to be simulated, using the SIMULINK model editor. The model depicts the time-dependent mathematical relationships among the system's inputs, states, and outputs. Then, you use SIMULINK to simulate the behaviour of the system over a specified time span. SIMULINK uses information that you entered into the model to perform the simulation.”[8]
3. **State-space control theory**

This part present the theoretical basic of the control methods used.

### 3.1. Transfer function and System Stability

The transfer function of a LTI-System (Linear Time Invariant-system) in Laplace domain is written as

\[ G(s) = \frac{u(s)}{y(s)} \]

Equation 3-1

\( u(s) \) is input function, \( y(s) \) is output function, and

\[ s = \sigma + j\omega \]

Equation 3-2

The Laplace-Transformation table tells

\[ \frac{1}{s - s_j} \rightarrow e^{s_j t} \cdot \mathcal{F}(f) \]

Equation 3-3

Whereby applying Equation 3-2

\[ e^{s_j t} = e^{(\sigma + j\omega) t} = e^{\sigma t} \cdot e^{j\omega t} \]

Equation 3-4

[10] P.172: The first factor tells about the envelope of the function. The second multiplicand decides about the oscillation, as the Figure 3-2 and Figure 3-3 demonstrate.

![Figure 3-1](image)

**Figure 3-1** a) and b), a): real and imaginary values propagate in time, b): projection on the real and time axes, \( \sigma = -0.05, \omega = 0.5 \)

It can be easily seen that with

\( \sigma < 0 \)

the function decreases to 0, like Figure 3-1 and Figure 3-3:
\[ \lim_{t \to \infty} e^{\sigma t} = 0. \]

\[ \text{Equation 3-5} \]

Oppositely if \( \sigma > 0 \), the result is infinity large, see Figure 3-2.

\[ \text{Figure 3-2 a) and b), } \sigma = 0.05, \omega = 0.5 \]

\[ \text{Figure 3-3 a) and b), } \sigma = -0.05, \omega = 0.0 \]

[11] p. 108: Through partial decomposition, a transfer function can be written as

\[ G(s) = \frac{c_1}{s-s_1} + \frac{c_2}{s-s_2} + \frac{c_3}{s-s_3} + \ldots + \frac{c_{n-1}}{s-s_{n-1}} + \frac{c_n}{s-s_n} \]

\[ \text{Equation 3-6} \]

Using Equation 3-2 the time-domain transfer function, a Heaviside-Entwicklung, will be obtained:

\[ g(t) = c_1 e^{s_1 t} + c_2 e^{s_2 t} + c_3 e^{s_3 t} + \ldots + c_{n-1} e^{s_{n-1} t} + c_n e^{s_n t} \]

\[ \text{Equation 3-7} \]

[12] P.147 says a system is only stable if

\[ \int_{0}^{\infty} |g(t)| \, dt < \infty \]

\[ \text{Equation 3-8} \]

That only happens by Equation 3-5 in terms of Equation 3-7 if and only if the real-parts of all poles (s) are smaller than 0, which means lying on the left hand-side of the pole-zero-configuration:
So far to system-stability: this knowledge will be applied on the state-space description in the following subchapter.

### 3.2. State-space description

In state-space description differential equations on state variables are used to describe a dynamic system. In this method, the differential equations are organized as a set of first order differential equations. The state variables describe the momentarily state of the system. The set of equations can then be formulated in matrix and vector format. Advantages of the state-space design are especially apparent when the system to be controlled has more than one control input or more than one sensed output.

[13] A differential equation of higher order can be represented as a system of (chained) first order differential equation. A comprehensible example is the Newton’s kinetics equation (Double integrator):

\[ m \cdot \ddot{y}(t) = F(t) \]  

Equation 3-10

For the position of a mass \( y \), the (state) variable \( x_1 = y \) is introduced. For the velocity, \( \dot{y} \), the term \( x_2 = \dot{y} \) is brought in. Then the second order differential equation Equation 3-10 can be written as a system of two (chained) differential equations of first order:

\[ \ddot{x}_1(t) = x_2(t) \]  

Equation 3-11

\[ \ddot{x}_2(t) = \frac{1}{m} \cdot u(t) \]  

Equation 3-12

For the system input \( F(t) \), the standard variable \( u(t) \) is chosen. Such an equation-system can be written clearer in form of matrixes and vectors:

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} =
\begin{bmatrix}
0 & 1 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} +
\begin{bmatrix}
0 \\
1/m
\end{bmatrix}
\begin{bmatrix}
0 \\
u
\end{bmatrix}
\]  

Equation 3-13

\[
y =
\begin{bmatrix}
1 & 0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
\]  

Equation 3-14

Or generalized as:

\[ \mathbf{x} = A \mathbf{x} + B \mathbf{u} \]  

Equation 3-15
While $x = (x_1 \ x_2)^T$ is the state-vector of the system, $A$ the system-matrix, $u$ the system-input and $B$ the input matrix. The second equation gives the output $y$. Whereby $C$ is called the output matrix and $D$ is the through-put matrix, which is in this case (not unusual) zero.

3.3. System stability in State-space

In the subchapter 3.1 the stability of system was introduced according to its transfer function and its poles. This knowledge can be applied on state-space description too. It can be evaluated whether a state-space described system is stable or not.

Let’s consider the general form of a system’s state-space description Equation 3-15 and Equation 3-16:

$\dot{x} = Ax + Bu$

$y = Cx + Du$

Remember that this description happens in the time domain and both equations can be transfer into Laplace domain, the same domain where the stability of system has been investigated in subchapter 3.1. The equation set Equation 3-15 and Equation 3-16 can be transfer into Laplace domain by applying the Laplace differential rule [14] P.306 and results in:

$sx(s) = Ax(s) + Bu(s)$

$y(s) = Cx(s) + Du(s)$

According to [13], the relation between state space description and transfer function can be established:

$sx(s) = Ax(s) + Bu(s)$

$\Rightarrow (s \cdot I - A) \cdot x(s) = Bu(s)$

$\Rightarrow x(s) = (s \cdot I - A)^{-1} \cdot Bu(s)$

Whereas $I$ is the unit-matrix with the same dimension as the system matrix $A$. Remember that at matrices-multiplication the sequence of factors is of importance. From the Laplace-transformed Output-equation Equation 3-18 and using Equation 3-19 it follows at the end:

$y(s) = Cx(s) + Du(s) = (C(I \cdot s - A)^{-1}B + D)u(s)$
As a result transfer system can be presented as:

\[ G(s) = \frac{y(s)}{u(s)} = C(I \cdot s - A)^{-1} B + D \]

Equation 3-21

Each state space description is assigned to an ambiguous transfer function. To a transfer function on the other side there exist many (unlimited) state space descriptions. The states can be renumbered or also linear combinations of the states can be built. According to [12] p.47 all poles of the transfer function \( G(s) \) are also Eigen-values of the Matrix \( A \). For a SISO- (Single Input Single Output) system an example from [15] shows this relationship more clearly:

\[
A = \begin{bmatrix}-5 & -6 \\ 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix}1 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix}0 & 1 \end{bmatrix} \quad D = (0)
\]

The inverse of a Matrix is defined by:

\[
[A^{-1}] = \frac{1}{\text{det } A} A_{ji}^{-1}
\]

Equation 3-22

Or for the special case dimension \( n = 2 \):

\[
\begin{bmatrix}a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} = \frac{1}{a_{12}a_{21} - a_{11}a_{22}} \begin{bmatrix}a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}
\]

Equation 3-23

Therefore following Equation 3-23:

\[
(I \cdot s - A)^{-1} = \begin{bmatrix}s + 5 & 6 \\ -1 & s \end{bmatrix}^{-1} = \frac{1}{s(s + 5) + 6} \begin{bmatrix}s & -6 \\ 1 & s + 5 \end{bmatrix}
\]

Further more using Equation 3-21

\[
G(s) = \frac{\begin{bmatrix}0 & 1 \\ 1 & s + 5 \end{bmatrix} \begin{bmatrix}s & -6 \\ 1 & 0 \end{bmatrix}}{s(s + 5) + 6} = \frac{\begin{bmatrix}0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix}s \\ 1 \end{bmatrix}}{s^2 + 5s + 6} = \frac{1}{(s + 3)(s + 2)}
\]

That means the poles of \( G(s) \) are:

\[ s_1 = -2; \quad s_2 = -3 \]

Because of Equation 3-9 this system is stable; all poles lay on the left hand-side of the pole-zero configuration.

It can be seen that only the determinant contributes to the poles:

\[ \text{det}(sI - A) = 0 \]
This statement can be extended to MIMO (Multiple input and Multiple output) system too, since Equation 3-21 is also valid for MIMO system.

The transfer function of a MIMO-system is the permutation of its inputs and output:

$$G_{ij} = \frac{y_i}{u_j} \quad \text{and} \quad G = \begin{pmatrix} G_{i1} & \cdots & G_{in} \\ M & \cdots & M \\ G_{m1} & \cdots & G_{mn} \end{pmatrix} \quad i = N \{1, \ldots, n\} \quad j = N \{1, \ldots, m\}$$

$n$: number of outputs  
$m$: number of inputs

### 3.4. Feedback state space control

In the last subchapter it has been shown, that stability of a state space described system is determined by the determinant of its system matrix $A$. However, instead of simply checking the stability of a system, the main task of control technology is rather to turn an instable system into a stable one, or even more advance, to compensate disturbances to the controlled system.

The state space description Equation 3-15 and Equation 3-16 can be expressed graphically. Figure 3-4 shows a block diagram of it. $k$ is the width of inputs, $n$ is the width of state vector, $l$ is the width of outputs. From these quantities the dimension of the $A$, $B$, $C$ and $D$ matrices can be determined. $A$ is $n \times n$; $B$ is $n \times k$; $C$ is $l \times n$; $D$ is $l \times k$.

![Figure 3-4 Block diagram of a state space described system](image)

To make a system stable, the total system must have all poles lying on the left-hand side of the pole-zero configuration. That can be achieved by adding an additional so called state space controller in a feedback-loop. The state space controller is able to assign any poles to the controlled system. Figure 3-5 shows the configuration of such a state space controller controlled system. In addition to previous block diagram, the Control Matrix $K$ and the Gain Matrix $N$ complement the original system. Matrix $K$ is the additional state
space controller in the feedback loop. At this moment the N-Matrix with its set points \( r \) are not the subject to consider. Or imagine for this moment that \( r \) is a vector of zeros.

The state space controller and the gain matrix make up the input, or in other words the control vector has the following relationship:

\[
u = Nr - Kx
\]

Equation 3-25

Therefore by substituting \( u \) of the system \( \dot{x} = Ax + Bu \) with \( u = Nr - Kx \), the close loop controlled system becomes:

\[
\dot{x} = Ax + BNr - BKx
\]
\[
y = Cx + DNr - DKx
\]

After applying the distributive rule for the matrices calculation, the following equations are obtained:

\[
\dot{x} = (A - BK)x + BNr
\]
Equation 3-26
\[
y = (C - DK)x + DNr
\]
Equation 3-27

By renaming the terms \( (A - BK) \) to \( A_k \), \( BN \) to \( B_k \), \( (C - DK) \) to \( C_k \), \( DN \) to \( D_k \) and \( r \) to \( u_k \) an equivalent structure to Equation 3-15 and Equation 3-16 ensues:

\[
\dot{x} = A_k x + B_k u_k
\]
\[
y = C_k x + D_k u_k
\]

Analogous to Equation 3-24
\[ \det(sI - A) = 0 \] for stability of system.

Equation 3-28 can be derived:

\[ \det(sI - A_R) = 0 \] for stability through control

\[ \det(sI - A + BK) = 0 \]

Equation 3-28

The pole-placement, directly affect the behaviour of controlled system. As mentioned several times, the real-part of the poles have to be negative to achieved stable control. Besides, as said in the subchapter 3.1 with Equation 3-4, the imaginary-part decides the oscillation of the system. Since system designers are normally not interested into an oscillating system, poles without imaginary part are chosen:

\[ \text{Im}\{s\} = 0 \]

Equation 3-29

Remark:

The positions how far left the poles are located on the pole-zero configuration, decide the pace of the system to reach its stable state. Because the more negative the \( \sigma'\)'s in Equation 3-4 are, the faster the equation declines to zero. The real pace desired or realizable of a system, depends often on the real physical constrains of the component in the controlled system, like the highest possible speed of the motor or the sensitivity of the sensors or the step time chosen between each control step.

[13] says that pole placement through Equation 3-28 however is normally not possible through analytical methods. In general, for the practical design of the Control Matrix \( K \) with desired poles, numerical procedures are used, which is part of today’s any mature mathematical software-library. In MATLAB e.g., for higher order or MIMO-systems the procedure place exits:

\[ \text{>> } K = \text{place}(A,B,P) \]

\( A \) and \( B \) are the state space matrices of the system, \( P \) is the vector of the desired poles. The explanation for the algorithm implemented in \( \text{place}( ) \) are renounced in this wok, since that does not belong to the essential part of the project anymore.

However \( \text{place}( ) \) has the disadvantage that the placement for multiple poles, as occurs in binomial-formula, is not allowed. \( \text{Place}( ) \) can be found in the Control System Toolbox of MATLAB.

To derive a formula for the gain matrix \( N \), two pre-condition are assumed, namely that for the steady state, also called equilibrium state:

\[ \delta = 0 \]  

Equation 3-30

\[ y = r \]  

Equation 3-31
$\dot{x}$ is the first derivative of the system state variables and can be interpreted as the alteration of the system. Because steady state is defined as a state, where the system becomes stable, in the steady state $\dot{x} = 0$ is valid. For a well functioning controlled system, the output meant to become the pre-defined reference value $r$, also called set-points, after the system has stabilized itself. Set-points are the values, which the system designer wishes the system output to be.

Taking Equation 3-26 and Equation 3-27 and inserting Equation 3-30 and Equation 3-31 the following is obtained:

\[
0 = (A - BK)x + BNr \rightarrow x = -(A - BK)^{-1} \cdot BNr \\
y = (C - DK)x + DNr = (C - DK)(-A + BK)^{-1}BNr + DNr
\]

\[
y = \left[(C - DK)(BK - A)^{-1} \cdot B + D\right] \cdot N \cdot r
\]

Applying Equation 3-31 the relationship follows:

\[
N = \left[(C - DK)(BK - A)^{-1} \cdot B + D\right]^{-1}
\]

Equation 3-32

An incompleteness left by this kind of basic feedback State Controller, is in the case of disturbances influencing the system. Figure 3-6 shows such a case. Certainly disturbances can occurs at many places of the system and controller, however due to the properties of LTI-system, all the disturbances can be transformed as disturbance $z$ at the input. With a stationary disturbance, a constant offset will retain between output $y$ and set-points $r$:

\[
y - r \neq 0
\]

![Figure 3-6 Controlled system with stationary disturbances](image)

From Figure 3-6 the derivative of the state vector is:

\[
\dot{x} = (A - BK)x + B(Nr + z)
\]
When $r = 0$ and steady state $\Rightarrow \dot{\xi} = 0$, it follows:

\[ 0 = (A - BK)x + Bz \]
\[ x = -(A - BK)^{-1} \cdot Bz \]  
\[ \text{Equation 3-33} \]

According to Figure 3-6 the output $y$ is:

\[ y = (C - DK)x + DNr + Dz \]  
\[ \text{Equation 3-34} \]

Inserting Equation 3-33 and $r=0$ into Equation 3-34, the equation:

\[ y = (C - DK)(-(A - BK)^{-1} \cdot Bz) + Dz \]
\[ y = [(C - DK)(-A + BK)^{-1} + D] \cdot z \]

is obtained. Comparing it with Equation 3-32 the relationship

\[ y = N^{-1} \cdot z \]  
\[ \text{Equation 3-35} \]

follows. From Equation 3-35 it can be seen that with control, disturbance can be reduced (when is $N^{-1}$ smaller than 1), but not totally eliminated.

