Modelling and Control of a Wheeled Mobile Robot

Group 1036i

February 3rd - June 3rd 2003

Department of Control Engineering
Institute of Electronic Systems
Aalborg University

The Faculty of Engineering and Science
Aalborg University

Institute of Electronic Systems

TITLE:
Modelling and Control of a Wheeled Mobile Robot

PROJECT PERIOD:
IAS10
February 3rd - June 3rd 2003

PROJECT GROUP:
03gr1036i

GROUP MEMBERS:
Hans H. Lausen
Jakob Bro Nielsen
Michael Schioldan Nielsen

SUPERVISORS:
Roozbeh Izadi-Zamanabadi
Michael M. Quottrup

Issues: 10

Pages: 154

Abstract:
The subject of this thesis is "Modelling and Control of a Wheeled Mobile Robot". Based on the Tamiya TXT-1 Xtreme Truck [Tamiya, 2003], an autonomous wheeled mobile robot is designed and implemented. Due to practical problems, only a limited amount of tests were performed on the system.
The works is split into four main parts: Hardware modifications and design, modelling, control and developing an implementation framework. In the modelling, three main models are been derived. A kinematic model, a dynamical model and an odometry model.
Two controllers are designed and simulated. Since the models are nonlinear, the controller design uses nonlinear methods. The first controller is based on feedback linearization using partial linearization. Since partial linearization is applied, only a controller for the kinematics is designed. The second controller is a passivity-based controller. Here both a controller for the kinematics and the dynamics.
The original intent of the project group was to compare the two in practical tests, but this was never achieved due to shortage in time. Instead the controllers are compared using simulations. The comparison reveals similarities of the controllers, and indicates that the passivity-based controller is more robust to modelling errors.
Finally an implementation framework for the onboard computer is designed and implemented and tests are performed on the odometry model.

Preface

This masters thesis is the documentation of the work done by the authors to earn their Masters Degree in Electrical Engineering at Aalborg University. The subject is “Modelling and Control of a Wheeled Mobile Robot”, and the work was performed from February 3^{rd} to June 3^{rd} 2003.

The project is written so satisfy the needs of two parties. The main party is the project supervisors. But since the project is intended to continue in the future, some of the documentation is intended for future project groups continuing the work of this project.

The project group wishes to offer their gratitude to the project supervisors for kind supervision through the project period.

Some practical informations are found on the following page.

Aalborg, June 3^{rd} 2003,

Hans H. Lausen Jakob Bro Nielsen Michael S. Nielsen

Instructions for the Reader

A CD is enclosed to this project. References to files on the CD looks like this: [CD, \dir\filename].

References to litterature are printed as [Lima, 1999] and refers to the Bibliography at the very last pages of the project.

Some of the articles from the Bibliography are available in pdf-format on the enclosed CD. The articles are found in \articles\. The filename is the same as the key used in the Bibliography. Example [Wang, 1988] is wang.pdf.

An overview of the matrices used in the project is given in appendix G on page 143 for easy lookup during reading.

Notational Conventions

Throughout the project matrix notation is used extensively. We refer to the $n \times n$ identity matrix by $I_{n \times n}$. For example:

$$I_{3\times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

As with the identity matrix, a $m \times n$ null matrix consiting of nothing but zeroes is referred to as $0_{m\times n}$:

$$0_{4\times 3} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The solution for x of the matrix equation $Ax = 0$ is called the null-space of the matrix A, and is refered to as:

$$\mathcal{N}(A)$$

Nomenclature

x	The position of the robot on the first axis of the frame $\mathbb{I}$.
y	The position of the robot on the second axis of the frame $\mathbb{I}$.
θ	The orientation of the robot frame $\mathbb{E}$ relative to the inertial frame $\mathbb{I}$.
$\dot{x}$	The velocity of the robot in the direction of the first axis of the frame $\mathbb{I}$.
$\dot{y}$	The velocity of the robot in the direction of the second axis of the frame $\mathbb{I}$.
$\dot{\theta}$	The change in orientation of the robot with respect to the frame $\mathbb{I}$.
q	Generalized coordinate vector describing the configuration of the robot. $q = [\xi \quad \gamma \quad \psi]^T$.
q_{pk}	Generalized coordinates for the posture kinematic model. 5×1.
ξ	Vector describing the posture of the robot. The vector contains x, y and θ.
γ	Vector containing the four steering angles of the robot wheels.
ψ	Vector containing the four wheel angles of the robot.
γ_i	The steering angle of the i'th wheel.
ψ_i	The wheel angle of the i'th wheel (the amount of angle of rotation for the i'th wheel around its angle of rotation).
ζ	Vector containing the changes in steering angles ζ_1 ... ζ_4.
u	Input to the kinematic model containing the linear velocity $\eta(t)$ and the change in steering angles ζ. 3×1.
$\eta(t)$	The linear velocity of the robot. 1×1.
ζ_1	The change in steering angle for wheels 1 and 4. 1×1.
ζ_2	The change in steering angle for wheels 2 and 3. 1×1.
ICR	Instantaneous Center of Rotation. The point the robot will be turning about.
$\mathbb{I}$	The inertial frame in which the robot moves.

$\mathbb{E}$	The robot frame with center in P and rotation θ with respect to the inertial frame $\mathbb{I}$.
O	The origin of the frame $\mathbb{I}$.
P	The origin of the frame $\mathbb{E}$.
W_i	The i'th wheel.
R_w	Radius of the wheels.
α_i	The angle between the first axis of the frame $\mathbb{E}$ and the line from P to the i'th wheel contact point A_i.
CM	Center of mass for the frame of the robot.
d	Distance from P to CM [m].
I_F	Moment of inertia for the frame.
I_W	Moment of inertia for a wheel.
λ	Lagrangian multipliers.
M_F	Mass of the frame for the robot [kg].
m	Mass of a wheel [kg].
ρ	Angle to CM in the $\mathbb{E}$-coordinatesystem [rad].
T_F	Kinetic energy of the frame of the robot.
T_W	Kinetic energy of a wheel.
$r(t)$	A vector containing the reference for the posture and the derivative of the posture. 6×1.
v	Input to the partial linearization. 3×1.
$\acute{v}$	Vector which is related to v.3×1.
z	Vector contaning the posture and the derivative of the posture. 6×6.
$m(q, u)$	Vector used to model the dynamics of the model. 5×1.
V	Storage function used in the passivity-based control design.

Contents

Part I

Introduction

CHAPTER 1

Introduction

In the recent years, as the hardware for mobile robots has been less expensive, the case of having multiple robots cooperating to accomplish tasks has become increasingly interesting for research and industry, with applications to areas such as deep water exploration, building surveillance, transportation of large objects and rescue-operations after large-scale disasters. In short it is called cooperative robotics, and is defined as a population of robots that are behaving as one distributed robot to accomplish tasks that would be difficult (or impossible) for one single robot. The area of cooperative robotics is of special interest to the project group since the group is following the masters program in "Intelligent Autonomous Systems" at Aalborg University.

At Aalborg University projects concerning cooperative robotics has been explored over the last years using two mobile robots, which are decribed in [Schiøler *et al.*, 2001]. The projects are now expanded to involve more than the two original robots, thus making the design and implementation of new robots necessary. It has been decided that the frame for one of the new robots should be based on the Tamiya TXT-1 Xtreme Truck [Tamiya, 2003], see figure 1.1, thereby making the scope of this project the modification of the TXT-1 into an autonomous wheeled mobile robot.

Figure 1.1: *The Tamiya TXT-1 Xtreme Truck that is to be modificated into an autonomous wheeled robot.*

There are a variety of possibilities in the choice of the task for the cooperative robots to accomplish. At this stage no final conclusion has been drawn on the subject of the objectives for the cooperative robots, but one possible scenario could be the task of distributing objects between different stations. That would resemble the situation of AGV's working in a storehouse.

The modification of the TXT-1 into an autonomous wheeled mobile robot is substantial work. The scope of the project is therefore limited to address the problems concerning control of the robot. Areas such as communication, development of a common intelligence system and obstacle detection and avoidance will not be posed.

1.1 Problem Specification

Regardless of the choice of the task for the cooperative robots to accomplish, the modified TXT-1 will need the ability to plan and track a trajectory between two points. The problem specification is therefore:

How can a Tamiya TXT-1 Xtreme Truck be modified in such a way that it is able to generate and track a trajectory between two given points?

To accomplish this, the overall requirements of the robot are that the following two things are possible: Trajectory planning and trajectory tracking. Problems needed to be solved to design and implement trajectory planning and tracking are elaborated in the following.

1.2 Classification of Problems

In order to fulfil the overall requirements stated in the above, various problem areas are adressed. These can be cassified into four main topics:

1. Hardware modifications and design.
2. Modelling.
3. Control and pathplanner design and implementation.
4. Software design and implementation.

Each of the four problem areas will be discussed in the following.

1.2.1 Hardware Modifications and Design

The TXT-1, as it is when bought, only includes a frame, wheels, two motors for propulsion and gearing between the wheels and motors. Motors for steering (called servos), batteries and hardware for control of motors are not included, so these missing parts are to be bought or designed. In other words: This area includes all the missing parts which needs to be bought plus additional hardware which needs to be designed in order for the parts to communicate with each other.

Since a control algorithm for the robot is to be designed, the ability to determine its position is necessary. Once finished the positioning of the robot should be based on a fusion of different sensor informations. In this project the used sensor equipment will be limited to encoders on the wheels, and thereby determine the position through odometry. The parts which needs to be bought or designed are:

- An onboard computer.
- Two servo motors for steering of the front and rear wheels.
- Speed controller to the motors for propulsion.
- Hardware to establish communication between the onboard computer and the motors.

- Hardware to establish communication between the onboard computer and the wheel encoders.
- Batteries to supply motors for propulsion, motors for steering and the onboard computer.

1.2.2 Modelling

In order to be able to implement one or more controllers for the robot, a model of the robot is needed. It is decided to develop both a kinematic and dynamical model for that purpose.

Having designed a controller using the kinematic and dynamical model, the outputs from the controller are the steering angles for the servos and the torque to be applied to the DC-motor for propulsion. A model of the relationship between the steering angles and the signal to be applied the servo is therefore needed, as well as a DC-motor model describing the relationship between the torque and the signal to be applied the DC-motor is needed.

The posture of the robot must be known at all times in order for the controller to work properly. For that purpose an odometry model is derived, which is capable of determining the change in posture using measurements from encoders.

1.2.3 Control and Pathplanner Design and Implementation

The pathplanner is responsible for planning the trajectory to be tracked by the robot controller. The trajectory is timevarying. This means that for each posture on the trajectory, there is a time defined. At this time, the robot should be at that posture.

The control design for trajectory tracking will be based on the models described in the above. Since the models will be nonlinear, the controllers will be designed using nonlinear methods.

1.2.4 Software Design and Implementation

Two software structures needs to be designed. The first software structure is to be common for all the robots in the cooperative robots project. Or in other words this software structure must be platform independent. This software structure will not be adressed in this project, since it easily amounts to a project in itself. The software structure that will be adressed here, is the second software structure which is platform dependent. In this software structure the modeling and control algorithm is implemented. In other words an implementation framework is designed, which makes implementation of the pathplanner and the controller possible. This includes interfacing with the servos and DC-motor and implementation of the odometry model.

1.3 Project Outline

Having elaborated the problems to be adressed in order to be able to solve the problem specification, the outline of the rest of the project follows here.

The original intention of the project group was to implement and compare two different controllers on the robot and to implement a pathplanner. Due to practical problems, it wasn't accomplished to implement the two controllers on the robot within the time limits of the project. Thus

two controllers are designed, but unfortunately none of them was implemented in the robot, and instead simulations of the two controllers are compared. Furthermore, due to time-shortage, only

The robot is equipped with servos at the front and rear wheel pair, but due to the odometry model derived later, it is only possible to use the servo at the front wheel pair. The problem is that for a odometry model for four-wheel steering to be developed, the steering angles of the front and rear wheel pair must be known. That is not the case in the current implementation of the robot. The control signal applied to the servos is known, and thereby it is known, what the steering angles were intended to be. But due to large friction between the tires and ground, the intended steering angles are not the same as the actual steering angles, and using these angles as steering angles for the odometry model would introduce an error. To avoid this problem the odometry model is developed with the rear wheel pair fixed in orientation. With this simplified odometry model, it is a requirement of the controllers that only front wheel steering is used. Although for future use of the controllers, they are designed using both front and rear wheel steering.

The project is split into six main parts: Introduction, Modeling, Control, Implementation, Conclusion and Appendix. In part I after this introduction, a description of the robot and the implementations made by the group is described in chapter 2. The general definitions of the project is in chapter 3. In part II on modelling the constraints for the robot are defined in chapter 4, and the kinematic model is derived in chapter 5. A dynamic model is derived in chapter 6, and the DC-motor and servo-models are presented in chapter 7 and finally the odometry model is derived in chapter 8. In part III the controllers are derived. 9, and next a passivity based controller is designed in chapter 10. The simple pathplanner is also described in the chapters concerning the controllers. In chapter 11 the two controllers are compared using simulations. In part IV the implementation of the software is described in chapter 12 and some tests are documented in chapter 13. The conclusion and recommendations for future work are offered in chapter 14.

The overall structure of the project is illustrated in figure 1.2.

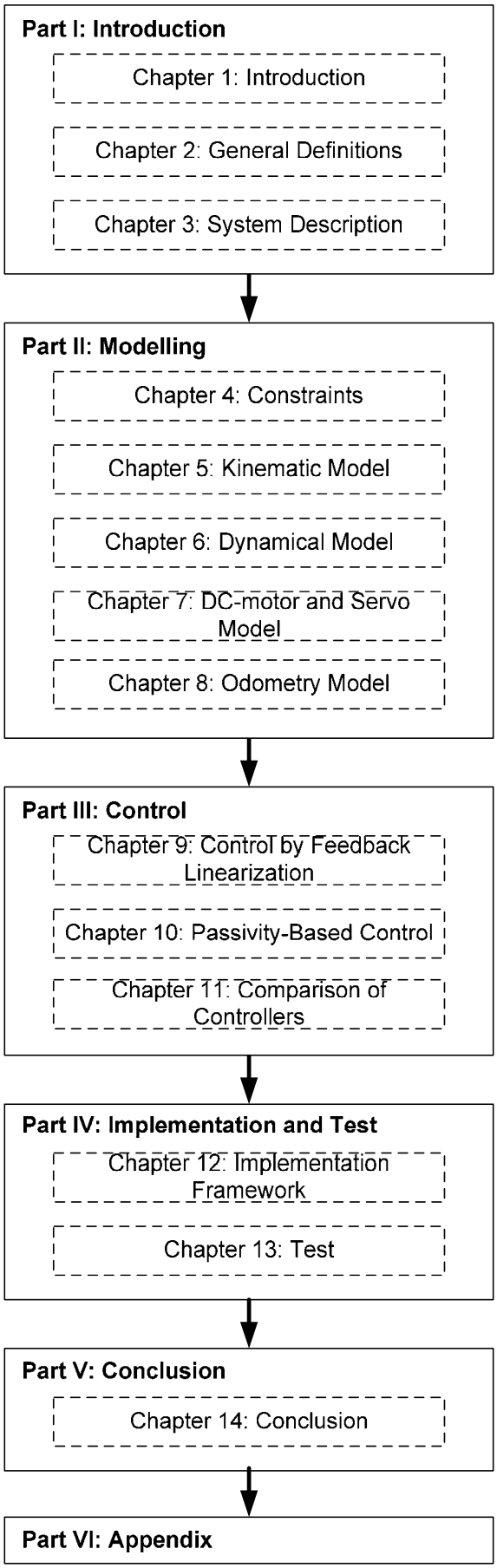

Figure 1.2: *The overall structure of the project.*

CHAPTER 2

System Description

This chapter describes the modifications needed to transform the TXT-1 truck from an ordinary model-car into a wheeled mobile robot. First a description is given of the starting point, how the TXT-1 truck was originally equipped. This is followed by the actually hardware modifications.

2.1 Original Equipment

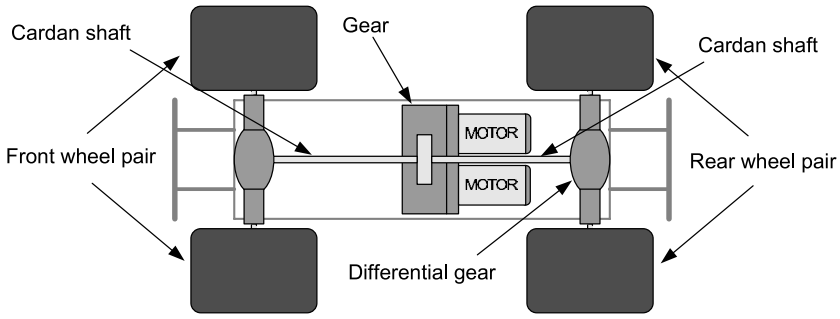

Figure 2.1: *A sketch showing the frame of the TXT-1 truck.*

Figure 2.1 shows the frame of the TXT-1 truck, as it appears when received from the manufacturer. The truck is delivered with and propelled by two DC-motors of Johnson type 540 [CD, \Datasheets\Johnson_HC683G.pdf]. The motors are run in parallel and are attached to a gear box. Two cardan shafts are attached to this gear box. One shaft drives the front wheel pair and the other drives the rear wheel pair. Therefore the torque provided by the motors is equally distributed between front and rear wheel pair. The cardan shafts are connected to differential gears. When the car makes a turn, the distance the left and right wheel has to drive is not the same, thus the left and right wheel has to rotate with different speeds. This distribution of speed among the wheels are maintained by the differential gears.