### 3.5. Integral Control

A further development of state space controller introduced in the previous subchapter is the integral Controller. This kind of Controller is able to completely compensate stationary disturbances to the system by an additional feedback path containing an integrator.

![Figure 3-7 Block diagram of integral control](image-url)
The vector $e$ with its derivative $\dot{e}$ is introduced as a supplementary state vector to the original state vector $x$. $e$ has the same length as the output vector $y$. With:

$$e = y - r$$

Equation 3-36

The state space description for such a system becomes:

$$\begin{pmatrix}
\dot{\mathbf{e}} \\
\mathbf{e}
\end{pmatrix} =
\begin{bmatrix}
A & 0 \\
C & 0
\end{bmatrix}
\begin{pmatrix}
x \\
e
\end{pmatrix} +
\begin{bmatrix}
B \\
D
\end{bmatrix}u$$

Equation 3-37

Whereby $u$ is now:

$$u = -(Kx \ K, e) = -(K K, \begin{pmatrix} x \\ e \end{pmatrix})$$

Equation 3-38

The new state space description can be rewritten into the familiar format:

$$\begin{pmatrix}
\dot{\mathbf{x}} \\
\mathbf{x}
\end{pmatrix} =
\begin{bmatrix}
A^* & B^* \\
0 & 0
\end{bmatrix}
\begin{pmatrix}
\mathbf{x} \\
\mathbf{e}
\end{pmatrix}$$

Equation 3-39

With

$$A^* = \begin{bmatrix}
A & 0 \\
C & 0
\end{bmatrix}, B^* = \begin{bmatrix}
B \\
D
\end{bmatrix}, x^* = \begin{pmatrix} x \\ e \end{pmatrix} \text{ and } K^* = \begin{pmatrix} K \\ K, \end{pmatrix}$$

According to Equation 3-28, Equation 3-39 can be stabilized by the control matrix which fulfills:

$$\det(sI - A^* + B^*K^*) = 0$$

(3.32)

Equation 3-40

The number of desired poles increases by the width of output $y$. The gain matrix $N$ is:

$$N = \left( (C - DK)(BK - A)^{-1} : B + D \right)^{-1}$$

(3.33)

Equation 3-41

,whereby $K$ is the first element of $K^*$.

The integral path delivers during steady state a constant value to compensate part of the disturbance $z$ which the Control Matrix $K$ cannot eliminate.

A trivial example where the integral controller is able to completely compensate, can be given in the following example

During steady state:
\[ y = r \Rightarrow \dot{e} = 0, \text{ means } e \text{ is a constant value which does not alternate.} \]

Without disturbance and integral control Equation 3-25 tells:

\[ u = Nr - Kx \text{ with } x \text{ being at steady state.} \]

That would be still true for the new (integral controlled) system, if it starts from the steady state, and no disturbance occurs. Because the integral path \( \dot{e} \) and \( e \) would be zero

\[ z = e = 0 \]

---

**Figure 3-8 Block diagram of Integral control with disturbance**

---

### 3.6. Observer Control

The derivation for observer control is skipped here. In case of interest please refer to [13]. The state space description of an observer is:

\[
\dot{x} = \begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix} x + \begin{pmatrix} B \\ 0 \end{pmatrix} u
\]

\[ y = \begin{pmatrix} C & 0 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} \]

**Equation 3-42**

**Equation 3-43**

The new system matrices are as follow:
\[
A^* = \begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix}, \quad B^* = \begin{pmatrix} B \\ 0 \end{pmatrix}, \quad C^* = (C & 0), \quad D^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}
\]

Equation 3-44

With

\[
L_o = \begin{pmatrix} L_x \\ L_z \end{pmatrix}
\]

Equation 3-45

The equation of standard observer system can be expressed as follows (see Figure 3-9):

\[
\hat{x} = A^* \hat{x} + B^* u + L_o (y - \hat{y}) = (A^* - L_o C^*) \hat{x} + B^* u + L_o y
\]

Equation 3-46

The state space matrices of observer can be deduced as follows:

\[
A_o = A^* - L_o C^*, B_o = \begin{pmatrix} B^* & L_o \end{pmatrix}
\]

Equation 3-47

\[
L_o \text{ can be calculated by using pole placement.}
\]

\[
L_o = place(A^*, C^*, poles)
\]

Equation 3-48
3.7. Controllability

\[
Q_c = [B, AB, A^2B, K, A^{n-1}B]
\]

Equation 3-49

\(n\) is rank of \(A\).

If the rank \((Q_c)\) is lower than \(n\), the system is uncontrollable.

3.8. Observability

\[
Q_o = \begin{bmatrix}
C \\
CA \\
M \\
CA^{n-1}
\end{bmatrix}
\]

Equation 3-50

\(n\) is rank of \(A\).

If rank \((Q_o)\) is lower than \(n\), the system is unobservable.
4. **State space control models of the robots**

To apply state space control on the AmigoBot robots, which are used in the project, it is necessary to describe the robots mathematically, and that in the manner of a set of differential equations.

As explained in subchapter 2.1, an Amigo robot, or more precisely the software interface of the robot, is able to receive two input quantities for controlling its motion. One is the speed difference of the two (left and right) wheels. The other is the average forward moving speed of the robot. The speed of the right wheel is the average speed adds the speed difference, while the speed of the left wheel is the average speed subtracts the speed difference. By the speed difference between the right and left wheel, motion can be controlled. Namely in the way, as [16] describes: With constant $V_{\text{left}}$ and $V_{\text{right}}$, the Centre of the robot moves with the speed $V = \frac{1}{2}(V_{\text{left}} + V_{\text{right}})$ on a circle that has its centre on the wheel axis, see Figure 4-1.

![Figure 4-1 Rotational motion of robot](image)

The initially aim of the project is to control two robots: one driving on a predefined reference track with a constant pace while the other following the first robot with a constant distance and also driving on the same reference track. Hence the control models of two robots differ slightly.

### 4.1. Control model of the leading robot

Since the leading robot should move forward with a constant pace. The quantity average speed $V$ is a constant value.
4.1.1. Differential equations and state space description

The speeds of the wheels are:

\[ V_{left} = R_{left} \cdot \omega \]
\[ V_{right} = R_{right} \cdot \omega \]

Equation 4-1

The speed of rotation \( \omega = \frac{\Delta V}{\Delta R} \) of the robot is thus given by

\[ \omega = \frac{V_{right} - V_{left}}{R_{right} - R_{left}} \]

Equation 4-2

If the average wheel velocity is represented by \( V \) and we introduce the control variable \( u \), we can express the two velocities as

\[ V_{left} = V - u \]
\[ V_{right} = V + u \]

Equation 4-3

Thus we obtain:

\[ \omega = \frac{2u}{b} \]

Equation 4-4

Where \( b \) is the wheels distance as shown in Figure 4-1. Equation 4-4 is a linear first order differential equation relating the orientation \( \theta \) of the robot to the control variable \( u \).

To get a full description of the robot motion, the translational motion, which is depicted in Figure 4-2, has to be considered. Here the guiding line (reference track) is used as the x-axis as well.

The translational motion of the mobile robot can be expressed as:
\[ q = V \cos \theta \]  
Equation 4-5

\[ q = V \sin \theta \]  
Equation 4-6

Here \( V \) is supposed to be a constant, which was chosen in Equation 4-3. From Figure 4-2 it can be seen that x-axis points to the same direction as the guiding line. The x-position is not an object to control in the case of the leading robot. In other words, only the distance of the robot from the x-axis, the y-position, will be controlled. Therefore the \( q \) equation, Equation 4-6 is not of importance. In the case of Figure 4-2, \( y \) is zero. Now there are only two differential equations: Equation 4-4 and Equation 4-6 of interest. Both equations can be put together if the state vector is chosen as \( x = (x_1 \quad x_2)^T = (\theta \quad y)^T \). The (differential) equations describing the system are:

\[ \dot{x}_1 = \frac{2}{b} u \]  
Equation 4-7

\[ \dot{x}_2 = V \sin x_1 \]  
Equation 4-8

\[ y = x_2 \]  
Equation 4-9

Here \( b \) is a physical property of the robot (distance of wheels) and its value is 220mm. \( V \) is a given constants. This is the state space description of the robot system. It is obvious that this robot system is non-linear because of the existence of \( \sin \). To utilize state space design, linearization of has to be performed.

### 4.1.2. Linearization

This part describes the linearization of the robot’s equations.

The steps of linearization are the following:

- Applying partial differentiation on each state variable and on each input variable.
- Inserting operating points, values during steady state, into the terms of partial differentiates.

\[ \dot{x} = f_n(x, K, x_n, u, K, u_k) \]  
Equation 4-10

\[ \Delta \dot{x} = \left[ \frac{\partial f_n}{\partial x_1} \right]_{x_0} \Delta x_1 + \left[ \frac{\partial f_n}{\partial x_2} \right]_{x_0} \Delta x_2 + K + \left[ \frac{\partial f_n}{\partial u_k} \right]_{x_0} \Delta u_k \]  
Equation 4-11

After linearization the state space description Equation 3-15 and Equation 3-16 turn into:
\[
\Delta \hat{\mathbf{x}} = A \Delta x + B \Delta u \\
\Delta y = C \Delta x + D \Delta u
\]

Equation 4-12

Equation 4-13

The linearized Equation 4-7 becomes

\[
\Delta \hat{\mathbf{x}} = \frac{2}{b} \cdot \Delta u
\]

Equation 4-14

while Equation 4-8 results into:

\[
\Delta \hat{\mathbf{x}} = V \cos x_i |_{x_i=0} \cdot \Delta x_1 = V \cdot \Delta x_1
\]

Equation 4-15

Equation 4-9 turns into

\[
\Delta y = \Delta x_2
\]

Equation 4-16

The state vector \( \Delta x \) is now \( \begin{pmatrix} \theta & e \end{pmatrix}^T \), with \( e \) is the distance in direction of y-axis between the robot the reference track.

Therefore the system matrices results into:

\[
A = \begin{pmatrix} 0 & 0 \\ V & 0 \end{pmatrix}, \quad B = \begin{pmatrix} \frac{2}{b} \\ 0 \end{pmatrix}, \quad C = (0 \quad 1) \quad \text{and} \quad D = (0).
\]

### 4.2. Control model of the tracing robot

The second robot (the tracing robot) has to keep a constant distance to the leading robot. Assuming in the normal case that the tracing does not start from a position, where it has the desired distance to the leading robot, then the quantity to control the robot forward motion need to be variable and must not be constant.

From another point of view, there are one more attribute to control, beside the angle and vertical distance to the reference track, namely the distance to the leading robot. In other words the state vector of the tracing needs to be extended by one state variable.

In term of translational motion with the x-axis or a parallel straight line to it as reference track (as described in previous subchapter), the extra state variable is the x-directional distance to the leading robot. Equation 4-5 formulizes the speed of x-directional motion.

Subtracting Equation 4-5 of the second robot from Equation 4-5 of the leading robot result into the differential of the new state variable (distance to the leading robot):
Due to the sum-rule for differentiation, it is allowed to relate Equation 4-17 to the distance of the robots, its’ integral. $u_{22}$ is the second input to the system, namely the average speed/forward moving speed.

The differential equation for the angle $\theta_2$ (orientation of the robot) is the same like for the first robot:

$$\dot{\theta}_2 = \frac{2}{b} u_{21}$$

Equation 4-18

The differential equation for the $y$-coordinate of the robots position Equation 4-6 remains nearly the same for the second robot, except that the speed average speed is no longer the constant $V$, but the variable input $u_{22}$.

$$\dot{x}_2 = u_{22} \cdot \sin \theta_2$$

Equation 4-19

After linearized Equation 4-18, Equation 4-19 and Equation 4-17, the following set of equations is obtained (the deltas are dropped for readability):

$$\dot{\theta}_2 = \frac{2}{b} u_{21}$$

Equation 4-20

The operating point for $u_{22}$ is $V$, because after initial transient, the second robot is supposed have the same speed as the leading robot:

$$\dot{\theta}_2 = u_{220} \cdot \cos \theta_2 \bigg|_{\theta_2 = 0} \theta_2 = V \cdot 1 \cdot \theta_2$$

Equation 4-21

For $\dot{x}_2$, Equation 4-17, the first term is neither a function of the second robot’s state variable nor its input. When the first robot has reached the steady state, the angle error $\theta_1$ will become zero; therefore when linearizing Equation 4-17, the first term can be regarded as a constant (, which has only a shifting effect on the equation). By applying Equation 4-11 on Equation 4-17, the following linearized equation is obtained:

$$\dot{x}_2 = -\cos \theta_2 \bigg|_{\theta_2 = 0} u_{22} + \sin \theta_2 u_{22} \bigg|_{u_{22} = V} \cdot \theta$$

$$\dot{x}_2 = -u_{22}$$

Equation 4-22

The new state vector is $x_2 = (\theta_2 \quad e_2 \quad d_2)^T$. With Equation 4-20, Equation 4-21 and Equation 4-22 the system matrices for the second robots are:
State space control models of the robots

\[
A = \begin{pmatrix}
0 & 0 & 0 \\
V & 0 & 0 \\
0 & 0 & 0
\end{pmatrix},
B = \begin{pmatrix}
\frac{2}{v} & 0 \\
0 & 0 \\
0 & -1
\end{pmatrix},
C = \begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\text{and } D = \begin{pmatrix}
0 & 0 \\
0 & 0
\end{pmatrix}
\]

Equation 4-23
5. Camera Calibration

In the optic it is well known that the geometry of the lens does distort the images captured with it. In the project the images captured by the two cameras are delivered by the camera software-driver as pixel images. The task of camera calibration is to setup a method to compensate this distortion, so that the pixel images are converted into geometric images (in millimetre) without or with less distortion effect of the camera.

The images converted into millimetre-format need to be accurate enough, so that the determined positions of the robots are good enough for the control task. Even though this is a rather relative statement, one reasonable measure of accuracy can be the definition of sub-pixel-accuracy[19].

The camera calibration method applied in this project has been overtaken from [6], also a work of the same department, and has been extended to the usage of two cameras. Their work and the references used in their work can be referred for more detailed information about camera model and camera calibration.

5.1. Camera distortion

This subchapter briefly presents the effect the optical phenomena of lens distortion. For further detailed information the sources used here can be referred.

“In this section, a camera model [18] will be described. This model defines four intrinsic parameters which describe the physical properties of the camera,

- Focal length $f_c$ (distance along optical axis from lens to the focal point).
- Principal point $c_c$ (the point which optical axis pierces the image plane).
- Skew coefficient $\alpha_c$ (define the angle between the x and y pixel axes).
- Distortions coefficients $k_c$ (radial and tangential distortion coefficients).

and two extrinsic parameters which describe the camera position in 3D coordinates.

- Rotational coefficient $R_c$.
- Translation coefficient $T_c$.

There are several ways to estimate the camera parameters, but none of them is easy due to the complexity of the distortion models.” [6]
Figure 5-1 visualizes the effect of a Complete Distortion Model of camera. The blue lines are the vectors of distortion of each point on the grid-field. That means the points of the grid appear at their real position plus the distortion vector. It can be seen that in the centre the distortion is smaller, while the further away from the centre, the larger is the distortion. The distortion is basically equally distributed on a circle with a certain radius and with the same centre as the captured image.

Two rectangles have been placed on the experiment field by using sticky tapes. In the image taking by a CCD camera the rectangles are distorted as shown in Figure 5-2; the straight line of the rectangles appear as curved lines in the image. Especially the corners of the rectangle are most distorted, since they are the furthest away from the middle of the image. In next section, a method based on interpolation will be applied to solve the distortion problem.
5.2. Direct Calibration approach

The direct Calibration approach used here, is taken over from the [6]’s project. It was shown there that the accuracy between the position real and the calibrated position is very satisfying, namely always below 4 mm and with a standard deviation of 1.3823 mm in X-direction and 1.6618 mm in Y-direction. With one pixel correspond to 4.75 mm, the accuracy of the calibration is in the sub-pixel range [19].

The basic idea of Direct Calibration approach is based on Bilinear interpolation. A mapping table from image to real world (or from pixel to mm) is created for the operation field. Then any point from the image can be mapped into mm by applying bilinear interpolation.

5.2.1. Bilinear interpolation

[6] Bilinear interpolation is based on linear interpolation; it is an extension of it in 2D space. The basic idea is to perform the linear interpolation in one direction, e.g. the X-direction, first, then in Y-direction.

Assumed that we search for the values of the unknown function \( f(x,y) \) at the point \( R = (x, y) \). When we know the four values of \( f \): \( P_{11} = (x_1, y_1) \), \( P_{12} = (x_1, y_2) \), \( P_{21} = (x_2, y_1) \), \( P_{22} = (x_2, y_2) \), then the points \( Q_1 = (x, y_1) \), \( Q_2 = (x, y_2) \) after interpolation in X-direction. See Figure 5-3 bilinear interpolation.

![Figure 5-3 bilinear interpolation](image)

The interpolation in X-direction yields:

\[
f(Q_1) = \frac{x-x_2}{x_1-x_2} f(P_{11}) + \frac{x-x_1}{x_2-x_1} f(P_{21})
\]

Equation 5-1

\[
f(Q_2) = \frac{x-x_2}{x_1-x_2} f(P_{12}) + \frac{x-x_1}{x_2-x_1} f(P_{22})
\]

\(^5\) This figure is taken from [6]
Then by processing interpolation in Y-direction results the following:

\[
 f(R) \approx \frac{y - y_2}{y_1 - y_2} f(Q_1) + \frac{y - y_1}{y_2 - y_1} f(Q_2)
\]

Equation 5-2

Through insertion of Equation 5-1 and Equation 5-2 into Equation 5-3 we obtain:

\[
 f(x, y) \approx f(P_{11}) \frac{(x - x_2)(y - y_2)}{(x_1 - x_2)(y_1 - y_2)} - f(P_{21}) \frac{(x - x_1)(y - y_2)}{(x_1 - x_2)(y_1 - y_2)}
 - f(P_{12}) \frac{(x - x_2)(y - y_1)}{(x_1 - x_2)(y_1 - y_2)} + f(P_{22}) \frac{(x - x_1)(y - y_1)}{(x_1 - x_2)(y_1 - y_2)}
\]

Equation 5-3

The result of bilinear interpolation is independent of the order of interpolation. If the interpolation is first performed in Y-direction and then performed in X-direction, the results will be the same.

5.2.2. Procedure of the Direct Calibration approach

First 165 equal spaced spots are meant to place on the operation field. These 165 spots form a grid of 11x15. In the practice such a sheet of paper as big as the operation field (of one camera) was not available. Therefore the grid is divided into four parts. After taking images of the grid, a boundary detection algorithm (explained in chapter XXX) and a centre fitting algorithm (explained in chapter 6.3) are employed to get the position of the spots in the distorted image. Since a grid of 11x15 is not fine for the relative large operation field (2000mm x 2800mm), a finer grid is produced by bilinear interpolation itself.

The MATLAB function gridata comes into play here. The usage in our case is:

\[
 \text{>>Xg= gridata(XYi(:,1),XYi(:,2),XYr(:,1),X,Y);} \\
\text{XYi(:,1) are the x-values in pixel of the 11x15 spots, XYi(:,2) are the y-values in pixel. } \\
\text{XYr(:,1) are the x-values in mm of the 165 spots. In other words these are the x-coordinate of the spots in the real world. X and Y is the finer grid in pixel with an equidistance of ten pixels. } Gridata( ) \text{ applies bilinear interpolation to create the output Xg, which is a set of X’s in mm with the fineness preset by the finer pixel grid X,Y. Analogous the set Yg is created by:} \\
\text{>>Yg= gridata(XYi(:,1),XYi(:,2),XYr(:,2),X,Y);} \\
\text{The datasets X, Y, Xg and Yg needs to be stored and loaded to the workspace of the active controller. When the controller model is running, the real position of the robots (or better the spots on the them) can be obtained by converting the pixel values into mm values by the MATLAB method “interp2”:}
\]
Camera Calibration

$$\text{>> p\_mm(:,1)=interp2(X, Y, Xg, p\_pix(:,1), p\_pix(:,2));}$$
$$\text{>> p\_mm(:,2)=interp2(X, Y, Yg, p\_pix(:,1), p\_pix(:,2));}$$

$(X,Y) \rightarrow Xg$ provides the mapping table between the pixel and mm domain. $p\_pix(:,1)$ and $p\_pix(:,2)$ are the $x$ and $y$ values in pixel of the spots on the robot. The function $\text{interp2}$ uses bilinear interpolation to interpolate the according $x$ or $y$ mm-values of $p\_pix(:,1)$ and $p\_pix(:,2)$. The result is corrected from the lens distortion, because the mapping table $(X,Y) \rightarrow Xg$ is a mapping from the distorted lens (pixel) domain to the correct real world (mm) domain.

5.3. Extension to two cameras

The second camera is supposed to be horizontal exact aligned, but vertically shifted against the first camera, as shown in chapter 2.3, see Figure 2-6. The calibration of it can be simply occurred by applying the same Direct Calibration Approach, as explained in 5.2, on its field of sight, just with an offset on the $XYr$-grid.

The offset takes the $x$-value of the left most and lowest point of the operation area of the second camera. In the project the real offset value is 2400mm. Further information about the choice of amount to shift and of the overlapping area size can be found in chapter 7.5.

5.4. Alignment of two cameras

Thoughts have been made about the alignment of two adjacent cameras. Certainly the simplest case of two cameras alignment is either a parallel shift of the second camera in horizontal direction or in vertical direction, as described in subchapter 5.3. However more complicate scenarios can be thought of, namely a combination of horizontal and vertical shifts, supplementary a plane rotation. Figure 5-4 demonstrates a possible situation. Such a camera-placement might be desired e.g. for having an operation field with the form of a curved corridor.

![Figure 5-4 Shift and rotation of two cameras field of sign](image-url)
5.4.1. Geometrical Transformation

Translation matrix for 2 dimensions looks like:

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
T_x & T_y & 1
\end{pmatrix}
\]

By multiplying with the translation matrix, a vector is shifted in x-direction by \( T_x \) and in y-direction by \( T_y \).

Rotation matrix 2 dimensions:

\[
\begin{pmatrix}
\cos \theta & \sin \theta & 0 \\
-sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

By multiplying with the rotational matrix, a vector is rotated by the \( \theta \).

If combine Equation 5-5 and Equation 5-6 the matrix follows:

\[
\begin{pmatrix}
\cos \theta & \sin \theta & 0 \\
-sin \theta & \cos \theta & 0 \\
\cos \theta \cdot T_x - \sin \theta \cdot T_y & \sin \theta \cdot T_x + \cos \theta \cdot T_y & 1
\end{pmatrix}
\]

5.4.2. Calibration of second camera

A way to calibration the cameras, in particular the seconds one, with consideration of the shifts and rotation is requested. One possible method to calibrate the second camera is described as follow:

- Choose the values (in mm) of horizontal shift, vertical shift and (in degree) of rotation
- Set the rotated and shifted coordinate-axes (for the second field) in the position chosen as before
- Mount the second camera on the ceiling at a position which is as precise as possible shifted and rotated by the previously chosen values to the first camera
- Place the calibration sheet on according to the rotated and shifted coordinate-axes
- Apply the Direct Calibration Approach described in the subchapter 5.2.2, with one extra step:
  Before execute the function
  
  \[
  \text{gridata}(XYi(:,1),XYi(:,2),XYr(:,1),X,Y)
  \]
  , apply the geometric transformation matrix, Equation 5-7, on \( XYr(:,1) \).
5.5. **Projection Transformation with two cameras**

The plates with the reflexion marks are mounted on the top of the robots. Due to the height of the robots, the positions seen by the cameras differ from the robot’s real position. According to Figure 5-5 $P_c$ is the position seen by the camera, $P_1$ is the spot on the robot, $P_1'$ is the real position of the robot. Therefore a projection from the 3D-world coordinate to the 2D-image plane is need.

![Figure 5-5 Projection Transform due to robot’s height](image)

Equation 5-8

$$P_1'(x) = O(x) + \frac{H-h}{H}(P_c(x) - O(x)),$$

$$P_1'(y) = O(y) + \frac{H-h}{H}(P_c(y) - O(y)),$$

$H$ is the distance between floor and camera. $h$ is the height of the robot. $O(x)$ and $O(y)$ are the $x$-value or rather $y$-value of the plumb line from the camera. In the project $O(y)$ has been set to 1000mm for both cameras, $O(x)$ is 1400mm for the first camera, 3400mm for the second camera.

5.6. **Accuracy check of overlapping area**

---

6 This figure is taken from [6]
After the realization of the project, thoughts have been spent for further improvement. Therefore an accuracy check on the overlapping area has been done to see how far the camera calibration in the overlapping area is away from the ideal situation.

In the ideal case spots in the overlapping area should be independent from which camera they are taken. The pixel-to-mm mapping should deliver for both cameras the same result for a physical spot (in the overlapping area of the operation area).

The test was carried out in the following way:

- Spots were placed in the over crossing area, each camera took a snap shot.
- mm-values of the spots (from both images) are determined by pixel-to-mm mapping, as in the real execution.
- Values (of the physical identical spots) from both images are compared and the differences are determined.

In the ideal case there should be no differences or only little differences. But the analysis will show that it is not the case and there is still room for improvement. Possible sources of fault will be look for and solutions are discussed.

Due to the relatively big size of the field, the spots are placed as two groups: first group in the upper area, second group in the lower area. The Figure 5-6 a) & b) shows the images as described.
taken from the left and right camera, with spots placing in the upper area. The same with Figure 5-7, but spots are placed in the lower area.

The spots are detected and converted into mm-values. The values are shown in Figure 5-8 and Figure 5-9. Whereby the green spots are taken from the left camera; the blue spots are taken from the right camera.

From both figures it can be seen that there is a clear discrepancy between the spots of both cameras. Furthermore the discrepancy in the x-direction is far bigger than in the y-direction. Actually the alignment in y-direction is quite close. Since in the real time test, see Chapter 8, the track driven over the crossing area are horizontal lines, this imperfection of the camera-calibration was not easy to discover by simple observing (by eyes). This has only bigger effect on the tracing robot’s acceleration, but less effect on the robots following the straight line.
The tables 5-1 and 5-2 show the differences of the spots in x-direction and y-direction. From these values the mean values and standard derivations of the difference is computed:

- mean of x_diff: \( 53.1378 \text{ mm} \)
- mean of y_diff: \( 23.3681 \text{ mm} \)
- standard derivations of x_diff: \( 14.0203 \text{ mm}^2 \)
- standard derivations of y_diff: \( 16.0275 \text{ mm}^2 \)

The inaccuracy in x-direction is more than twice as in the y-direction. In the relative big field, an inaccuracy of about 23mm, in case of y-direction, can still be regarded as acceptable, even it is not nice. But the inaccuracy in x-direction, which is more than 50 mm, is definitely out of desirable range. Sources of the imperfection should be identified and solution need to be provided.

The one source of error is the separation of the fields during calibration. During the calibration, a sheet of spot-grid, which is one quarter of the size of one camera observation-field, is used. It is namely placed four times after each other to cover each quarter of the field. For both camera-calibrations the spot-grid was relocated 8 times. Due to human inaccuracy, every relocation means introduction of certain error (because absolute accurate alignment of the sheet cannot be guaranteed). To avoid this source of
error a big sheet of spot-grid should be used. Instead of applying a small sheet several times sequentially, the big sheet of spot-grid should cover the entire operation field (of both cameras) at once.
6. Image Processing and Analysing

The robot’s states are its position and its orientation. On each robot there are two reflection marks placed. The bigger spot is at the centre of the robot. Besides, these marks are in line, so that the orientation of the robot can be determined too. The cameras mounted on the ceiling capture images periodically with each time-step. The captured images need to be processed and analysed in order to get the desired information: position and orientation of the robots. These are the topics treated in this chapter.

6.1. Overall procedure

The flowchart Figure 6-1 shows the simplified overall procedure for the image processing and analysing. The whole procedure is implemented in the SIMULINK s-function SCameraFunc3. The part of the flowchart surrounded by the green contour is the logical sufficient part of the procedure. Logical sufficient means it logically delivers the correct output: the position and orientation of the robots. However, in the practical operation it turns out that this initial flow cannot safely fulfill the timing requirement of real-time control system. (More about the difficulties related to timing is written in 7.8.) Therefore methods to speed up the image processing and analysing are needed. These are realized as an add-on to the initial flow, outside the green surrounded part.

The green surrounded part of Figure 6-1: First spots are search in the whole area, means the two full size images acquired by the two cameras. To find the spots, the images are converted into black-white-images, and then the bright (white) areas are identified through boundary detection. The centres of the bright areas are derived by a circle-fitting algorithm. After having obtained the spots, the spots are divided into two groups according from which camera they are captured. This separation into groups is needed because later when converting the pixel-values into mm-values, different set of calibration-data (see chapter 5) are used for pixel-values from different camera. Having obtained the spots positions in pixel-value, they need to be converted into mm-value. In case more than 4 spots are found, it means one or more of the spots lie in the overlapping area of the cameras and are captured twice. These double taken spots need to be merged into one spots. Then the spots are sorted by their sizes, because the bigger two spots are attached on the leading robot, and the two smaller spots are attached on the tracing robot. At last but not at least the robots \((x, y)\) positions and orientation are returned.

The part outside of the green contour: At the end of each run, the positions of the robots are stored in global variables. In case these global variables are not empty (that is the case after the first successful run), they are used to obtained the two reduce images, for each robot an image. It is checked if more than four spots are detected in these two images. If so, that means some spots were detected doubled; they appear on both images. Then the double detected spots have to be merged into a single one. If not more than four spots are detected, it is checked if less than four spots are detected. If that is the case, it means the estimated position was too bad and the robots spots are not in the range of the reduced image. Then a search over the whole area is performed once more to detect the spots.
Figure 6-1 flow diagram of image processing procedure
6.2. Reduction of image size

The timing analysis (see chapter 7.8) shows that the procedures boundary detection and black-white conversion take up a big portion of processing time. The reason is the large amount of data (pixels) needed to be processed. To overcome this problem, the image area to be processed can be limited, by e.g. estimating the current positions of the robots. If the estimation is accurate enough, only the area around the estimated position need to be process. Or in other words: the more accurate the estimation, the smaller can be the size of the reduced image.

6.2.1. Methods to estimate position

The different approaches to estimate the robots positions have been think of. They are using observer, (extended) leading-spot and quadtree decomposition.

6.2.1.1. Observer state variable

Several methods to estimate the robot position at the next sample time were considered. One of the more accurate but also more complicate methods is the usage of observer-control. An observer model is a mathematical description of the system to control, where the control-values are determined by the differences of the real system output and the observer model output. The mathematical background of observer control is given in chapter 3.6. The advantage of using observer-control to estimate the state variable is its accuracy. The drawback of this method is that it is relatively processing power consuming. That will eat up part of the time gained by the reduced image.