The car is able to turn both the front and rear wheel pair. Naturally this turning is limited. In figure 2.2 a top view picture of one of the wheel houses is seen, and the picture shows that the turning is limited within $\pm 20^\circ = \pm \frac{pi}{9} rad$.

2.2 Overview of Hardware Modifications

Figure 2.3 shows the structure of the parts in the hardware modification. As mentioned only the DC-motors were available from the beginning. The remaining items were either bought or

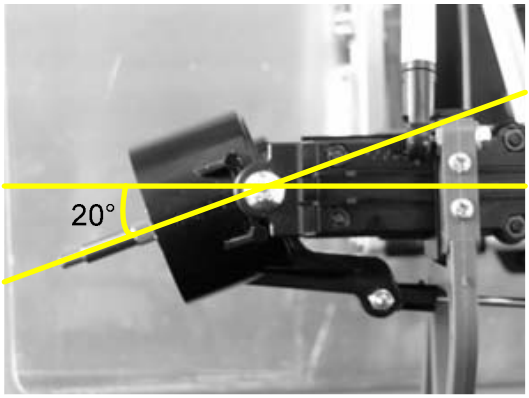

Figure 2.2: *The maximum steering angle for one wheel is* 20° *to each side.*

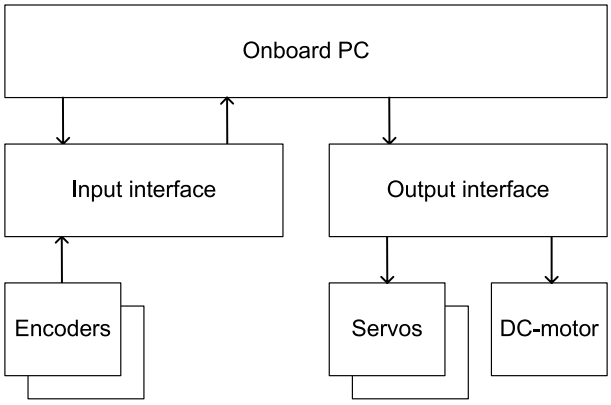

Figure 2.3: *The structure of the electrical hardware that was bought and implemented.*

designed by the project group. In the top of figure 2.3 is the onboard computer. This is where the controller is eventually implemented. The onboard computer maintains communication between two parts: The input and output interface. Both input and output interfaces were designed or bought. Below these interfaces, the actuators and encoders are seen. The motors (servos) to make the car turn had to be bought. The encoders at the wheels to keep track of the wheels movement had to be designed or bought as well. In the following sections a more detailed description is given of the parts just mentioned.

2.3 Encoders

The task of the encoders is to keep track of the rotation of the wheels. The requirements for the encoders are elaborated in the following.

2.3.1 Requirements

The encoder must be able to register the rotation of either the DC-motor axle, cardan shaft or the wheel itself. The size and resolution of the encoder must be compatible with the size of the truck and the desired precision of the trucks movement. Furthermore the encoders has to be robust and

be able to cope with disturbances (shocks) when the car is in motion. Robustness against dirt and dust is also preferable. The demands for the encoders can be summed up to:

- Register rotation of motor axle, cardan shaft or wheel itself.
- Suitable resolution and size to meet the demands of precision in the trucks movement.
- Robust against small bumps and dust from the surroundings.

2.3.2 Design and Implementation

The idea of mounting an encoder on either the motor or the cardan shaft was rejected due to a slackness or backlash between the gear attached to the motor and the differential gear at the wheels. This backlash causes a slip between the rotation of the motor axle and the rotation of the wheel. Therefore it was decided to mount the encoders directly at the wheels.

The choice of encoder resulted in the HEDR-8000 optical encoder from Agilent Technologies ([CD, \Datasheets\HEDR-8xxx.pdf]). This encoder uses reflective technology to sense rotary or linear position. The sensor consists of an LED light source and a photodetector IC in a single SO-8 surface mount package. The sensor is used with either a reflective codewheel or codestrip. In this project a codewheel of type HEDR-5120 is used.

The advantage of the chosen encoder is first and foremost its size. It is very small, thus implementable directly behind the wheel, attached to a piece of print inside the wheel housing. Figure 2.4 shows how the HEDR-8000 encoder is mounted inside the wheel house. The code

Figure 2.4: *HEDR-8000 in the wheel house attached to a piece of print.*

Figure 2.5: *Codewheel glued on the back of the wheel rim.*

wheel is glued to the back of the wheel rim. This is depicted in figure 2.5. When the encoder is placed correctly with high precision, the encoder is able to detect the reflective/nonreflective pattern of the codewheel. When the wheel rotates this results in two signals shown on figure 2.6. The signals, A and B are delayed 180 electrical degrees relative to each other. These two channels are used to detect whether the wheel is rotating clockwise or counter clockwise. This interpretation is done on the basis of which signal, channel A or B, leads the other.

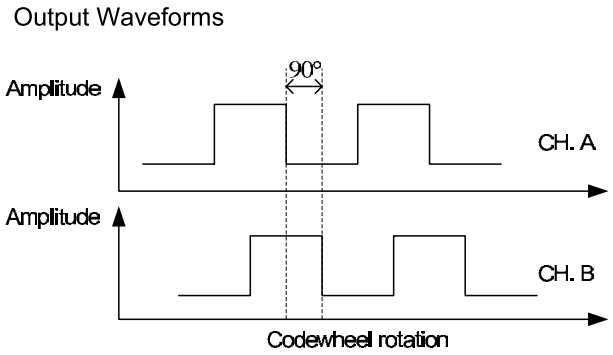

Figure 2.6: *When the wheel rotates and the encoder and codewheel is placed correctly relative to each other, the signals appears as depicted above. Two channels, A and B, delayed 180 electrical degrees relatively.*

2.4 Input Interface

The raw signals from the two encoders is useless for the PC. They will have to be translated into understandable data, which then can be transmitted up to the PC. Both translation and transmission to the PC is done by the input interface.

2.4.1 Requirements

First of all the input interface has to retrieve the signals from the encoders and convert them into useful data for the PC. The encoder signals has to be converted into a number that tells how much the wheel has rotated. Furthermore the input interface has to send the information about the wheels position to the PC. The requirements for the interface are:

- Decode signals from encoders.
- When requested, send information about wheels position to PC.

2.4.2 Design and Implementation

For decoding the signals from the encoders, it was decided to use the HCTL-2020 qudrature decoder/counter from Agilent Technologies ([CD, \Datasheets\HCTL-2000.pdf]). This device can decode the quadrature signals from the encoders, and has a 16-bit up/down counter. In addition it has a parallel bus interface. Due to this bus interface it was decided to add a microprocessor to the input interface. The chosen mocroprocessor is PIC16F877 from Microchip ([CD, \Datasheets\30292c.pdf]). This microprocessor shall both handle the communication to the HCTL-2020 as well as the communication to the PC. The communication to the PC is done from the microprocessor to a COM port on the PC using the RS-232 standard. The amplitude of the signals in the RS-232 standard uses voltages ranging between $-15V$ and $+15V$ and the signals from and to the microprocessor uses the TTL standard $0V$ and $+5V$. Therefore a driver/receiver is neccesary to convert the signals between the two standards. The MAX232 device ([CD, \Datasheets\MAX232,232I.pdf]) was chosen to perform this conversion. Figure 2.7 shows the path of communication from the encoders to the PC.

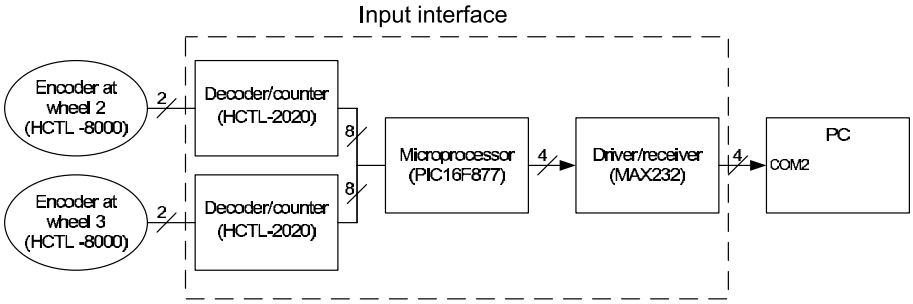

Figure 2.7: *The path of communication between the two encoders at the wheels to the PC.*

The microprocessor in figure 2.7 is the intelligent device in the input interface and software had to be written to make it work as the connecting link between the encoders and the PC. Instead of programming the microprocessor in assembler, the software was written in C. This C-code was then compiled into a hex-file using a PIC C-compiler named CC5X. The CC5X is available in a free limited edition, and this compiler was used with MPLAB. MPLAB is a development software that besides being an editor also gives the programmer the possibility of testing the code before implementing it in the microprocessor. MPLAB is free and can be downloaded at the Microchip homepage.