6.2.1.2. Leading spot

Another rather primitive method to estimate the position at the next time step, is simply take the leading spot from the pair of spots as the estimated position. The leading spot is a good estimation, because the leading spot pointing pretty much to the forward moving direction of the robot. Several conditions may create trouble for this method: the defined track has extremely sharp curves, the speed of the robot is extremely fast (the position of the robot is further way from the previous leading spot) or the controller is extreme swiftly (it does huge angular correction). These conditions practically do not or only partly occur in this project. Therefore even though this method is very simple, it is very useful in the practice. It does not require extra processing power, since the spots need to be detected for the control-purpose anyway.

6.2.1.3. Extended leading spot
To overcome one drawback, the speed-dependency, of the “leading spot method”, a simply additional calculation can be done: using the orientation of the robot, the steady state forward moving speed and the time step to estimate to next position. Figure 6-2 shows a robot over 2 time steps. The big blues circle is originally position of it. The smaller blue circle is the leading spot. The magenta arrow is the speed-vector of the robot. After one time step $\Delta t$ the robot moves to the position of the big green circle. The new position can be computer by the Equation 6-1. The advantage of this method is that with very little extra work, the robots positions can be estimate more accurate.

$$
\begin{align*}
    x_{n+1} &= x_n + (V \Delta t \cdot \cos \theta) \\
    y_{n+1} &= y_n + (V \Delta t \cdot \sin \theta)
\end{align*}
$$

Equation 6-1

6.2.1.4. quadtree decomposition

Quadtree decomposition algorithm is another possible approach to reduce an image down areas of interest. Unlike position estimation, which uses knowledge about the system, quadtree decomposition purely uses information out of the image. It is a recursive method, which subdivide an image into 4 equal-sized quadrangles as long as information of interest is available. E.g. the quadtree decomposition implemented in MATLABs image-processing toolbox qtdecomp uses homogeneity as criterion to decide whether to divide or not. That means area with big contrast and many edges are divided finer, while plain areas remain undivided. Figure 6-3 shows an example.

Figure 6-3 a demonstration of quadtree decomposition

---

7 This figure is taken from MATLAB’s demo
For our purpose, after decomposition, only squares smaller than a certain size needs to be examined. The other areas can be ignored. The draw back of this method is that the Quadtree decomposition algorithm is rather complicated and needs relative long processing time. Besides after division, some spots might lie between several square, which need to be composite back again. This composition algorithm will take extra time too.

6.2.2. Size of reduced image

![Figure 6-4 minimum size of reduced image](image)

The minimum size of the reduced image is the distance of the both spots plus their radius’s, as shown in Figure 6-4. With distance being 10 cm, $R_1$ and $R_2$ being 5 cm and 4 cm, the square will be 19 cm x 19 cm. A pixel size approximately equals 4.75mm. That means the reduced image should have a minimum size of 64 x 64 pixels. In the project squares of 100 x 100 pixels were used as reduced areas.

6.3. Object detection

The procedure of objection detection (from the image) is the following: colour conversion, boundary detection and Least Squares Fitting.

6.3.1. Colour conversion

In colour conversion a threshold value is used. If the pixel-gray value is bigger than this threshold, that pixel is assigned as white, otherwise as black.

6.3.2. Boundary detection
Boundary tracing is one of the basic segmentation techniques. It is used to detect edges and get the exterior boundary pixels in the image of object. In our project the regions of reflex marks can be detected in corresponding BW images. The boundary tracing is used to detect the inner boundary of the detected regions. In our project boundary tracing can be used to retrieve boundary chains of the reflex marks regions and then use these boundary chains to get the centre of the reflex marks.

6.3.3. Least Squares Fitting

The idea of this method is to find the centre positions of the reflex marks by fitting the contours to the retrieved boundary chains. Least Squares Fitting (LSF) is one method to fit simple contours to experimental data which is one of the basic problems in pattern recognition and computer vision. The method of least squares assumes that the best-fit curve of a given type is the curve that has the minimal sum of the deviations squared (least square error) from a given set of data [21].

Given \( n \) values \((x_i, y_i), 1 \leq i \leq n\), where \( x \) is the independent variable and \( y \) is the dependent variable, The fitting curve \( f(x) \) has the deviation (error) \( d \) from each data point, i.e., \( d_i = y_i - f(x_i), 1 \leq i \leq n \). According to the method of least squares, the best fitting curve has the property that:

\[
F(d) = \sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 = a \min
\]

Equation 6-2

Since the reflex marks are in circle-shape, then the LSF method can be used to performing circle fitting to the given boundary pixels coordinates. For the realization of circle fitting the work [6] can be referred.

6.4. Merging of objects

In the overlapping areas, but also when using size-reduced images, a spot can be captured twice. A reliable method to decide which spots are captured twice is required.

On the one hand, the number of expected spots is well known: \( 2 \times (\text{number of robots}) \). That means in the case more \( 2 \times (\text{number of robots}) \) of spots are detected, assuming all light disturbances has been successfully filtered out, then some spots were captured twice: \((\text{Number of Captured spots}) - (2 \times \text{number of robots}) = (\text{number of twice captured spots})\).

\[
M - (n \times 2) = k
\]

Equation 6-3
On the other hand, the twice-captured spots have the same, or nearly the same coordinates in the mm-domain. Therefore the Euclidian distance (or square root error) of two spots is a good measure to decide whether the two spots are actually the same spot or not.

Using these two relations, a method to determine double captured spots is developed. The steps of the methods are as described below:

- Collect detected spots into an array for each camera (or image)
- Swap array if the second is longer than a first one.
- Build a two-dimensional table, with the elements of the first array as elements of the columns; the elements of the second array as rows
- Compute the distance of each element-pairs; for reducing computation, squared distance (without root extraction) is sufficient: \( \Delta x^2 + \Delta y^2 \). Enter the results into the table.
- Find out the smallest \( k \)-entries in the table, these are the pairs the spots, which are representing actually the same physical spot.

Table 1 shows an example of such a table. In the first image 4 spots (3 fully, 1 partly) have been detected, 3 spots have been detected in the second image, see Figure 6-5. In the cells the distances between the two spots are entered. According to Equation 6-3 three \( \text{Distance}_{nm} \) indicate which spot-pairs are actually the same spots. As a control-mechanism to guard the correctness of this method, the rule can be applied, that each row and each column is only allowed to posses one of the smallest distances.

<table>
<thead>
<tr>
<th>Img1</th>
<th>Spot1</th>
<th>Spot2</th>
<th>Spot3</th>
<th>Spot4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spot1</td>
<td>( \text{Distance}_{11} )</td>
<td>( \text{Distance}_{12} )</td>
<td>( \text{Distance}_{13} )</td>
<td>( \text{Distance}_{14} )</td>
</tr>
<tr>
<td>Spot2</td>
<td>( \text{Distance}_{21} )</td>
<td>( \text{Distance}_{22} )</td>
<td>( \text{Distance}_{23} )</td>
<td>( \text{Distance}_{24} )</td>
</tr>
<tr>
<td>Spot3</td>
<td>( \text{Distance}_{31} )</td>
<td>( \text{Distance}_{32} )</td>
<td>( \text{Distance}_{33} )</td>
<td>( \text{Distance}_{34} )</td>
</tr>
</tbody>
</table>

Table 1

Figure 6-5 shows a possible situation of the Table 1. The red spots belong to the leading robot, the magenta to the tracing robot. The blue hatched field is the reduced image according the estimated position of the leading robot; the green hatched field is the reduced image according the estimated position of the tracing robot. The numbers in the circles in front of the semicolon is the numbering of spots of the second image, behind the semicolon is of the first image.

Remark:
The merge function has two operational modes: merge and drop. In the merge-mode the coordinates of the spots to merge are really merged, that means the arithmetic mean is taken from the two spots to merge. In the drop-mode the coordinates of only one spots is kept, the coordinates of the other spots is dropped.

The merge-mode is meant to use in the “whole-area search” case, while the drop-mode should be applied in the “reduced-area search” case. In the “whole-area search” the spots in the operation field will be captured fully, therefore merging of coordinates can guarantee a smooth transition from one camera to the other. Oppositely, in the “reduced-area search” case spots might be captured only partly. After the boundary
detection and shape fitting algorithm, the coordinates of a partly captured spots is distorted. Therefore only coordinates of spots captured fully should be taken.

Figure 6-5 two reduced images with spots captured doubled
7. Implementation Issues

The project has been implemented successfully, so that the desired control can be archived. Several steps of implementation have been gone through and different technical difficulties have been solved in order to archive the final result. These steps and solutions to the difficulties are the objects of this chapter. Therefore this chapter is not focused on one topic as in other chapters, but is rather a collection of various topics.

The basic steps from the start to the result were the following:

- test of the hardware in the used combination
- build simulation control-model
- modification the simulation model into real-time model, which means mainly replacing the system matrix by the camera and robot interface-functions
- solve the image processing problems
- check for timing problems and resolve of them

Certainly there have been many iterative work, means going back to previous steps, which is not unusual in technical projects.

7.1. Hardware feasibility

The feasibility of use a single camera and a single robots has been demonstrated in[6]. The logical extension is to test the feasibility of using multiple cameras and multiple robots.

7.1.1. Multiple cameras

By checking the technical documents of the camera, at least two possibilities to use two cameras at the same time were found. At first connect the camera in daisy chain, which means connecting the first camera via a firewire with the second one, then connecting the second camera to the workstation. There is no direct connection between the first camera and the workstation.

The other configuration is using a firewire hub. Both cameras are plugged to the hub, and the hub is connected to the workstation. This is the configuration chosen in the project.

After having installed the driver of the camera, by using the right device-id number MATLAB can address the devices with the following commands:

```matlab
vid1 = videoinput('winvideo', 1);
vid2 = videoinput('winvideo', 2);
```
The AriaSfunc provide by [17] is the interface between MATLAB and ARIA AmigoBot’s application interface. Figure 7-1 shows their relation. This S-Function can be modified in order to control the robots through different COM-ports. For more information about the AriaSfunc itself, please refer to [17].

The number of COM-port, to which a certain robot or rather its radio transmission station should be connected, is set through the constructor of the class ArSimpleConnector from the ARIA-library:

```c
argv[1]="--robotPort";
argv[2]="COM2";

simpleConnector = new ArSimpleConnector(&argc, argv);
```

For the compilation step refer to the compilation guide [23]. The DLL’s are compiled by Microsoft Visual C++.

The compiled DLL’s can then be called from Simulink.

### 7.2. Simulation models

The SIMULINK models are basically the technical realization of the abstract control model.

Figure 7-2 shows the most upper level of the simulation model. On the left hand side is the leading robot, on the right hand side the tracing robot. The variables, sent from the
first robot and the second robot, are need for computing the distance between both. Details about the computation of the distance are written down in subchapter 7.2.3.

### 7.2.1. Model of leading robot

![Figure 7-3 Model of leading robot control system](image)

Figure 7-3 shows the subblock of leading robot control system. The robotAddCamera Block receives as input the speed difference of the wheels and outputs the quantity $\theta$, $x$- and $y$-position of the robot. It basically outputs the same as the ScameraFunc3 (see Chapter 6.1) does, with the difference that the angle $\theta$ is an accumulated value and might exceed $2\pi$. Therefore the block “normalize” is needed, but only in simulation, to convert $\theta$ into values between $–\pi$ to $\pi$.

The block “coortranOval” calls the M-function which transfers the robot’s absolute coordinate and orientation into relative coordinate and orientation to the defined reference track. The coordinate transformation is described in 7.2.3. The output of the “coortranOval”-block is multiplied with the controller gain $K$ which is derived as in chapter 3.4 described. The $N$-matrix at the top-left in the figure is the gain matrix for the reference value according to Equation 3-32.

![Figure 7-4 leading robot model](image)

In Figure 7-4 the inside of the subblock RobotAddCamera is presented. It is basically the signal-flow diagram for the Equation 4-4, Equation 4-5 and Equation 4-6, which describe
the robots steering mechanism mathematically. The integrators connect the state variables with its derivatives.

### 7.2.2. Model of tracing robot

![Figure 7-5 Model of tracing robot control system](image)

The inside life of the tracing robot is shown in Figure 7-5. Instead of a basic controller, an integral controller is added here (left bottom corner). The reason of using integral controller for the tracing robot is because it is able to compensate completely constant disturbance, as shown the chapter 3.5. The leading robot moves with a constant speed forward, but when it steers the direction, the robot might varies its speed component in the direction of the reference track. This variation can be regarded as disturbance to the second robot. Therefore an integral controller has been chosen. Due to the larger state vector the dimensions of $K$ and $N$ are extended (3 state variables). Chapter 7.4.1 will show the difference between with and without the integral part.

![Figure 7-6 tracing robot model](image)
The subblock robAddCamera2 is similar to the robotAddCamera block of the leading robot, except that the speed is no longer a constant gain, but a (controlled) input. See Figure 7-6.

### 7.2.3. Define reference Track and transfer Coordinate

The differential equation system established for the robots in Chapter 4, Equation 4-4, Equation 4-5 and Equation 4-6, was based on translational motion along the x-axis. However our intention is rather to define contour, like straight line, circle or combination of those, as reference track. Therefore the y-coordinates and orientation of the robots are needed to be transformed according to the desired track. The basic idea behind the coordination transformation is to set the origin of the coordinate axis on the reference track, namely on a point, which has the shortest distance to the robot; in other words where the reference track is orthogonal to the robot current position.

Figure 7-7 a) and b) shows two examples of coordinate transformation. The black axes are the original axes, the blue line is the desired track and the green axes are the transformed axes. The transformation is especially simply for straight line and circle, because the orthogonal anchor point can be easily found.

**Straight line**

When the reference track is a straight line parallel to the x-axis, the y-coordinate simply need to be subtracted by the height $q$, the orientation $\theta$ remain the unchanged due to the trigonometry rules, see Figure 7-8. The x-coordinate does not play any role for the control of the leading robot. For the tracing robot, the state variable “distance to leading robot” is obtained by subtracting tracing robot absolute x-coordinate from leading robot absolute x-coordinate, assuming the moving direction should be from left to right, otherwise reversal. Since only the difference of both values is of importance, no especially coordinate transformation is needed for the x coordinate.
For circular track the situation is slightly more complicate. The momentarily coordinate axes are permanently changing according to the position of the robot, whereby the y-axis is always pointing exactly to the middle of the circle. Figure 7-9 a) shows two examples where the driving direction is assumed to be anti-clockwise.

When the \( g \) is defined as the distance between the circle-middle and robot position, \( g \) is obtained by:

\[
g = \sqrt{(x_c - x)^2 + (y_c - y)^2},
\]

Equation 7-1

, with \( x_c \) and \( y_c \) being the absolute coordinates of the circle-middle.

Then the state variable \( e \), the y-orthogonal distance to reference track, is:

\[
e = R - g
\]

Equation 7-2

, with \( R \) being the radius of the circle.

See Figure 7-9 b): The angle \( \zeta \), the absolute angle of the perpendicular of the robot to the circle, is obtained sign-corrected by the MATLAB function \( \text{atan2}(X, Y) \):
\[ \zeta = a \tan 2(y - y_c, x - x_c) \]

According to the rule of trigonometry:

\[ \alpha = \text{normalized}(\zeta + \pi / 2) \quad \text{Equation 7-3} \]

, and finally \( \theta_n \), the transformed orientation of the robot is:

\[ \theta_n = \theta_o - \alpha \quad \text{Equation 7-4} \]

, with \( \theta_o \) being the absolute angle of robot orientation.

![Figure 7-10 definitions of distance on circle](image)

Different definitions of distance can be thought of. In this project the definition has been chosen, which means the way between the two perpendicular-anchors on the reference track, see Figure 7-10. No performance analysis regarding control behaviour has been carried out between this definition and other possible definitions, which may be an object to investigate in coming project.

The proportion \( d \), the distance, to circumference \((2\pi R)\) is the same proportion \( \zeta_{diff} \) to \( 2\pi \).