Usually the generated code is written into the microprocessors memory using an EEPROM writer. Instead of this approach the software written by this project group was downloaded to the microprocessor using a bootloader. In short the bootlader is a piece of software that is written to the microprocessors memory in the usual way. The bootloader makes it possible to download new usercode (i.e. written by the project group) through a serial connection an unlimited number of times, without conflicting with the bootloader code. This has been a fast and reliable way of programming the microprocessor. The bootloader is explained in greater detail in appendix H on page 149.

In figure 2.8 the software flow in the microprocessor is seen. A further description of the code will not be given in this report, but the code can be found in [CD, \Code\PIC16F877].

2.5 Servos

The servos task is to turn the wheels of the truck when requested from the controller. In the following the requirements for the servos are listed.

2.5.1 Requirements

- Turn the wheels when requested from the controller.
- Fast and reliable response.

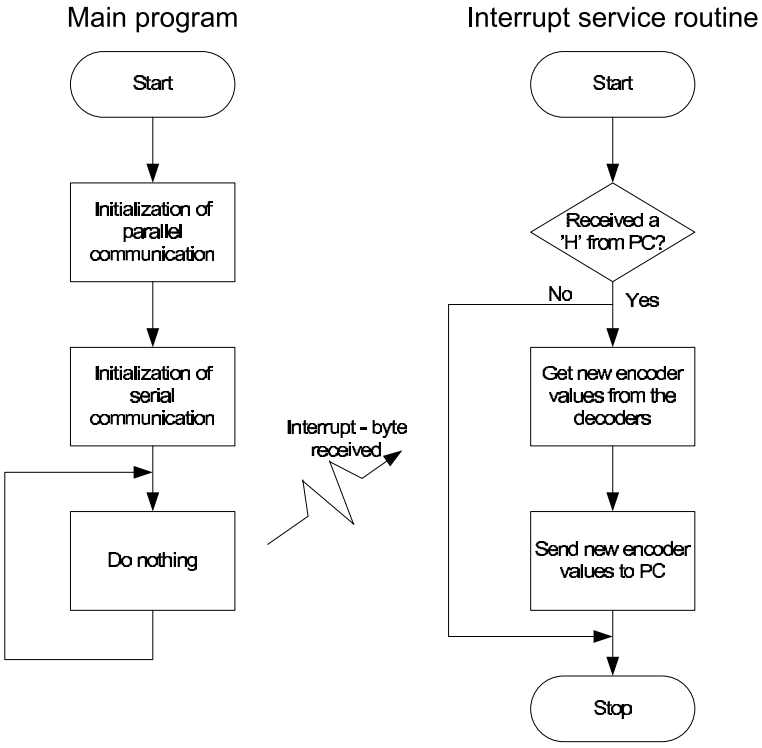

Figure 2.8: *The main software flow in the microprocessor.*

2.5.2 Design and Implementation

Tamiya, the producer of the TXT-1 truck, has some recommendations in order of selecting servos. These recommendations though is not compatible with the needs in this project. The modifications of the truck have resulted in a considerable weight increase. The servos needed, have to be more powerful than the ones suggested by Tamyia. Two servos of type Hitec HS-705MG ([CD, \Datasheets\HS-705MG.pdf]) was bought. These units are high torque servos which are controlled with a $3V$ to $5V$ peak to peak square wave pulse signal. The power supplied to the servos must be within the range 4.8V to 6V.

2.6 Output Interface

Both the DC-motors and the servos is controlled by pulse width modulated (PWM) signals. Therefore a link between the PC and the actuators is required. This link is the output interface, which will be explained in the following.

2.6.1 Requirements

The main task of the output interface is to convert data signals from the PC into PWM powered signals delivered to the DC-motors and the servos. The data signals from the PC are values calculated from the controller that corresponds to the steering angle on the wheels and the desired output of the DC-motors. Like the input interface, it was decided that the communication between the PC and output interface should be done through a RS-232 based serial connection. The requirements

for the output interface are:

- Serial communication with PC, based on RS-232 standard.
- Convert data signals from PC into PWM powered signals to DC-motors and servos.

2.6.2 Design and Implementation

Instead of designing the hardware in the output interface from scratch, it was chosen to buy the parts needed. To make the communication from the PC possible the Mini SSC Serial Controller ([CD, \Datasheets\ssc2_mnl.pdf]) was chosen. Input to this board, from now on called the SCC-controller, is a three byte command written through a serial connection. Output is up to eight PWM signals. The two servos are connected directly to this board. It is not possible to connect the DC-motors directly to the SSC-controller, due to higher power consumption than the SCC-controller is able to handle. Instead a speed-controller was bought to supply the DC-motors with the needed power. The speed-controller is connected to the SCC-controller. Figure 2.9 shows the path of communication from the PC to the motors and servos.

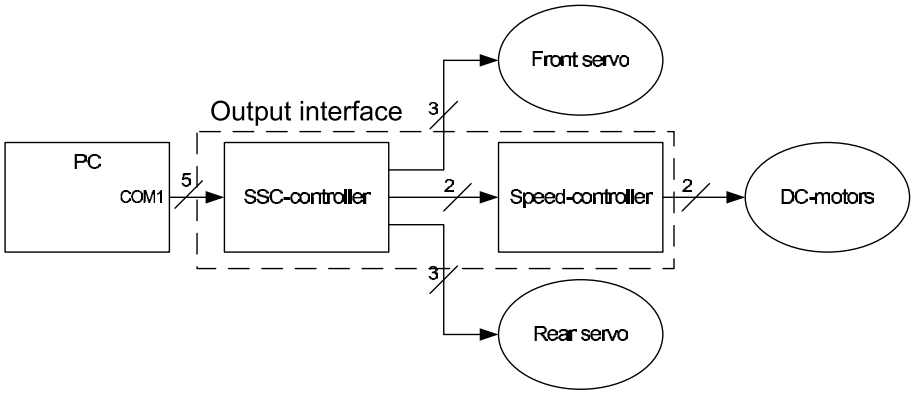

Figure 2.9: *The path of communication in the output interface from the PC to the actuators.*

2.7 Onboard Computer

The final part part is the PC. Several things were taken into account before the PC was purchased. These considerations is presented in the following section.

2.8 Requirements

First of all the size of the computer is of great importance, due to the limited space available on the robot. The power consumption of the computer is significant too, because it is supplied by batteries. The power consumption is tightly related to the CPU size, but still a powerful processor is needed to make it compatible with future projects as well. These future project most likely involving processor demanding image processing. Furthermore it is important that the PC offers

possibilities for extensions in the need of using different cards i.e. using PCI or PC-104 technologies. Last of all, the economic aspects is of great importance. All the above considerations leads to the following requirements:

- Physical size shall be compatible with the space available on the robot.
- Low power consumption.
- Sufficient CPU power (i.e. also to be used in future projects, involving image processing).
- Room for extensions (PCI and PC-104 slots).
- Affordable within the budget for this project.

2.8.1 Design and Implementation

Three types of computers were candidates to be used on the robot. These were a PC-104, a labtop and a single board computer. Advantages and disadvantages were weighted. From earlier work and experiences with mobile robots the project group suggested the use of a labtop. Due to economic aspects this proposal was rejected. The labtop itself is expensive to buy, and in addition to the computer itself comes the different devices to interface the accessories, i.e. encoders and actuators. The single board computer was selected, mainly because of greater CPU power than the one supplied in the PC-104. Besides, the single board computer has easy access of using regular PC-interface cards, which in general is cheaper than the accessories available for a PC-104. The chosen PC was the LS-561 ([CD, \Datasheets\LS-561-11.pdf]), a single board computer with a VIA C3 1000 MHz lowpower consumption processor. The boards main features are:

- VIA C3 1000 MHz processor.
- 128 MB RAM.
- 20 GB harddisk 2.5”.
- 5.25” EBX socket 370 PC133 littleboard.
- 4xAGP 3D SVGA.
- Intel PRO/100+ LAN.
- PC/104+ interfaces.
- 4 COM/ LPT/2 USB.

The single board computer is not embedded in a box when delivered. Some kind of protection is needed when the board is attached to a wheeled mobile robot. Therefore the computer and harddisk was built in a plastic box. The harddisk mounted in an anti vibration kit. Furthermore parts of the input and output interfaces was built in the box. (Måske et bette billede af vores fine kasse her OG EN LILLE AFRUNDNING)

CHAPTER 3

General Definitions

In this chapter the definitions for the robot will be stated. The definitions will be used in future modeling and control design. The definitions includes coordinate frames, coordinate vectors and notation.

3.1 Coordinate Frames

To describe the posture (position and orientation) of the robot, two coordinate frames $\mathbb{I}$ and $\mathbb{E}$ are defined. $\mathbb{I}$ is the inertial frame, in which the robot is moving. Attached to the robot center is the frame $\mathbb{E}$. The frame $\mathbb{E}$ has origin in the center of the robot (the point P), and is rotated θ with respect to frame $\mathbb{I}$. The two frames are illustrated in figure 3.1.

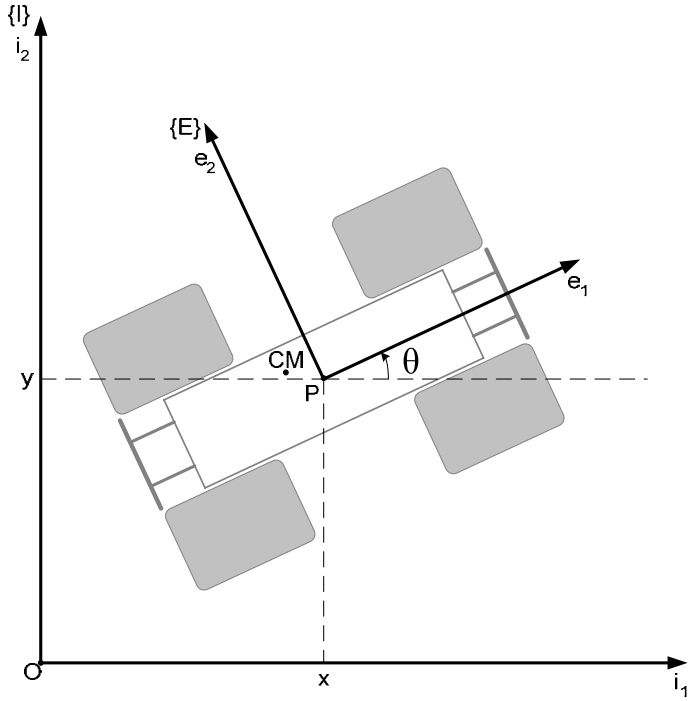

Figure 3.1: *The frames $\mathbb{I}$ and $\mathbb{E}$. The coordinates x and y decribes the position of the center of the robot with respect to $\mathbb{I}$, and θ describes the orientation of the robot with respect to $\mathbb{I}$.*

3.2 Rotation Between Frames

A point x in frame $\mathbb{I}$ is written as ${}^{\mathbb{I}}x$ and ${}^{\mathbb{E}}x$ is the notation for the same point in frame $\mathbb{E}$. The rotation of a point in frame $\mathbb{I}$ to $\mathbb{E}$ is performed with the rotation matrix $R(\theta)$

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

The rotation from $\mathbb{E}$ to $\mathbb{I}$ is performed using $R^{-1}(\theta) = R^T(\theta)$.

3.3 Coordinate Vector

The posture of the robot is described entirely by the coordinate vector ξ, which is defined as

$$\xi = \begin{bmatrix} x & y & \theta \end{bmatrix}^T \tag{3.2}$$

This is the decription of the frame $\mathbb{E}$ with respect to the frame $\mathbb{I}$.

3.4 Wheels

The wheels are numbered $W_i, i = 1..4$, and the contact point of each wheel are called A_i. The contact point is defined as a single point, where the wheel and the surface meet.

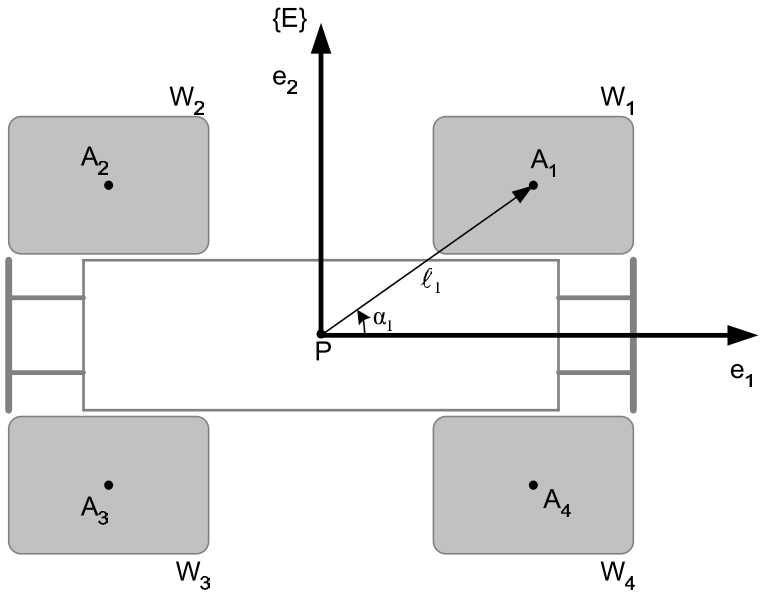

Figure 3.2: *The frame $\mathbb{E}$ with origin in the center of the robot.*

3.4.1 Position

The position of each of the four wheels on the robot are described in $\mathbb{E}$ by the polar coordinate (l_i, α_i), where the length l_i is the distance from the origin of $\mathbb{E}$ to the contact point. α_i is the angle with respect to the first axis of $\mathbb{E}$.

3.4.2 Steering Angle

To describe the steering angles of the wheels with respect to the first axis of the frame $\mathbb{E}$ the angles γ_i are defined, as illustrated in figure 3.3. Since the front wheel pair as well as the back wheel pair are connected mechanically by a rod, the two wheels in front as well as the two wheels in the back have the same steering angles:

$$\gamma_1 = \gamma_4 \ \wedge \ \gamma_2 = \gamma_3 \tag{3.3}$$

Thus it is only necessary to consider the two angles γ_1 and γ_2 to describe the steering angles of the robot. These two angles are members of the vector γ:

$$\gamma = \begin{bmatrix} \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 \end{bmatrix}^T \tag{3.4}$$

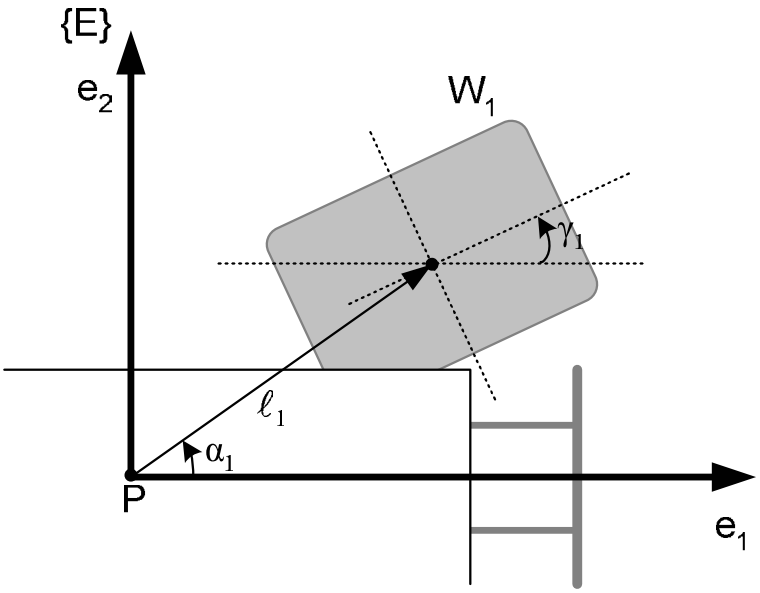

Figure 3.3: *Wheel 1 seen from above. The steering angle of the wheel is described by the angle γ. Here only wheel 1 is illustrated.*

3.4.3 Wheel Angle

To describe the rotation of each wheel in the wheel plane around the rotation axis of the wheel, the angle ψ_i is defined as illustrated in figure 3.4. The vector ψ is defined to contain the four angles ψ_i:

$$\psi = \begin{bmatrix} \psi_1 & \psi_2 & \psi_3 & \psi_4 \end{bmatrix}^T \tag{3.5}$$

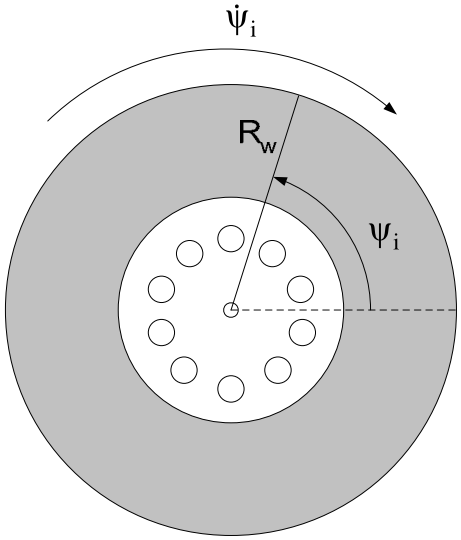

Figure 3.4: *A wheel on the robot seen from the side.*

3.5 Generalized Coordinate Vector

The generalized coordinates of the robot is defined as the vector q

$$q = \begin{bmatrix} \xi \\ \gamma \\ \psi \end{bmatrix} \tag{3.6}$$

The generalized coordinate vector describes the full configuration of the robot including posture and steering and wheel angles.

Part II

Modelling

CHAPTER 4

Constraints

In order to be able to simulate and design a controller to the robot, a mathematical model of the robot is needed. The overall model can be split into two main parts: A kinematic and a dynamic model. The dynamic model describes the relation between the applied torques (from the motors) and the resulting changes in velocities. These velocities are translated into displacement by the kinematic model. The development of the two models and a model for the motor will be the subject of the following.