Therefore the state variable \( d \) is:

\[ d = \frac{\zeta_{diff}}{2\pi} \cdot 2\pi \cdot R = \zeta_{diff} \cdot R \quad \text{Equation 7-5} \]

Whereby

\[ \zeta_{diff} = \text{normalized}(\zeta_1 - \zeta_2) \quad \text{Equation 7-6} \]

### 7.3. Curvature error
The occurrence of curvature error bases on the permanent alternation of the momentarily coordinate axes as described in 7.2.3.

Here the curvature error is investigated particularly for circle track. Assuming starting at equilibrium state, during the first control step the robot moves along the tangent for the distance $s$:

$$s = V \cdot t_{\text{control\_step}}$$  \hspace{1cm} \text{Equation 7-7}

$V$ is the average speed as defined in the chapter 4.1, while $t_{\text{control\_step}}$ is the duration of each control step. At the end of the first control-step and with the start of the second control step, the robot is no longer located on the defined track, but on the tangent, namely $s$ mm away from the starting position. At this time, the new momentarily coordinate axes are the perpendicular of the robot with the circle and its tangent. This error needs to be corrected in the second control step.

Therefore the curvature error (in case of a circle) depends on $V$, $t_{\text{control\_step}}$, and $R$ (average speed, control-step time and radius). Due to Pythagoras $C$ is:

$$C = \sqrt{R^2 + S^2}$$  \hspace{1cm} \text{Equation 7-8}

On the other hand $S_{\text{err}}$ is:

$$S_{\text{err}} = C - R$$  \hspace{1cm} \text{Equation 7-9}

Setting Equation 7-8 into Equation 7-9 follows:

$$S_{\text{err}} = \sqrt{R^2 + S^2} - R$$

, and with Equation 7-7 $S_{\text{err}}$ can be determined by

$$S_{\text{err}} = \sqrt{R^2 + (V \cdot t_{\text{control\_step}})^2} - R$$  \hspace{1cm} \text{Equation 7-10}
From the aspect of control, the curvature error (of a circle) can be considered as a constant disturbance, which can be reduced (or even eliminated) by an additional integral controller, see chapter 3.5.

The curvature error for the state variable $\theta$, orientation of the robot, is because of:

$$\tan(\theta) \frac{S}{R} = \frac{V \cdot t_{\text{control, step}}}{R},$$

Equation 7-11

$$\theta = \arctan\left(\frac{V \cdot t_{\text{control, step}}}{R}\right)$$

Equation 7-12

### 7.4. Simulation result

This chapter shows a selection of simulation results of different control models and different tracks. In 7.4.1 the general difference between the feedback and integral controller for the robot's control should be investigated. Therefore sometimes rather unreal numbers were applied on reference values and initial values. In contrast to 7.4.1, 7.4.2 is a preparation for the real time implementation. Accordingly more realistic numbers were used there.

#### 7.4.1. Basic control vs. integral control

First we look at the simulation result of a model, where both robots use the basic controller, means no integral controller is involved. In this section the reference track used is a simple straight line (parallel to the y-axis).

![Figure 7-12 a) and b): Track of leading robot, Track of tracing robot](image)

Figure 7-12 a) and b) show the simulated track of two robots. The reference track is defined as a horizontal straight, which is parallel to the x-axis. The starting conditions are the following:
1. Robot

Reference value for e (y-directional distance to reference track): 5
Initial value for robot orientation $\theta$ (in rad): 0.1
Initial value for robot position x-coordinate: 10
Initial value for robot position y-coordinate: 0.1

2. Robot

Reference value for e (y-directional distance to reference track): 5
Reference value for d (x-directional distance to 1. robot): 5
Initial value for robot orientation $\theta$ (in rad): -0.1
Initial value for robot position x-coordinate: -1
Initial value for robot position y-coordinate: -10

At the first glance on the figures both robots reached the steady state and drove on the reference track, 5 mm above the x-axis. To see more detailed information, the state variables of the robots are displayed in Figure 7-13 and Figure 7-14 over time. From them, the necessity of the integral controller becomes clearer.

Figure 7-13 State variables of leading robot

Figure 7-13 shows three graphs: the two state variables theta, y-coordinate and the x-coordinate of the robot. Starting from their initial values, the theta is tangent to zero and the y-coordinate asymptotes to the set-value 5, as expected. The x-coordinate is a linear increasing function, which shows that the robot was moving parallel to the x-axis very soon after the start of the simulation.
Figure 7-14 State variables of tracing robot without integral part

Figure 7-14 shows the state variables of the tracing robot when no integral controller is used. \( \theta \) and \( y \) behave similar to the first robot; means reach the zero and 5 after a short while of simulation-begin, except having started from different initial values. The third curve in the figure presents the distance to the leading robot. Here we can see that it increases from 10 up to around 200 and stays stable there. It is stable but relatively far away from the set-value 5. In other word an undesired offset is introduced to the reference value. The reason is the disturbance introduced by the leading robot, as explained in chapter 7.2.2.

Figure 7-15 a) and b) State variables of tracing robot with integral part
To overcome the problem, the integral-part comes into play. Figure 7-15 a) shows the state variables when embedding the integral controller. The third curve shows that after the distance between both robots has reached a maximum of over 200 mm, it starts to decrease. At the end the distance stabilized at the desired value 5, as shown in Figure 7-15 b).

In order to gain a better understanding between the different behaviours of basic control and integral control, it is helpful to explore the input path of the tracing robot. Figure 7-16 shows the second element (responsible for the average speed) of the input branch $K \cdot x$ and the second element of the input vector $u$ itself when only basic controller is used. Figure 7-17 illustrates the behaviour while an integral part is added. It shows the second element of input branch $K \cdot x$, $K \cdot e$ and of the input vector $u$ itself. Comparing the both inputs (the $u$’s): an overshot is discernible at the time between 2 and 4 second in the case of integral control, while this bump is missing in the basic control. From Figure 7-17 b) it can be seen that for control with integral part the maximum of $u_{22}$ reaches a value of about 220 mm/s, while for basic control $u_{22}$ does not exceed 200 mm/s. This overshot is responsible for catching up the leading robot and avoids the constant offset, which occurs in the case of basic control.

When further look into the input branches of Figure 7-17, it can be recognized that the integral path provides the extra force to compensate the disturbance introduced by the leading robot. The disturbance is introduced at the beginning of the simulation when the system has not reached the steady state yet.
7.4.2. Simulation result of oval track

With the combination of straight lines and circles, more complicate track contour can be formulated. Figure 7-18 shows an oval track which has been defined and tested in the project. The oval track consists of 6 sectors: two horizontal straight lines (2 and 3), a half circle (1), a vertical straight line (5) and 2 quarter circles (4 and 6). The coordinate of important corner point are noted. Depending on the position of the robot, the various coordinate transformation mode, as described in 7.2.3, are applied. The green lines show the extension of the sector boarders. The driving direction is set as anti-clockwise.

The initial condition of the robots for the simulation is:

\[ V = 100 \]

1. Robot
Reference value for $e$ (y-directional distance to reference track): 0
Initial value for robot orientation $\theta$ (in rad) : 0
Initial value for robot position x-coordinate : 3000
Initial value for robot position y-coordinate : 200

2. Robot

Reference value for $e$ (y-directional distance to reference track): 0
Reference value for $d$ (distance to 1. robot) : 500
Initial value for robot orientation $\theta$ (in rad) : 0.1
Initial value for robot position x-coordinate : 2300
Initial value for robot position y-coordinate : 20

Figure 7-19 shows the leading robot’s track of the simulation, Figure 7-20 the tracing robot. Overall the robots are driving quite exactly on the defined track, the green one. However one can clearly see the effect of curvature error, as explained in 7.3, at the corner points, especially at the bottom left and top right corners.

![Figure 7-19 (left): leading robot track, poles: -0.7; -0.71](image)
![Figure 7-20 (right): tracing robot track, poles: -0.7, -0.71, -0.61, -0.62, -0.8](image)

Figure 7-21 and Figure 7-22 show the state variables and x-, y-coordinated of the robots. The red dashed line divided the curves into the sections according to Figure 7-18.

In the first 10 seconds for leading robot or rather the first 12 second for the tracing robot, the robots are in section 3, the lower straight line. The robots start from positions which are not so closed to the equilibrium state. Therefore the state variables are relatively far away from the desired value. On the other hand it can be seen that the controllers are fast enough to control the robots very close to the steady state at the end of this section.

When comparing section 1 of both robots the effect of integral control and basic control can be observed clearly: due to the curvature error, as described is 7.3, occurring in this section, the state variable y/orthogonal-distance ($e$) of the leading robot, which is controlled by the normal controller, get stabilized to an constant value, but not to zero. On the other side, in the same section, the state variable y/orthogonal-distance ($e$) of the tracing robot, which is controlled with a integral controller, also get drifted away from
zero (the steady state) first, but after having reached a peak, it starts to work its way to zero.

When comparing section 6 and 4 with section 1, in case of the leading robot the offset of y-distance is larger in the sections 6 and 4. Equation 7-10 confirms this behaviour: when \( V \) and \( t \) are constant, with smaller radius \( R \), the curvature error is bigger. Section 5, 2 and 3 are straight lines, therefore the state vector reaches the steady state. The same can be said according to the overshot of \( \theta \) (theta) in those sections referring to Equation 7-12.

The duration of section 6 and 4 (the two quarters circles on the right hand side) is too short, so that the (feedback + integral) controller of the tracing robot is not able to make \( e \) reaches zero. The state variable \( d \) (x-distance) remains close to the reference value, 500, with same bigger bumps at the time 60. and 80. second, see Figure 7-22. These are the time when the leading robot entry and leave the half-circle section. Smaller bumps can be found during the transitions between the sections 3, 6, 5, 4 and 2 too.
7.4.3. Improvement through pole placement

From the simulation results of the chapter previous subchapter it has been found out that the control system is too slow to follow the defined oval-track. In other words the poles are placed not left enough. On the other the poles cannot be placed arbitrarily left, because that will results into a nervous and instable system. Therefore different poles values have been tested and significant improvement was achieved. The speed used is $V = 200$.

Feedback control (for leading robot):

The simulations results with the new poles are presented on the Figure 7-23. Figure 7-23 a) is the leading with solely feedback control, b) with integral control. The red tracks are
control with the new poles. The blue lines are the track when using old poles, green is the reference track. In both case, the leading- and tracing- robot drive closer to the reference track.

<table>
<thead>
<tr>
<th>Leading robot</th>
<th>Tracing robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old poles</td>
<td>New poles</td>
</tr>
<tr>
<td>-0.7</td>
<td>-1.2</td>
</tr>
<tr>
<td>-0.71</td>
<td>-1.21</td>
</tr>
</tbody>
</table>

Integral control (for both robots):

For better visibility x-/y-axis of the figures are not scaled to 1:1 from here on. From Figure 7-24 a) it can be seen that also for the integral controlled leading robot improvement has been achieved through more suitable placement. In general it can be said that the robots get close to the reference track faster through faster control. However there is a price to pay with faster poles. The range the allowable initial error is reduced with a faster reacting control system. The new poles are applied in the real time tests too (chapter 8).

![Figure 7-24 a) & b) improvement through pole placement, integral control for leading & tracing robot](image-url)

<table>
<thead>
<tr>
<th>Leading robot</th>
<th>Tracing robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old poles</td>
<td>New poles</td>
</tr>
<tr>
<td>-0.7</td>
<td>-1.2</td>
</tr>
<tr>
<td>-0.71</td>
<td>-1.21</td>
</tr>
<tr>
<td>-0.8</td>
<td>-1.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Old poles</th>
<th>New poles</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.7</td>
<td>-1.2</td>
</tr>
<tr>
<td>-0.71</td>
<td>-1.21</td>
</tr>
<tr>
<td>-0.8</td>
<td>-1.1</td>
</tr>
<tr>
<td>-0.61</td>
<td>-1.0</td>
</tr>
<tr>
<td>-0.62</td>
<td>-1.001</td>
</tr>
</tbody>
</table>
7.4.4. Improvement through prediction

At the end of the project thoughts for further improvement were made. One of them is how to eliminate or reduce the effect to curvature error. In the subchapter 7.3 it was explained that curvature error is caused by the constant change of the momentarily (relative) coordinate-axes. The distance to the reference track is taken as one of the state variable and the control input is derived among others from it. The controller tries to drive the robot to the tangent (the momentarily y-axis) to minimize the error to track. However after one control-step-time, the tangent to steer to is already another one.

Therefore it is more intelligent to predict the robot position after the time of one control-step, and use this predicted position to determine the momentarily coordinate-axes. For simplification reason, it was assumed that the velocities of both wheels are the same and the robot’s next position is on the straight line of it’s momentarily orientation. Or in other words it is:

\[ x_{pre} = x_{cur} + \cos(\theta) V \cdot t_{step} \]  \hspace{1cm} \text{Equation 7-13}

\[ y_{pre} = x_{cur} + \sin(\theta) V \cdot t_{step} \]  \hspace{1cm} \text{Equation 7-14}

As said, it is a simplification, which does not take the velocity difference of the wheels into account. However for the short time duration of one step time, this simplification does not introduce big difference.

Figure 7-25 shows the simulation result for the feedback controlled leading robot with the improved poles (see last subchapter 7.4.3) running with an average velocity of 400 mm/s. The red line is the track when using predicted position; the blue is without prediction. It can be seen that little but certain improvement is achieved. One has to take into account that the speed of 400 mm/s is relatively fast for the controller.
Beside for the feedback control, the prediction method was also tested for the integral control. Here the improvement through prediction is, relatively seen, more significant. At many points on the track the improvement is about 30 to 40 percent, as it can be seen by eye. The reason is that the integral reacts quicker anyway, so that the controlled track was relatively close reference track. By reducing the curvature error, which is relatively seen not so large, the control is able to steer the robot in very short time to the reference track. It should also be mentioned, that the track is in general much less oscillating, means less over-steering. That is because the controller is fed with future information of the track, so that unnecessary control, which will be compensated by the track alternation anyway, can be saved. It is as the controller drives foresightedly.

The prediction method was only simulated for leading and has not been implemented into the real-time test due to the shortage of time in the project. But it shows a promising way to improve the control at curves. Especially in the real-time implementation, with prediction beside the track alternation, also the signal transmission delay can be compensated.

### 7.5. Width of overlapping area

The width of the overlapping area needed to be chosen correctly to ensure a proper function of the image processing. When no reduced image mechanism, as described in chapter 6.2, is used, the minimum width of the overlapping area must be bigger than the diameter of the biggest reflexion spot. Otherwise it might happen that none of the both cameras is enable to capture the entire spot/circle, as shown in Figure 7-27 a). The diameter of the biggest spot is 10 cm. During the testing phase without having integrated the reduced image mechanism, the overlapping area was set to 200 mm, which has been proven as stable.
When the reduced image mechanism is applied, which is the case of the final version of implementation, the overlapping area has to be increased; the area must be at least as width as the square of the reduced image, with the same reason as mentioned above. The size of the reduced images is squares of 100 x 100 pixels. With a pixel-size of 4.75mm pro pixel, the overlapping must be at least 475mm wide, assuming the position-estimation approach works absolute correctly. In the final implementation, the wide of the overlapping area was set to 800 mm and enabled a stable operation.

7.6. Saturation vs. overdrive

In the paper [22] extensive investigation has been made on non-linear control of a mobile robot. It treated the same type of robot as used in this project. The investigation shows the problem of linear control with large initial error for this type of robot-steering. When the initial error is too big, the control will over steer the robot. The robot will drive a curve, which is so sharp that it is no more the shortest way to the reference track. The left track on Figure 7-28 shows such a situation. The red dashed line shows another possible track; it will never reach its steady state. This undesired effect is introduced through the linearization of the system-model.