The robot used in this project is a low cost prototype, and therefore presents the non ideal conditions of electromechanical systems, namely friction, gear backlash, wheel slippage and actuator saturation. Taking these non ideal conditions into account, the modeling of the robot is based on some assumptions. It is assumed that the robot is constructed as a rigid frame. The wheels mounted on the robot are considered nondeformable and perfectly round. Furthermore they only have contact with the ground in a point (the contact point), and since the robot is to be used for indoor experiments, the robot is assumed to be moving on a horizontal plane at all times.

The method used to derive both the kinematic and dynamic models are described in [Champion, 1996]. Both models are derived using the general pure rolling without slipping-contraint, meaning that no slip and no lateral movement is allowed for any of the wheels. [Further description of the method used is needed here]. The method has been used succesfully in the modeling of a robot for agricultural use [Sørensen, 2001], [Tarozzi, 2002].

In general two types of motion for the robot are considered: Straight and turning. During straight motion all four wheels are parallel with the robot frame, and the wheels are covering the same distance (the torques applied to each wheel are all equal). If the robot is turning, the setup is not that simple. In order to make the robot turn and comply with the pure rolling and no slipping conditions, the robot has to turn around an instantanous center of rotation (in the following: ICR) according to [Champion, 1996], see figure 4.1. Due to the construction of the wheel suspension the two front wheels as well as the two rear wheels have the same steering angle. This means that it is not possible to have an ICR, and thereby it is not possible to have turning motion without violating the constraints. It is though assumed that the error in steering angle (and ICR) will be neglectable. The error of ICR is illustrated in figure 4.2.

Two constraints are imposed on each of the four wheels: They are not allowed to slip during motion, and no lateral movement is allowed. The constraints are described in detail in the following.

Rolling without slipping A wheel slips if the velocity of the contact point A_i in the wheel plane

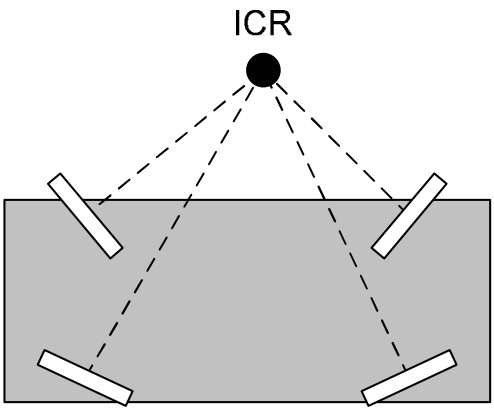

Figure 4.1: *The concept of an instantaneaus center of rotation (ICR). The axles of rotation of all four wheels intersect in the ICR.*

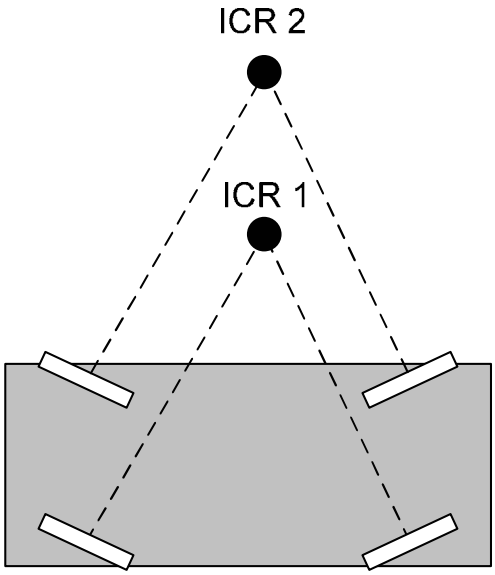

Figure 4.2: *An ICR is not possible for the robot in this project, since the wheels in both the front and rear wheel pair have the same steering angles.*

is different from the linear tangential velocity of the wheel at the contact point.

No lateral movement Lateral movement occurs when the sum of velocity components orthogonal to the wheel plane in the contact point is different from zero.

In the following the two constraints will be derived for one wheel. Afterwards the constraints will be generalized to all four wheels.

4.1 Rolling without slipping

As stated in the previous, the contact between the wheel and the ground is supposed to satisfy the rolling without slipping condition. This means that the velocity of the contact point A should be equal to the linear velocity of the wheel. The situation is illustrated in figure 4.3 and can be expressed as

$$R_w \dot{\psi}_i = v_A \tag{4.1}$$

where R_w is the radius of the wheel, $\dot{\psi}$ the angular velocity of the wheel and v_A is the velocity of the contact point. The velocity components are illustrated in figure 4.4. The coordinate of the contact point A is (a_x, a_y, a_z), with a_z equal to zero, since only the x and y dimension are of interest. Calculations of the velocity of A yields:

$$v_A = \cos(\theta + \gamma_i)^{\mathbb{I}} \dot{a}_x + \sin(\theta + \gamma_i)^{\mathbb{I}} \dot{a}_y \tag{4.2}$$

and inserting equation 4.2 in 4.1 results in

$$\cos(\theta + \gamma_i)^{\mathbb{I}} \dot{a}_x + \sin(\theta + \gamma_i)^{\mathbb{I}} \dot{a}_y = R_w \dot{\psi}_i \tag{4.3}$$

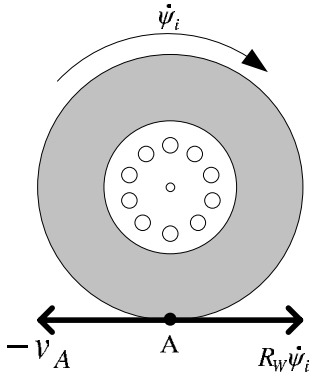

Figure 4.3: *Wheel with linear velocity $\dot{\psi}_i R_W$.*

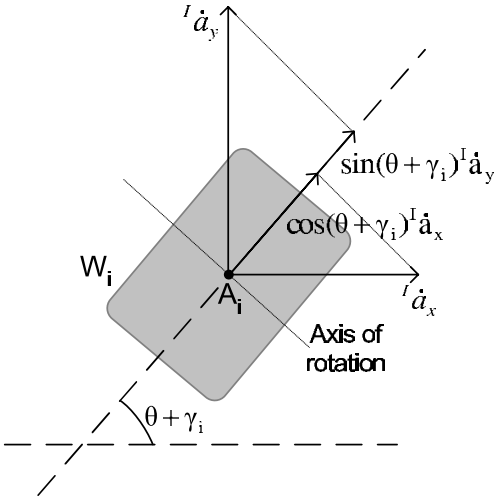

Figure 4.4: *No slip condition.*

To reduce this expression, we seek expressions for ${}^{\mathbb{I}}\dot{a}_x$ and ${}^{\mathbb{I}}\dot{a}_y$ in the following. The position of the contact point A in the frame E can be expressed using its polar coordinates as

$$\begin{bmatrix} {}^{\mathbb{E}}a_x \\ {}^{\mathbb{E}}a_y \\ {}^{\mathbb{E}}a_z \end{bmatrix} = \begin{bmatrix} \ell_i \cos\alpha_i \\ \ell_i \sin\alpha_i \\ 0 \end{bmatrix} \tag{4.4}$$

The position of the contact point A with respect to the inertial frame $\mathbb{I}$ can be determined as

$$\begin{aligned} \begin{bmatrix} {}^{\mathbb{I}}a_x \\ {}^{\mathbb{I}}a_y \\ {}^{\mathbb{I}}a_z \end{bmatrix} &= R^T(\theta) \begin{bmatrix} {}^{\mathbb{E}}a_x \\ {}^{\mathbb{E}}a_y \\ {}^{\mathbb{E}}a_z \end{bmatrix} + \begin{bmatrix} {}^{\mathbb{I}}x \\ {}^{\mathbb{I}}y \\ {}^{\mathbb{I}}z \end{bmatrix} = R^T(\theta) \begin{bmatrix} \ell_i \cos\alpha_i + {}^{\mathbb{I}}x \\ \ell_i \sin\alpha_i + {}^{\mathbb{I}}y \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} x + \ell_i(\cos\theta\cos\alpha_i - \sin\theta\sin\alpha_i) \\ y + \ell_i(\sin\theta\cos\alpha_i + \cos\theta\sin\alpha_i) \\ 0 \end{bmatrix} = \begin{bmatrix} x + \ell_i\cos(\theta+\alpha_i) \\ y + \ell_i\sin(\theta+\alpha_i) \\ 0 \end{bmatrix} \end{aligned} \tag{4.5}$$

Taking the derivatives yields

$${}^{\mathbb{I}}\dot{a}_x = \dot{x} - \dot{\theta}\ell_i \sin(\theta+\alpha_i) \tag{4.6}$$

$${}^{\mathbb{I}}\dot{a}_y = \dot{y} + \dot{\theta}\ell_i \cos(\theta+\alpha_i) \tag{4.7}$$

Inserting the derivatives from equations 4.6 and 4.7 in 4.3 and applying geometrical relations yields

$$\begin{aligned} &\cos(\theta+\gamma_i)\dot{x} + \sin(\theta+\gamma_i)\dot{y} + \dot{\theta}[\sin(\theta+\gamma_i)\cos(\theta+\alpha_i) - \\ &\sin(\theta+\alpha_i)\cos(\theta+\gamma_i)]\ell_i - R_w\dot{\psi}_i = 0 \\ &\Downarrow \\ &\dot{x}(\cos\theta\cos\gamma_i - \sin\theta\sin\gamma_i) + \\ &\dot{y}(\sin\theta\cos\gamma_i + \cos\theta\sin\gamma_i) + \\ &\dot{\theta}\ell_i \sin(\gamma_i - \alpha_i) - R_w\dot{\psi}_i = 0 \end{aligned} \tag{4.8}$$

Rearranging this expression using $R(\theta)$ and $\dot{\xi}$ yields the following matrix-equation, which expresses the no lateral movement-constraint for one wheel.

$$[\cos(\gamma_i) \quad \sin(\gamma_i) \quad \ell_i \sin(\gamma_i - \alpha_i)] R(\theta)\dot{\xi} - R_w \dot{\psi}_i = 0 \tag{4.9}$$

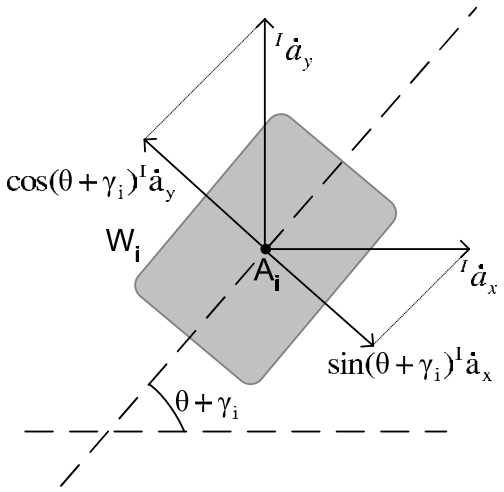

Figure 4.5: *No lateral movement.*

4.2 No Lateral Movement

To ensure no lateral movement of a wheel, the sum of the two velocity components orthogonal to the wheel plane must be zero, see figure 4.5. This can be expressed as

$$\sin(\theta + \gamma_i)^{\mathbb{I}}\dot{a}_x - \cos(\theta + \gamma_i)^{\mathbb{I}}\dot{a}_y = 0 \tag{4.10}$$

Once again inserting equations 4.6 and 4.7 in 4.10 and applying geometrical relations a similar result to the one found earlier is derived:

$$\begin{aligned} &\dot{x}\,(\sin\theta\cos\gamma_i + \cos\theta\sin\gamma_i) - \\ &\dot{y}\,(\cos\theta\cos\gamma_i - \sin\theta\sin\gamma_i) - \\ &\dot{\theta}\cos(\alpha_i - \gamma_i) = 0 \end{aligned} \tag{4.11}$$

$$\Downarrow$$

$$[\sin\gamma_i \quad -\cos\gamma_i \quad -\ell_i\cos(\alpha_i - \gamma_i)] \; R(\theta)\dot{\xi} = 0 \tag{4.12}$$

This is the matrix-equation describing the no lateral constraint for one wheel.

4.3 Constraint Matrix

The constraints of the robot can be written under the general matrix form extending the equations 4.9 and 4.12 to a general matrix-form expressing the constraints for all four wheels. These matrix

equations can be substituted into the constraint matrix Λ, and will be the subject of the following. For the no slip-constraint:

$$J_1(\gamma)R(\theta)\dot{\xi} - J_2\dot{\psi} = 0 \tag{4.13}$$

Where:

$$J_1(\gamma) = \begin{bmatrix} \cos\gamma_1 & \sin\gamma_1 & \ell_1 \sin(\gamma_1 - \alpha_1) \\ \cos\gamma_2 & \sin\gamma_2 & \ell_2 \sin(\gamma_2 - \alpha_2) \\ \cos\gamma_3 & \sin\gamma_3 & \ell_3 \sin(\gamma_3 - \alpha_3) \\ \cos\gamma_4 & \sin\gamma_4 & \ell_4 \sin(\gamma_4 - \alpha_4) \end{bmatrix} \tag{4.14}$$

And:

$$J_2 = \begin{bmatrix} R_w & 0 & 0 & 0 \\ 0 & R_w & 0 & 0 \\ 0 & 0 & R_w & 0 \\ 0 & 0 & 0 & R_w \end{bmatrix} \tag{4.15}$$

And for the no lateral movement-constraint:

$$C_1(\gamma)R(\theta)\dot{\xi} = 0 \tag{4.16}$$

With C_1 being:

$$C_1(\gamma) = \begin{bmatrix} \sin\gamma_1 & -\cos\gamma_1 & -\ell_1 \cos(\alpha_1 - \gamma_1) \\ \sin\gamma_2 & -\cos\gamma_2 & -\ell_2 \cos(\alpha_2 - \gamma_2) \\ \sin\gamma_3 & -\cos\gamma_3 & -\ell_3 \cos(\alpha_3 - \gamma_3) \\ \sin\gamma_4 & -\cos\gamma_4 & -\ell_4 \cos(\alpha_4 - \gamma_4) \end{bmatrix} \tag{4.17}$$

To describe the constraints imposed on the robot, the constraint matrix Λ is constructed as:

$$\Lambda = \begin{bmatrix} J_1(\gamma)R(\theta) & 0 & -J_2 \\ C_1(\gamma)R(\theta) & 0 & 0 \end{bmatrix} \tag{4.18}$$

such that:

$$\Lambda\dot{q} = \Lambda \begin{bmatrix} \dot{\xi} \\ \dot{\gamma} \\ \dot{\psi} \end{bmatrix} = 0 \tag{4.19}$$

CHAPTER 5

Kinematic Model

The objective of developing a kinematic model is to find a description of the change in generalized coordinates ($\dot{q}$) for the robot based on the inputs to the model, which in this case will be the linear velocity of the robot, and the changes in the steering angles in the front and back wheel pairs. The mathematical notation of the model is

$$\dot{q} = S(q)u \tag{5.1}$$

where $S(q)$ is the kinematic model matrix, u the model input and $\dot{q}$ is the change in generalized coordinates.

The kinematic model is derived by considering the constraints derived in the previous section. Taking the no lateral movement-constraint into consideration it is seen that every vector $R(\theta)\dot{\xi}$ is constrained to lie in the null-space of $C_1(\gamma)$ since:

$$C_1(\gamma)R(\theta)\dot{\xi} = 0 \tag{5.2}$$

is of the form $Ax = 0$. Defining $\Sigma(\gamma)$ as the null-space of $C_1(\gamma)$:

$$\Sigma(\gamma) = \mathcal{N}\left(C_1(\gamma)\right) \tag{5.3}$$

then for every t, there exist a signal $\eta(t)$ such that:

$$R(\theta)\dot{\xi} = \Sigma(\gamma)\eta(t) = \mathcal{N}\left(C_1(\gamma)\right)\eta(t) \tag{5.4}$$

when the constraints are fulfilled. By rearranging the expression $\dot{\xi}$ can be isolated:

$$\dot{\xi} = R^T(\theta)\Sigma(\gamma)\eta(t) \tag{5.5}$$

Since it has been established that $R(\theta)\dot{\xi} = \Sigma(\gamma)\eta(t)$ the following substitutions in equation 4.13 results in:

$$\begin{aligned} & J_1(\gamma)R(\theta)\dot{\xi} - J_2\dot{\psi} = 0 \\ \Downarrow & \\ & \dot{\psi} = J_2^{-1}J_1(\gamma)\Sigma(\gamma)\eta(t) \end{aligned} \tag{5.6}$$

Having found expressions for $\dot{\xi}$ and $\dot{\psi}$ an expression for the change in steering angles $\dot{\gamma}$ is needed. The steering angles of the front and back wheel pair are used as control inputs, so introducing the

vector ζ containing two signals for change in the steering angles in the front and rear of the robot:

$$\dot{\gamma} = \zeta = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \\ \zeta_4 \end{bmatrix}, \ \zeta_3 = \zeta_2 \wedge \zeta_4 = \zeta_1 \tag{5.7}$$

5.1 The rank of $C_1(\gamma)$

The rank of C_1 is of special interest. If $\text{rank}(C_1(\gamma)) = 3$ the only solution to equation 5.2 is $R(\theta)\dot{\xi} = 0$ which implies zero velocity and thereby no motion of the robot. Therefore the rank must be less than three:

$$\text{rank}(C_1(\gamma)) \leq 2 \tag{5.8}$$

The rank of $C_1(\gamma)$ depends on the design of the mobile robot in question. Furthermore the rank of $C_1(\gamma)$ is equal to the number of wheels that can be oriented independently in order to steer the robot [Champion, 1996].