The investigation’s result in the paper [22] leads to the idea to reduce the over steering effect by insert saturation blocks to limit the angular error. Since the shortest way to any reference track is the perpendicular to it, the angular error should be limited to \( \pi/2 \). The saturation blocks need to be inserted into any paths, which contribute to the control-variable speed difference. In case of integral control, in the paths \( e \) a saturation need to be inserted, so that that input become 0. The theoretical behaviour of a saturated system is displayed on the right of Figure 7-28.
7.7. Real-time models

The variables $x_{lead}$, $y_{lead}$, $teta_{lead}$ and $sec_{lead}$, see Figure 7-29, are information of the location of the leading robot and are used by the tracing robot to determine the distance to the leading robot.

In the timer block the S-function “TimerSfunc1” is located. This function is essential for the real time operation. This function halts the whole system when the set step-time is not used up yet. More about timing consideration can be found in 7.8.

The main difference between the simulation models and the real time models is the replacement of the mathematical robot model through the robot’s and camera’s software-interfaces. They are called in the S-functions CameraSfunc3 (see Figure 7-29) and RobotCom2 (see Figure 7-30) or rather RobotCom3 (see Figure 7-31). Besides the normalization blocks for angle are not needed anymore, because the angle output by CameraSfunc3 are in the range between $\pi$ and $-\pi$ anyway. As explained in 7.6 saturation blocks have been inserted (see Figure 7-30 and Figure 7-31) to avoid overdrive.
7.8. Timing consideration

By timing it is meant in this project the time for each control-step and related issues. This subchapter is dedicated to these issues. This is an essential topic for the successful operation of the entire control system. Due to the increasing number of hardware involved and the enlarged operation field, the timing become much more critical compare to the previous projects.

7.8.1. Timing in real time system

Our system to control has been described as a continuous system. However we use MATLAB/SIMULINK discretely. In other words the system computes step by step according to a set fixed step-time. In order to run a continuous system correctly in a discontinuous environment, some preconditions need to be fulfilled.

First at all, the step time, also called sampling time in system theory, must be relatively short. The step time must be short compare to the system behaviour and controller pace.
Furthermore the step time should be always or most of the time constant. Otherwise the system would not be deterministic.

However this can lead to contradiction: on the one hand the step-time should be as short as possible; on the other hand the step-time should be long enough to allow the system finishes all tasks.

In order to ensure constant step-time, an idea of creating artificial time step is applied. In the concrete implementation, a function, TimerSfunc1, was implemented for this purpose. After the system has processed all tasks (hardware-interfacing, image-processing, coordinate transformation, control, etc.) this function still halts the system as long as the defined step-time is not over yet.

7.8.2. Timing challenges

The possibility for modify the system/control pace is only limited, because we cannot make the controller too slow, otherwise it will not be able to catch up with the curvature error, speed of the leading robot $V$ and other system disturbance like quantization error.

Even thought the largest possible step-time for reasonable control has not been determined analytically, tests have shown that a step-time longer than 0.15 sec starts to create troubles. Better would be step-time of around 0.1 sec. Previous project works used this step-time and were proven being reliable.

In the initial implementation the image reduction mechanism was not built in yet. During this testing phase the timing requirement, a step-time shorter than 0.15 sec, could not be achieved.

Therefore the challenge lies in making the system being able to process one complete step within 0.15 sec.

In the previous project [6] the MATLAB commands “tic” and “toc” were used to analyse the timing of the functions. With “tic” a timer is started and with “toc” it is stopped. “tic” and “toc” are two very useful tools because of their ease to use. On the other side they are lack of sophisticated analytical capability. Due to the grown complexity of the project, especially in the SCameraFunc3 function, more advanced timing analysis method was required. It turns out that the “profile”-function in MATLAB is an extremely powerful timing analysis tool. The HELP in MATLAB describes it as following:

“The profile function helps you debug and optimize M-files by tracking their execution time. For each function in the M-file, profile records information about execution time, number of calls, parent functions, child functions, code line hit count, and code line execution time. Some people use profile simply to see the child functions; see also depfun for that purpose. The Profiler user interface, opened with profile viewer, provides information gathered using the profile function, but presents the information in a different format from profile report.”
Real-time tests were running with the profile-function switch on executed in the background to collect timing function of each called function. After the test, the collected timing information are displayed and analysed.

From the result of the function “profile”, it turns out that the processing of the image consumes the largest portion of time. Especially the rather simple algorithm im2bw, (convert to black-white image) takes up extraordinarily a lot of time compare to the other functions. The reason for it is the huge amount of data needed to be processed. By using two cameras, 2 times 480 x 640 pixels are needed to be processed. That is 614400 pixels!

![Figure 7-32 result of profiler (without reduction of image size)](image)

Figure 7-32 shows the resulted output of the profile-function in case no reduction of image size is applied. The most critical functions are circled. im2bw is circled with red ellipses. This function takes up the biggest portion of computation time, namely 21.1% of it. Other function, which use up much time is the “bwboundariemex”, is an internal function used by boundary detection function bwboundaries.

From this insight, the necessity to speed up the image processing becomes unambiguous. An efficient way to achieve this is to reduce the number of pixels to be processed. The idea to reduce the image to process down to an estimated area was born. The details about image reduction are written down in chapter 6.2.

Furthermore from the result of the “profile” function, it can be seen that the hardware (cameras and robots) interfacing takes up over 30% of the execution time, namely imaqdevice/getdata 8.3%, ariacom3sfunc 11.8% and 11.8% ariacom2sfunc. That is a significant portion of one step-time (0.1 sec).

Further increment of robots or cameras might lead to new timing challenges. Beside improving system speed, other attempts might need to be explored. One possibility is to convert the continuous system into a discontinuous system.
Figure 7-33 shows the profiler’s result after including the image reduction algorithm to the image processing flow. The time consuming contribution of *im2bw* drops down to 2.3%. The internal function *bwboundariemex* even disappeared from the top of the list.

### 7.9. Analyse controller matrix

Analysing the control matrix, which is computed by MATLAB, is not absolute necessarily, but might be very useful for debugging and understanding the system. In particular hints about reasonable pole placement and correctness of the system equations can be found.

The input for the first robot is calculated by:

$$ u = [k_1 \quad k_2] \begin{bmatrix} \theta \\ e \end{bmatrix} $$

Equation 7-15

, where

$$ K = \begin{bmatrix} 148.5 \\ 0.2609 \end{bmatrix} $$

From $K$ it can be seen that the input $u$, difference of wheel speed, dependents on both quantity: $\theta$ and $e$. That also means that with $u$ both state variables are controlled; by steering the robot direction, its orientation and distance to the reference track are controlled.

By using a saturation block $\theta$ is limited between $-\pi/2$ to $\pi/2$. That leads for $k_1 \cdot \theta$ to a maximum value of 233.2633 and a minimum value of -233.2633. With $V$ set as 200, controller is designed a little bit too fast; means the poles are placed too left. However we know that linearized state space control is inappropriate for large initial error anyway. $\theta$
around $\pi/2$ or $-\pi/2$ is certainly a large error. Therefore $k_1$ is still in the acceptable range. The second term which contributes to $u$ is $k_2 e$. Let say the initial $e$ is 500mm, that is a large error considering a field of 2000mm high, the term would result to $0.2609 \pm 500 = \pm 130.4500$. This is in a reasonable range with $V = 200$.

For the second robot the feed back controller $K$ is:

$$K = \begin{bmatrix} 213.15 & 0.7152 & 0 \\ 0 & 0 & -1.41 \end{bmatrix}$$

It can be seen that the new state variable $d$ only contribute to input $u_2$ (forward speed), and not to $u_1$ (speed difference of the wheels). Reversely $\theta$ and $e$ remain only effecting $u_1$, but not $u_2$. This aspect of the system equation is correct too, because $u_2$ is the only input which can change the distance to the leading robot.

$k_{11}$ and $k_{12}$ are in the similar magnitude as $k_1$ and $k_2$ of the first robot, therefore the reasonableness can be overtaken. If the initial distance between the robots is 300 mm more than the desired distance, $u_2$ (neglecting the effect of the integral controller) will become $-(-1.41) \cdot 300 = 423$ mm/s. That is in the range where the motor of the robot can drive.

The integral controller $Ki$ is:

$$Ki = \begin{bmatrix} 0.1588 & 0 \\ 0 & -0.497 \end{bmatrix}.$$ 

The Equation 3-38:

$$u = -(Kx - Ke) = -(K - K_i) \begin{bmatrix} x \\ e \end{bmatrix}$$

tells that the input $u$ is contributed by the integral controller in form of:

$$u_i = K_i \cdot e$$

Equation 7-16

e_1$ is the integral of the difference between the first reference value $r_1$ and the second state variable $x_2$: the desired distance to track and the momentary distance to track. The same can be said for $e_2$, but instead from the track distance, it is computed from the distance to the leading robot. Since $k_{i12}$ and $k_{i21}$ are zero, $e_1$ and $e_2$ affect the control absolute separately. That confirms the independency between the track-distance and distance of the robots.
8. Track Following Tests

In this chapter the end results, the controlled real track, are presented and analysed. For the tests two different speeds were used in order to show the performance and limits of the controlled system. Different models were tested too, to show the inadequacy and improvement of each model. The reference track is the same oval-track, which was used for simulation, see subchapter 7.2.3. Therefore a direct comparison to the simulated results is possible. All results are reproducible with the files appended in CD.

The data from the real-time tests were simply stored into the workspace of MATLAB during execution. The analysis on the data was done separately after the execution of the tests. This enables the most realistic real-time behaviour of the system, since analysis during the execution would take up extra computation time and distort the timing-behaviour.

[6] Theoretically, this real-time model should work the same as the simulation model if the robot system is in ideal condition. This means the states are measured correctly, the experimental environment does not produce uncertainties or disturbances, no transmission delay occurs, the feedback input is reacted on the velocities of the two wheels \( V_{left} = V-u, V_{right} = V+u \) precisely etc. However that is not the case for the real robot system. One reason is the speed quantization in real robot system. The quantization step is 40mm/s. This affects like disturbance to the system. The controlled velocity cannot be precisely realized on the driving wheels. Beside transmission delay is introduced, as the signals need certain time from the work station to the wheel. Furthermore the calibration inaccuracy in the overlapping area causes another source of error, see subchapter 5.6.

The controllers are determined by using the same poles as in the chapter 7.4.3 “Improvement through pole placement”. The distance to the leading robot is set to 500 mm, just as in the simulation.

To avoid confusion, it is mentioned here that the initial states (position and orientation) of the robots are NOT the same for the simulations and real-time tests.

8.1. Low speed 200 mm/s results

200 mm/s was applied in the low-speed version of the series of tests. Feedback control and integral control has been tested. It should be take care that the x-y-axes of some figures are not scaled equally due to readability.

8.1.1. Feed back control

Figure 8-1 shows the track of the leading robot when only feedback control is applied on it. As explained in 7.4.2 already, the curvature error leads to constant offset to the reference track in some parts. This effect is in the real-time tests even stronger than in the simulated results. One reason for the amplification of this effect is the transmission delay,
since curvature error depends on the step-time. Time is needed to transmit signals from MATLAB/ SIMULINK’s API to the Firewire-PCI card, to transmission station, to radio modem on the robot, to robot micro-controller, to motors, until the motor turns the signals into motion. The delay has not been investigated in the project. However it is obviously that this kind of system-time delay has a larger effect on sudden change of direction than gradual change.

Quantization is a possible source of inaccuracy. However it is hard to predict when and how large the effect of quantization error will be exactly. Because that depends how closed the quantized value is to the non-quantized value. That can differ from case to case.

In distance the reference track is bigger in the curves (see y-distance on Figure 8-3), the same as in the simulation, because feedback cannot compensate the disturbance introduced by the curvature error. The need of integral control is even more obvious.

Different than in the simulation, the distance to track has a little jump around the overlapping area 2800 mm. Inaccuracy in the overlapping area is discussed in 5.6.

Figure 8-2 shows the track of the tracing robot. For reasons mentioned in subchapter 4.2 the tracing robot control system is implemented with integral controller by default. The benefit of using integral control is even clearer through real-time tests. On the big half-circle (section 1) the robot drives very close to the idea track.

Figure 8-2 (right) track of tracing robot, V=200, with integral control; poles -1.2, -1.201, -1.0, -1.001, -1.1

The distance to the leading robot is very close to 500 in the sections 2, 1 and 3, but varies little (less than 25mm) in the section 6, 5 and 4, see Figure 8-4. That is nearly the same result like the simulation result, see Figure 7-22. The distance to track shows a very similar behaviour to the simulated result too; in some locations it is even better. These locations are when leaving the big half circle (section 1) and leaving the upper small quarter circle (section 4). Possible reason is the quantization for the velocity.
Actually the curvature error and system-delay error have to not be irremovable. Both might be solved or reduced by intelligent predictive method. By knowing the speed, the control step time (plus system-delay), the value for correction can be determine, see 7.4.4.

The quantization error is more difficult to remove. One possible solution is to replace the AmigoBot robots by robots with continuous speed control.
Figure 8-5 shows the rest-time produced by the STimerFunc1. As mentioned in chapter 7.8.1 STimerFunc1 halts the system until the define control step-time is expired. It can be seen that the rest-time is always positive. That is very important in order to obtain deterministic behaviour and a stable system. The rest-time is averagely around 0.025 to 0.05 sec. The step-time is 0.1 sec.

### 8.1.2. Integral control

To overcome the offset-problem of the leading robot, an integral controller is also added to it. The modified model looks like as Figure 8-6.
From Figure 8-7 significant improvement through the integral control can be observed. Compare to Figure 8-1 the real-track is much closer to the simulated track. Especially in the curved area, the improvement is considerable. When looking the track-distance curve of Figure 8-9 the improvement can be quantified: the error is not bigger than 50 mm in the section 6 and 4 and in the section 1 (the big half circle) after about 5 sec the robot basically drives on the track.

The tracing robot behaves similar to the previous test, since its model has not been modified. The state variables are displayed on Figure 8-10.
From Figure 8-11 it can be seen that the rest-time is always positive too.

### 8.2. High speed results

In the high speed tests-series the poles were remain the same on purpose to fully show the effect of the increased speed. Feedback and integral control were tested with the high speed. Tests from [17] have shown that the larger is the velocity, the larger is also the difference between the desired and speed, see Figure 8-12. Therefore in case of high speed control, beside quantization error and curvature error, the controlled speed inaccuracy plays a crucial role too. The transmission delay also has a bigger effect through the higher velocity; with the same transmission delay but a higher speed, the robot moves further away from the reference track in curves.
8.2.1. Feedback control

Figure 8-13 (left) track of leading robot, V=400, solely feedback control; poles: -1.2, -1.201
Figure 8-14 (right) tracing robot track, V=400, with integral control; poles:-1.2, -1.201, -1.0, -1.001, -1.1

Figure 8-13 shows the track of leading robot when it is solely controlled by a feedback controller. The result differs significantly from the simulation result. As far as considered, the transmission delay, curvature error and velocity inaccuracy at high speed are possible reason for this poor result.

Let’s just take the crossing from section 3 (bottom straight line) to section 6 (bottom right quarter circle) as an example to analysed: at the end of the straight line the control still controls the robot to follow the extension of the straight line, even though it is no more the future reference track. With a velocity of 400 mm/s and a step-time of 0.1s that results into a way of 40mm. Additionally to the curvature error, the transmission delay need to be taking into account. If assuming that the transmission is also 0.1s another 40 mm is added. That means when the robot crosses the position x=4000mm the controller has not started to control (to steer the robot to the curve) yet, but not before the robot has further driven on the straight line until the position x=4080mm. From the Figure 8-13 it can be seen that after section 3 the robot starts quite late to steer to the curved, may be around 4100mm.

Once enter the curve the error is kept, because the feedback control is not able to fully

---

8 This figure is taken from [17]
compensate this constant offset. At high velocity, the controlled velocity is not accuracy anyway. Around $V=400\text{mm/s}$ the difference is about 30 mm/s, see Figure 8-12. That can cause even bigger error in the curve.

From Figure 8-15, y-distance, it can be seen that the y-distance (error to reference track), drop to about -200 mm in the sections 6, 4 and 1. In section 1 it remain around this value as disturbance results in feedback control to constant offset.

Figure 8-14, the track of the tracing robot (controlled by integral controller) also differs significantly from this simulated result. However it is much less as the leading robot. The biggest errors occur at the left bottom corner and the right top corner. After entering the section 1 (big half circle), the error gets slightly bigger. That is different in the simulation, where is error the correct gradually. Possible reasons for this misbehaviour are the speed quantization and the inaccuracy at high velocity. In the right top corner (section 4) the curvature error is big, so that the controller has difficulty to follow the reference track. Also the transmission delay is another factor of disturbance.

![Figure 8-15 state variables of leading robot V=400, solely feedback control](image)

From Figure 8-16 it can be seen that is distance to the leading robot mostly is about 500 mm, the set value. Other jumps of the distance to leading robot are caused by the speed quantization of the leading robot and the tracing robot itself, leading robot entering curved sections and also camera calibration inaccuracy at overlapping area.

Figure 8-17 shows that the system is fast enough to process each control step in the giving time-step (0.1s). Averagely there was 0.02 to 0.045 sec. left.
8.2.2. Integral control

Comparing the ideal-, simulated- and real-track from Figure 8-18, it can be seen that the simulated result differs from the reference track. But the real track is even worse. As it was investigated in [17], “the higher the velocity is, the larger the deviation between simulation and reality.” The higher speed introduces larger inaccuracy. However it can also be seen, in case enough time is given, that the robot can be controlled to the reference again. The clear discrepancy at the corner points are momentarily error caused by the curvature change. It can be observed that the robot tends the follow the previous track for a short while, even after having entered into a new section. The idea “improvement through prediction”, see subchapter 7.4.4, exactly addresses this phenomenon. Due to shortage of time the prediction algorithm has not been included. However the simulation shows promising results. Besides the step-time, also the transmission delay can be taken care by the prediction. That will deliver is even better result. It is expected that the robot will drive less swinging, but more foresightedly.
Track Following Tests

Figure 8-18 (left) leading robot track, V=400, with integral control; poles: -1.201, -1.2, -1.3
Figure 8-19 (right) tracing robot track, V=400, with integral control; poles: -1.2, -1.201, -1.0, -1.001, -1.1

In Figure 8-21 the state variable of the tracing robot are displayed. There it can be seen that the distance to the leading robot is much more stable and always close to the desired value: 500mm. Since only the leading robots control model is changed compare to previous test 8.2.1, it can also be said, with a more accurate controlled robot, the tracing benefits too.

Just like the other tests, the timing requirement is met in this case too, see Figure 8-22.

Figure 8-20 state variables of leading robot, V=400, with integral control
All problems, except the inaccuracy in the overlapping area, are not directly related to the usage of multiple robots and multiple cameras. Therefore the perspective of achieving better result through better pole placement, more advance control methods, predictive control are good.
9. Conclusion

In this project two robots have been controlled to follow a static track, which lies in an area observed by two cameras, whereby the second robot is controlled to follow the first robot with certain constant distance. The robots were each attached with two reflexion stickers, so that the cameras mounted on the ceiling are able to detect their positions and orientation. The two cameras were mounted in a way, so their fields of vision are overlapped in the middle. This constellation enlarges the field of operation and enables the robots to cross the two vision fields smoothly. The controllers were determined from the state base description of the robots.

First the basic technical feasibility was tested. These are the cooperation of two cameras and control of two robots from the MATLAB/SIMULINK. After having understood the physical motional behaviour of the robots, state space system models were formulated for both robots. Two different types of controller were realized, they are feedback control and feedback with integral control. The controllers were simulated and verified for their correctness.

In order to adapt the simulation models into the real system, image acquisition and image processing are needed to be incorporated to the system. Before the cameras can be used, they needed to be calibrated, so that data for pixel-to-mm mapping are collected. The general procedure of image processing is the following: starting with image capturing, reduction of image down area of interest, conversion to black/white-image, boundary detection, LSF-fitting for circles, elimination of unreasonable size of spots, merging of doubled captured spots, pixel-to-mm conversion, projection transformation, storing of positions in pixel values into global variable and at last ending with output of positions & orientation of robots in mm.

Besides building control models and establishing the coordination transformation for the desired reference track, major challenges met during the project realization dealt with timing and image-data handling in the overlapping area. Both problems were successfully solved. The timing problem was resolved through reducing image size by estimating the robots positions. Double captured spots in the overlapping area are handled through a merging algorithm. It merges spots from the images, which are physically identical spots.

After having created a stable platform, different control models and tracks were developed and tested in real time. The results were analysed. Apart from extreme conditions, e.g. bad light condition, high robot’s speed, huge initial errors and sudden & large direction change of track, the results were very satisfying. That means starting from positions, which are not on the reference track, the robots find their way to the reference track, and then run on it or close to it. The tracing robot keeps a constant distance, with certain tolerance, to the leading one.

Further issues and topic worth to investigate in future work are among others:

- more advance control
- include predictive method into real-time system
- create more accurate calibration method for the overlapping area
- Include more robots and cameras
- Modify the function in the sense of modularity
- investigate transmission delay
- realize more intelligent formation,
- formation alternation during operation.
- find compensation at high velocity for inaccurate quantization step of the wheel.
- adopt LSF-fitting for other shapes than just circles.
Acknowledgement

I would like to express our great thanks and appreciation to the many people who have helped me through this thesis. First of all I would like to thank our supervisor, Prof. Dr. sc. pol. Thomas Holzhuetter for his helps and instructive suggestions. I really appreciate many valuable comments from Prof. Dr. Holzhuetter for my thesis. Many people in the automatic control laboratory gave me a lot of helps. Mr. Suchan built the mobile robot for me and I could get everything I needed from Mr. Zeyn-Kranz. They also inspired us often during discussion. I am grateful to them.
References


[10]. Thomas Frey, Martin Bossert, Signal- und Systemtheorie, B.G Teubner Verlag, 2004


[13]. Thomas Holzhueter, Zustandsregelung, Lecture notes, Hamburg University of Applied Sciences, 2004


[15]. Claussen, Lecture notes: Control technology, winter semester 2004, Hamburg University of Applied Sciences

[16]. Thomas Holzhueter, Signals and systems, Lecture notes, Hamburg University of Applied Sciences

[17]. Li Fang, Camera based track control for a mobile robot, Master thesis, Hamburg University of Applied Sciences. 2004

[18]. Jean-Yves Bouguet, Camera Calibration Toolbox for MATLAB, June 2005
http://www.vision.caltech.edu/bouguetj/calib_doc/

[19]. Paul Nettle, Sub-Pixel Accuracy, July 2005
http://www.gamedev.net/reference/articles/article661.asp

http://www.efunda.com/math/leastsquares/leastsquares.cfm

[22]. Thomas Holzhüter, Thomas Klinker, Method to solve the nonlinear infinite horizon optimal control problem with application to the track control of a mobile robot, *NDES2005, Sep. 2005*

Figure-Index

Figure 1-1 Basic setup of the project .................................................................2
Figure 2-1 different views of AmigoBot ..............................................................2
Figure 2-2 radio communication through radio modems .......................................2
Figure 2-3 ARIA’s architecture ............................................................................2
Figure 2-4 CCD camera on the ceiling ...............................................................2
Figure 2-5 the visible field of a single camera .....................................................2
Figure 2-6 the whole operation field .................................................................2
Figure 3-1 a) and b), a): real and imaginary values propagate in time, b): projection on the real and time axes, $\sigma = -0.05$, $\omega = 0.5$ ...........................................................2
Figure 3-2 a) and b), $\sigma = 0.05$, $\omega = 0.5$ ......................................................2
Figure 3-3 a) and b), $\sigma = -0.05$, $\omega = 0.0$ ....................................................2
Figure 3-4 Block diagram of a state space described system .................................2
Figure 3-5 Block diagram of state space controller with gain matrix for reference values ...............................................................2
Figure 3-6 Controlled system with stationary disturbances ................................2
Figure 3-7 Block diagram of integral control .......................................................2
Figure 3-8 Block diagram of Integral control with disturbance ............................2
Figure 3-9 Structure of Observer control ...........................................................2
Figure 4-1 Rotational motion of robot ...............................................................2
Figure 4-2 Translational motion of robot ............................................................2
Figure 5-1 complete Distortion Model of the camera ..........................................2
Figure 5-2 Experiment-field captured by the CCD camera ...................................2
Figure 5-3 bilinear interpolation ...........................................................................2
Figure 5-4 Shift and rotation of two cameras field of sign ....................................2
Figure 5-5 Projection Transform due to robot’s height ........................................2
Figure 5-6 a) & b) images of different cameras with spots in upper overlapping area ...............................................................2
Figure 5-7 a) & b) images of different cameras with spots in lower overlapping area ...............................................................2
Figure 5-8 upper overlapping area .......................................................................2
Figure 5-9 lower overlapping area .......................................................................2
Figure 6-1 flow diagram of image processing procedure .....................................2
Figure 6-2 a robot during two time-steps ............................................................2
Figure 6-3 a demonstration of quadtree decomposition ........................................2
Figure 6-4 minimum size of reduced image .......................................................2
Figure 6-5 two reduced images with spots captured doubled ...............................2
Figure 7-1 Interaction flowchart between MATLAB and AmigoBot ....................2
Figure 7-2 Overview of simulation model ..........................................................2
Figure 7-3 Model of leading robot control system .................................................2
Figure 7-4 leading robot model ............................................................................2
Figure 7-5 Model of tracing robot control system .................................................2
Figure 7-6 tracing robot model .............................................................................2
Figure 7-7 a) and b): different reference tracks ...................................................2
Figure 7-8 coordinate transformation of straight line ...........................................2
Figure 7-9 a) and b): coordinate transformation of circle .....................................2
Figure 7-10 definitions of distance on circle .......................................................2
Figure 7-11 snapshot of a robot between two control steps ..................................2
Figure 7-12 a) and b): Track of leading robot, Track of tracing robot ....................2
Figure 7-13 State variables of leading robot .......................................................2
Figure 7-14 State variables of tracing robot without integral part ..........................2
Figure 7-15 a) and b) State variables of tracing robot with integral part .....................................................2
Figure 7-16 controller input of feedback control ..........................................................................................2
Figure 7-17 a) and b) controller input with integral control ..........................................................................2
Figure 7-18 construction of the oval track ....................................................................................................2
Figure 7-19 (left): leading robot track, poles: -0.7, -0.71 .................................................................................2
Figure 7-20 (right): tracing robot track, poles: -0.7, -0.71, -0.61, -0.62, -0.8 ..................................................2

**Figure 7-21 leading robot variables** ........................................................................................................2
Figure 7-22 tracing robot’s state variables .....................................................................................................2
Figure 7-23 a) & b) improvement through pole placement, feedback control for leading robot & tracing robot ..........................................................................................................................2
Figure 7-24 a) & b) improvement through pole placement, integral control for leading & tracing robot .................................................................................................................................................2
Figure 7-25 feedback control V=400 ...........................................................................................................2
Figure 7-26 V=600 integral control prediction used ......................................................................................2
Figure 7-27 a) and b). a)spot in the overlapping area  b)reduced image in the overlapping area ............................................................................................................................................................................................................................................................................................................................................................................................2
Figure 7-28 linear and non-linear control .....................................................................................................2
Figure 7-29 complete real-time system model ..................................................................................................2
Figure 7-30 leading robot control model ........................................................................................................2
Figure 7-31 tracing robot control model .........................................................................................................2
Figure 7-32 result of profiler (without reduction of image size) ....................................................................2
Figure 7-33 result of profiler (with reduction of image size) ..........................................................................2
Figure 8-1 (left) track of leading robot, V=200, solely feedback control; poles: -1.2, -1.201, -1.0, -1.001, -1.1 .................................................................................................................................................................................................................................................................................................................................................2
Figure 8-2 (right) track of tracing robot, V=200, with integral control; poles -1.2, -1.201, -1.0, -1.001, -1.1 .................................................................................................................................................................................................................................................................................................................................................2
Figure 8-3 state variables of leading robot V=200, solely feedback control ....................................................2
Figure 8-4 state variables of tracing robot V=200, with integral control ..........................................................2
Figure 8-5 rest-time during the execution .......................................................................................................2
Figure 8-6 leading robot with integral control ................................................................................................2
Figure 8-7 leading robot track, V=200, with integral control; poles: -1.201, -1.2, -1.3 ..................................................................................................................2
Figure 8-8 tracing robot track, V=200, with integral control; poles:-1.2, -1.201, -1.0, -1.001, -1.1 ............................................................................................................................................................................................................................................................................................................................................................................................2
Figure 8-9 state variables of leading robot, V=200, with integral control .........................................................2
Figure 8-10 state variables of tracing robot, V=200, with integral control .....................................................2
Figure 8-11 rest-time during the execution .....................................................................................................2
Figure 8-12 theoretical and real velocity quantization ......................................................................................2
Figure 8-13 (left) track of leading robot, V=400, solely feedback control; poles: -1.2, -1.201, -1.0, -1.001, -1.1 .................................................................................................................................................................................................................................................................................................................................................2
Figure 8-14 (right) tracing robot track, V=400, with integral control; poles:-1.2, -1.201, -1.0, -1.001, -1.1 .................................................................................................................................................................................................................................................................................................................................................2
Figure 8-15 state variables of leading robot V=400, solely feedback control ......................................................2
Figure 8-16 state variables of tracing robot, V=400, with integral control ........................................................2
Figure 8-17 rest-time during the execution .....................................................................................................2
Figure 8-18 (left) leading robot track, V=400, with integral control; poles: -1.201, -1.2, -1.3, -1.3, -1.0, -1.001, -1.1 .................................................................................................................................................................................................................................................................................................................................................2
Figure 8-19 (right) tracing robot track, V=400, with integral control; poles:-1.2, -1.201, -1.0, -1.001, -1.1 .................................................................................................................................................................................................................................................................................................................................................2
Figure 8-20 state variables of leading robot, V=400, with integral control ........................................................2
Figure 8-21 state variables of tracing robot, V=400, with integral control ........................................................2
Figure 8-22 rest-time during the execution .....................................................................................................2
Appendix A: Alphabetical Function List

Interrelationship between the functions with CameraSfunc3

CameraSfunc3

Function \[ \text{sys}, x0, \text{str}, \text{ts} \] = CameraSfunc3(t, x, u, flag, numOfRob)

File CameraSfunc3.m

Usage S-function
Connect to camera and set initial states in initial module;
Calculate and output robots states in output module
Disconnect to camera in terminate module

Input t - time variable
x - states
u - input
flag - system flag
numOfRob - number of robots in the system

Output sys - system vector, robot states measured from images
x0 - state initialization
str - state ordering strings
ts - inherited sample time

Function Call
defSearchArea – return reduced image
get_pos_pix3 – get robot position in pixel
cutSmallBigSpots – drop spots with unreasonable size
iteraClearNaN – clear a matrix from rows which have NaN elements
nrOfRowOfMartix – return number of row of a matrix
mergeOverlappedSpots – merge spots in the overlapping area
regroup – group spots according from the camera they are captured
swapArray – swap the bigger array to the front
sort_spot – sort spots by their size
reversePrjTrans – do reverse projection transformation
mm2px – map mm-value to it accordance pixel value
### defSearchArea
**Function**
\[ \text{[BWsub, x\_offset, y\_offset]} = \text{defSearchArea}(BW, x, y, \text{camNr}) \]

**File**
defSearchArea .m

**Usage**
Return a reduced image from BW according to the coordinate x,y

**Input**
- BW – a Black-White image
- x,y – coordinate of the centre of the reduced image
- camNr – number of the camera which captured BW

**Output**
- BWsub – the reduced image
- x\_offset – the x-coordinate of the left-bottom corner of BWsub
- y\_offset – the y-coordinate of the left-bottom corner of BWsub

### get_pos_pix3
**Function**
\[ \text{[Xc, Yc, R]} = \text{get_pos_pix3}(I, \text{isDisp}) \]

**File**
get_pos_pix3.m

**Usage**
Return the middle points (x,y) and radius of all circles found in I

**Input**
- I – input image
- isDisp – switch on debug mode

**Output**
- Xc – vector to stores all x-values of middle of found circles
- Yc – vector to stores all y-values of middle of found circles
- R – vector to stores all radius of found circles

**Function Call**
circfit - finds the best fit to an circle for the given set of points

### cutSmallBigSpots
**Function**
\[ \text{retainSpots} = \text{cutSmallBigSpots}(\text{originSpots}) \]

**File**
cutSmallBigSpots.m

**Usage**
return only those spots which’s radius are in a certain range

**Input**
- originSpots – array of spots

**Output**
- retainSpots – array of spots, which fulfil the requirement

### iteraClearNaN
**Function**
\[ \text{[output\_matrix]} = \text{iteraClearNaN}(\text{input\_matrix}) \]

**File**
iteraClearNaN.m

**Usage**
return matrix which is the same as the input matrix, except all rows, that contain NaN element are deleted

**Input**
- Input\_matrix – Input matrix

**Output**
- output\_matrix – output matrix without NaN elements

**Function Call**
iteraClearNaN – recursive function
**nrOfRowOfMartix**

**Function**  
number = nrOfRowOfMartix(inputmatrix)

**File**  
nrOfRowOfMartix.m

**Usage**  
return number of row of the input matrix

**Input**  
inputmatrix – inputmatrix

**Output**  
Number – number of rows of the input matrix

---

**mergeOverlappedSpots**

**Function**  
mergeSpots=mergeOverlappedSpots(p_mm1overlap,p_mm2overlap,mode)

**File**  
mergeOverlappedSpots.m

**Usage**  
Merge physical identical spots from two groups of spots

**Input**  
p_mm1overlap – first group of spots
p_mm2overlap – second group of spots
mode – mode=1: averaging mode; mode=2 drop mode

**Output**  
mergeSpots – all physical unique spots out of p_mm1overlap and p_mm2overlap

**Function Call**  
nrOfRowOfMartix – return number of row of a matrix

---

**regroup**

**Function**  
[grouped1,grouped2] = regroup(SpotsPixelNoNan,col)

**File**  
regroup.m

**Usage**  
Separate the input into two groups according to the certain property: camera number

**Input**  
SpotsPixelNoNan – the group of spots to separate
col – number of column, which holds the property for grouping decision

**Output**  
grouped1 – group of spots which’s camera value = 1

[grouped2 – group of spots which’s camera value = 2

---

**sort_spot**

**Function**  
[sorted_spots] = sort_spot(to_sort_spots)

**File**  
sort_spot.m

**Usage**  
Sort the inputs spots from the biggest radius down to the smallest

**Input**  
to_sort_spots – spots to sort

**Output**  
sorted_spots – sorted spots
## swapArray

<table>
<thead>
<tr>
<th>Function</th>
<th>[p_mm1NoNan, p_mm2NoNan] = swapArray(p_mm1NoNan,p_mm2NoNan)</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>swapArray.m</td>
</tr>
<tr>
<td>Usage</td>
<td>Check which of the two input array is longer and swap the longer array to the front</td>
</tr>
<tr>
<td>Input</td>
<td>p_mm1NoNan – the first input array to check</td>
</tr>
<tr>
<td></td>
<td>p_mm2NoNan – the second input array to check</td>
</tr>
<tr>
<td>Output</td>
<td>p_mm1NoNan – the longer array out of the two input arrays</td>
</tr>
<tr>
<td></td>
<td>p_mm2NoNan – the shorter array out of the two input arrays</td>
</tr>
</tbody>
</table>

## reversePrjTrans

<table>
<thead>
<tr>
<th>Function</th>
<th>[x,y] = reversePrjTrans(px,py,ox,oy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>reversePrjTrans.m</td>
</tr>
<tr>
<td>Usage</td>
<td>Perform reverse projection transformation on a x-/y-coordinate pair; from location on the floor to location on the plane with the height of the robot</td>
</tr>
<tr>
<td>Input</td>
<td>px – x-value of the x-/y-coordinate pair</td>
</tr>
<tr>
<td></td>
<td>py – y-value of the x-/y-coordinate pair</td>
</tr>
<tr>
<td></td>
<td>ox – x-value of the camera’s position</td>
</tr>
<tr>
<td></td>
<td>oy – y-value of the camera’s position</td>
</tr>
<tr>
<td>Output</td>
<td>x – x-value of the transformed reverse projection</td>
</tr>
<tr>
<td></td>
<td>y – y-value of the transformed reverse projection</td>
</tr>
</tbody>
</table>

## mm2px

<table>
<thead>
<tr>
<th>Function</th>
<th>[x_px,y_px,r_px,cam] = mm2px(x_mm,y_mm,r_mm,x_mm_real_pos)</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>mm2px.m</td>
</tr>
<tr>
<td>Usage</td>
<td>Convert mm-value into accordance pixel value</td>
</tr>
<tr>
<td>Input</td>
<td>x_mm –</td>
</tr>
<tr>
<td></td>
<td>y_mm –</td>
</tr>
<tr>
<td></td>
<td>r_mm –</td>
</tr>
<tr>
<td></td>
<td>x_mm_real_pos</td>
</tr>
<tr>
<td>Output</td>
<td>x_px –</td>
</tr>
<tr>
<td></td>
<td>y_px -</td>
</tr>
<tr>
<td></td>
<td>r_px,cam -</td>
</tr>
</tbody>
</table>
Appendix-A

Interrelationship between the functions with `coorTransOvalM2`

**Function** \( y = \text{coorTransOvalM2}(x) \)

**File** `coorTransOvalM2.m`

**Usage** Function to transfer tracing robot’s position and orientation into state variables

**Input**
- \( x \) – vector with length of 7
  - \( x(1) \) – angle of tracing robot
  - \( x(2) \) – y-position of tracing robot
  - \( x(3) \) – x-position of tracing robot
  - \( x(4) \) – angle of leading robot
  - \( x(5) \) – y-position of leading robot
  - \( x(6) \) – x-position of leading robot
  - \( x(7) \) – section of leading robot

**Output**
- \( y \) – vector with length of 3
  - \( y(1) \) – tracing robot orientation to reference track (state variable: theta)
  - \( y(2) \) – tracing robot distance to reference track (state variable: e)
  - \( y(3) \) – tracing robot distance to leading robot (state variable: d)
Function Call

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sectionDecision – determine the current track section of the robot</td>
<td></td>
</tr>
<tr>
<td>coorTranCir1 – perform coordinate transform for circular sections</td>
<td></td>
</tr>
<tr>
<td>coorTranHor1 – perform coordinate transform for horizontal straight line sections</td>
<td></td>
</tr>
<tr>
<td>coorTranVert1 – perform coordinate transform for vertical straight line sections</td>
<td></td>
</tr>
<tr>
<td>course_diff – to get the distance to the leading robot</td>
<td></td>
</tr>
</tbody>
</table>

sectionDecision

Function: \( j = \text{sectionDecision}(x,y) \)

File: sectionDecision.m

Usage: Return the section in the oval track from a \( x,y \)-value pair

Input:
- \( x \) – \( x \)-coordinate of the \( x,y \)-value pair
- \( y \) – \( y \)-coordinate of the \( x,y \)-value pair

Output:
- \( j \) – number of section

coorTranCir1

Function: \([\theta_{\text{out}}, \eta, e_{\text{out}}] = \text{coorTranCir1}(\theta, x, y, r, m_x, m_y)\)

File: coorTranCir1.m

Usage: perform coordinate transform for circular sections to obtain state variables

Input:
- \( \theta \) – absolute angle
- \( x \) – absolute \( x \)-coordinate
- \( y \) – absolute \( y \)-coordinate
- \( r \) – radius of circular track
- \( m_x \) – \( x \)-coordinate of middle of circular track
- \( m_y \) – \( y \)-coordinate of middle of circular track

Output:
- \( \theta_{\text{out}} \) – relative angle to the tangent of the circle
- \( \eta \) – angle of the perpendicular to circle middle
- \( e_{\text{out}} \) – perpendicular distance to the tangent of the circle

coorTranHor1

Function: \([\theta_{\text{out}}, \eta_{\text{out}}, y_{\text{out}}] = \text{coorTranHor1}(\theta, x, y, \text{dir})\)

File: coorTranHor1.m

Usage: perform coordinate transform for horizontal straight line sections to obtain state variables

Input:
- \( \theta \) – absolute angle
- \( x \) – absolute \( x \)-coordinate
- \( y \) – absolute \( y \)-coordinate
- \( \text{dir} \) – direction, 1: from left to right, otherwise: from right to left

Output:
- \( \theta_{\text{out}} \) – angle to the horizontal line
- \( \eta_{\text{out}} \) – for \( \text{dir} = 1 \) \( \eta_{\text{out}} = 0 \), otherwise \( \eta_{\text{out}} = \pi/2 \)
- \( y_{\text{out}} \) – perpendicular distance to horizontal line

Function Call

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normalizeAngle</td>
<td></td>
</tr>
</tbody>
</table>
## coorTranVert1

**Function**

\[ \text{[theta\_out, teta\_out, y\_out]} = \text{coorTranVert1}(\text{theta, x, y, dir}) \]

**File**

coorTranVert1.m

**Usage**

perform coordinate transform for vertical straight line section to obtain state variables

**Input**

- theta – absolute angle
- x – absolute x-coordinate
- y – absolute y-coordinate
- dir – direction, 1: from bottom to top, otherwise: not defined yet

**Output**

- theta\_out – angle to the horizontal line
- teta\_out – for dir =1 teta\_out=0, otherwise teta\_out=\pi/2
- y\_out – perpendicular distance to horizontal line

**Function Call**

normalizeAngle

---

## course_diff

**Function**

d\_out = course_diff(x\_1, y\_1, teta\_1, sec\_1, x\_2, y\_2, teta\_2)

**File**

course_diff.m

**Usage**

to get the distance to the leading robot

**Input**

- x\_1 – absolute x-coordinate of leading robot
- y\_1 – absolute y-coordinate of leading robot
- teta\_1 – angle of the perpendicular to circle middle of leading robot, when in circular section
- sec\_1 – section of leading robot
- x\_2 – absolute x-coordinate of tracing robot
- y\_2 – absolute x-coordinate of tracing robot
- teta\_2 – angle of the perpendicular to circle middle of tracing robot, when in circular section

**Output**

d\_out – distance to leading robot

---

## normalizeAngle

**Function**

\[ \text{phi\_out} = \text{normalizeAngle}(\text{phi}) \]

**File**

normalizeAngle.m

**Usage**

To normalize the input angle into the range from \(-\pi\) to \(+\pi\)

**Input**

- phi – input angle

**Output**

- phi\_out – normalized angle
### coorTransOvalM

**Function**  
\[ y = \text{coorTransOvalM}(x) \]

**File**  
`coorTransOvalM.m`

**Usage**  
Function to transfer leading robot’s position and orientation into state variables

**Input**  
- \[ x \text{ – vector with length of 3} \]
  - \[ x(1) \text{ – angle of leading robot} \]
  - \[ x(2) \text{ – y-position of leading robot} \]
  - \[ x(3) \text{ – x-position of leading robot} \]

**Output**  
- \[ y \text{ – vector with length of 3} \]
  - \[ y(1) \text{ – leading robot orientation to reference track (state variable: theta)} \]
  - \[ y(2) \text{ – leading robot distance to reference track (state variable: e)} \]
  - \[ y(3) \text{ – current section of the leading robot} \]
  - \[ y(4) \text{ – angle of the perpendicular (of leading robot) to circle middle when in circular section} \]

**Function Call**  
- \[ \text{sectionDecision} \text{ – determine the current track section of the robot} \]
- \[ \text{coorTranCir1} \text{ – perform coordinate transform for circular sections} \]
- \[ \text{coorTranHor1} \text{ – perform coordinate transform for horizontal straight line sections} \]
- \[ \text{coorTranVert1} \text{ – perform coordinate transform for vertical straight line sections} \]
Appendix B: script to compute controller matrices

ABCDKNduo.m:

% first Robot: system matrix and control matrix

V = 200;
% b =210;
A = [0,0;
V,0];
B = [2/b;
0];
P = [-.91;
-0.9];

% P = [-0.71;
% -0.7];
K=place (A,B,P);
C = [0,1];
D = [0];
N = -inv((C-D*K)*inv(A-B*K)*B+D);

Ap= [0,0,0;
V,0,0;
0,1,0];
Bp = [2/b;
0;
0];

% Pp = [-2.1;
% -2;
% -0.3];
Pp = [-0.91;%stable pol:
-0.92;
-1.0];

Kp=place (Ap,Bp,Pp);
Cp = [0,1];
Dp = [0];
KpTrim=Kp(1:1,1:2);

Np= -inv((C-D*KpTrim)*inv(A-B*KpTrim)*B+D);
% N2= inv((C2-D2*K2pTrim)*inv(B2*K2pTrim-A2)*B2+D2);

% N=inv(C*inv(B*K-A)*B);
%second Robot: system matrix and control matrix

\( A_2 = \begin{bmatrix} 0,0,0; \\
V,0,0; \\
0,0,0 \end{bmatrix} \);

\( B_2 = \begin{bmatrix} 2/b,0; \\
0,0; \\
0,-1 \end{bmatrix} \);

\( P_2 = [-0.00; \ %\text{this pole must be chosen as -1, why? p21 == -} \\
K_{p23} \\
-0.51; \\
-0.52] \);

\( K_2 = \text{place}(A_2,B_2,P_2); \)

\( C_2 = \begin{bmatrix} 0,1,0; \\
0,0,1 \end{bmatrix} \);

\( D_2 = \begin{bmatrix} 0,0; \\
0,0 \end{bmatrix} \);

\( N_2 = \text{inv}(C_2*\text{inv}(B_2*K_2-A_2)*B_2); \)

\( A_{2p} = \begin{bmatrix} 0,0,0,0,0; \\
V,0,0,0,0; \\
0,0,0,0,0; \\
0,1,0,0,0; \\
0,0,1,0,0 \end{bmatrix} \);

\( B_{2p} = \begin{bmatrix} 2/b,0; \\
0,0; \\
0,-1; \\
0,0; \\
0,0 \end{bmatrix} \);

\( P_{2p} = [-0.5; \ %\text{stable pols:} \\
-0.2; \\
-0.21; \\
-1.3; \\
-1.31]; \)

\( P_{2p} = [-0.9; \ %\text{stable pols:} \\
-0.91; \\
-1.01; \\
-1.02; \)
\[-0.8\];
K2p = place (A2p, B2p, P2p);
K2pTrim = [K2p(1,1), K2p(1,2), K2p(1,3);
           K2p(2,1), K2p(2,2), K2p(2,3)];
N2 = -inv((C2 - D2*K2)*inv(A2 - B2*K2)*B2 + D2);
N2p = -inv((C2 - D2*K2pTrim)*inv(A2 - B2*K2pTrim)*B2 + D2);
%N2 = inv(-(C2*inv(A2))*B2 + D2);
Declaration

I declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that: this master report has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, November 1st 2006