The restrictions put on the robot mobility in equations 4.13 and 4.16 restricts any non-straight motion of the robot to turn around an ICR. The axis of rotation of all the wheels should intersect in this point. This is not possible for this robot, but as stated earlier it is assumed that the error will be neglectable.

In appendix F.3 the rank of $C_1(\gamma)$ has been evaluated using algebraic and numerical methods. It it shown that the rank of $C_1(\gamma)$ is 2 when the following is satisfied:

$$\gamma_1 = \gamma_2 \text{ or } \gamma_1 = -\gamma_2 = 0 \pm p\pi, \ p = 0, 1, 2... \tag{5.9}$$

Further numerical analysis of the singular values reveals that the effective rank of $C_1(\gamma)$ can be assumed to be 2. The assumption that the rank is 2 for all angles of γ implies that two of the rows in $C_1(\gamma)$ are linear combinations of the two other rows and allows the following reduction of $C_1(\gamma)$ into $C_1^*(\gamma)$:

$$\begin{aligned} C_1(\gamma) &= \begin{bmatrix} \sin\gamma_1 & -\cos\gamma_1 & -\ell_1\cos(\alpha_1 - \gamma_1) \\ \sin\gamma_2 & -\cos\gamma_2 & -\ell_2\cos(\alpha_2 - \gamma_2) \\ \sin\gamma_3 & -\cos\gamma_3 & -\ell_3\cos(\alpha_3 - \gamma_3) \\ \sin\gamma_4 & -\cos\gamma_4 & -\ell_4\cos(\alpha_4 - \gamma_4) \end{bmatrix} \\ &\sim \begin{bmatrix} \sin\gamma_i & -\cos\gamma_i & -\ell_i\cos(\alpha_i - \gamma_i) \\ \sin\gamma_j & -\cos\gamma_j & -\ell_j\cos(\alpha_j - \gamma_j) \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = C_1^*(\gamma) \end{aligned} \tag{5.10}$$

The indices i and j are referring to the two wheels that define the ICR, the robot will be turning about. Due to the design of the wheel suspension (and thereby the fact that an ICR for all four wheels is impossible), the wheels in question will be different according to the direction of the turn. This matter will be discussed in more detail in section 5.3.

5.2 Calculation of $\Sigma(\gamma)$

Having come to the conclusion that the effective rank of $C_1(\gamma)$ is 2, $\Sigma(\gamma)$ is calculated as the null-space of $C_1^*(\gamma)$ using MAPLE. The result is:

$$\Sigma(\gamma) = \begin{bmatrix} \ell_i \cos(\gamma_j)\cos(\alpha_i - \gamma_i) - \ell_j \cos(\gamma_i)\cos(\gamma_j - \alpha_j) \\ \ell_i \sin(\gamma_j)\cos(\alpha_i - \gamma_i) - \ell_j \sin(\gamma_i)\cos(\gamma_j - \alpha_j) \\ \sin(\gamma_i - \gamma_j) \end{bmatrix} \tag{5.11}$$

5.3 Determining *i* and *j*

As stated earlier the indices *i* and *j* in 5.11 refer to the numbers of the wheels (1 and 2 or 3 and 4) defining the ICR. The ideal solution to determining which wheels to use, would be to use the same two at all times. This is not possible though. If for instance wheel 1 and 2 were to define the ICR at all times, the robot would not be able to turn as hard to the one side as the other. Or in other words, all slippage is assumed to happen on wheel 3 and 4.

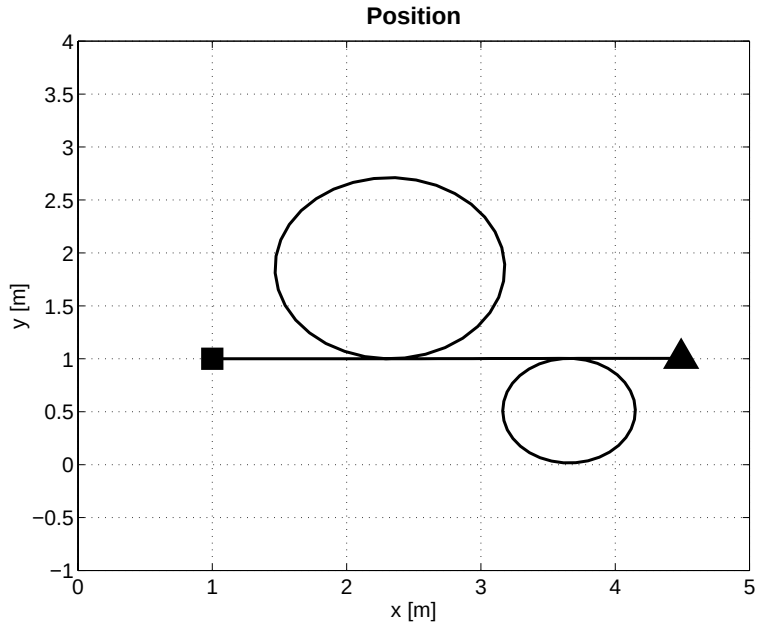

Figure 5.1: *A simulated trajectory of the robot with constant linear velocity* $\eta = 0.6\frac{m}{s}$ *and the same magnitude of* ζ_1 *and* ζ_2 *(the change of steering angles) when turning. The two ICR used are both defined from wheel 1 and 2.*

The resulting trajectory could in that case be as illustrated in figure 5.1, where the inputs to the model are the same when driving in both circles. Clearly the radii of the circles are not equal as they were supposed to be. The problem arises of the fact that the two ICR are defined from wheel 1 and 2. If wheel 1 and 2 defines the ICR at all times, the distance from the center of the robot to the ICR will be different according to which side the robot will be turning to. The problem is illustrated in figure 5.2.

The solution is to use one version of $\Sigma(\gamma)$ for turning left, and another version for turning right. [Missing: Only necessary in simulations, not in “real life”] The two matrices will be called

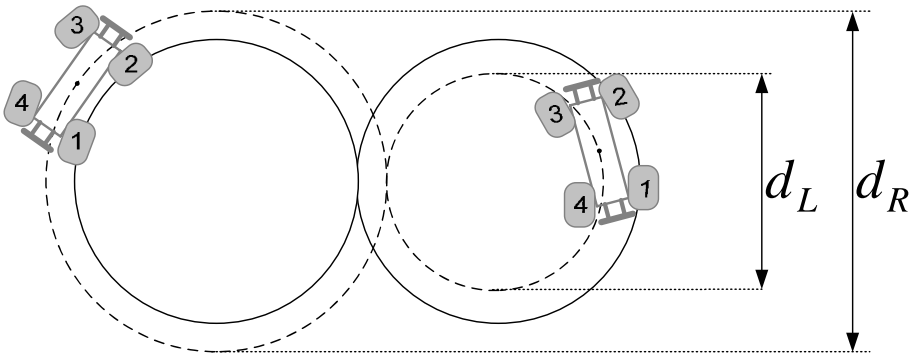

Figure 5.2: *The cause of the different radii. The ICR is defined by wheel 1 and 2. The result is that wheels 1 and 2 will cover the solid line circle, and the center of the robot will move on the dashed line circle. As seen the diameter of the two dashed line circles are not the same, thus the trajectory is incorrect.*

$\Sigma_L(\gamma)$ for left and $\Sigma_R(\gamma)$ for right:

$$\Sigma_L(\gamma) = \Sigma(\gamma)\Big|_{\substack{i=1\\j=2}} \tag{5.12}$$

$$\Sigma_R(\gamma) = \Sigma(\gamma)\Big|_{\substack{i=3\\j=4}} \tag{5.13}$$

Since $\ell_1 = \ell_2 = \ell_3 = \ell_4$ the only differences between $\Sigma_L(\gamma)$ and $\Sigma_R(\gamma)$ are the angles α_i and α_j and the angles γ_i and γ_j. γ_i and γ_j are variable, but α_i and α_j are constants. Replacing the i's and j's results in:

$$\Sigma_L(\gamma) = \begin{bmatrix} l\cos(\gamma_2)\cos(\alpha_1-\gamma_1) - l\cos(\gamma_1)\cos(\gamma_2-\alpha_2) \\ l\sin(\gamma_2)\cos(\alpha_1-\gamma_2) - l\sin(\gamma_1)\cos(\gamma_2-\alpha_2) \\ \sin(\gamma_1-\gamma_2) \end{bmatrix} \tag{5.14}$$

$$\Sigma_R(\gamma) = \begin{bmatrix} l\cos(\gamma_4)\cos(\alpha_3-\gamma_3) - l\cos(\gamma_3)\cos(\gamma_4-\alpha_4) \\ l\sin(\gamma_4)\cos(\alpha_3-\gamma_3) - l\sin(\gamma_3)\cos(\gamma_4-\alpha_4) \\ \sin(\gamma_3-\gamma_4) \end{bmatrix} \tag{5.15}$$

The criterion used to choose between $\Sigma_L(\gamma)$ and $\Sigma_R(\gamma)$ is defined by the angles γ_1 and γ_2 as illustrated in figure 5.3. By evaluating the difference between the steering angle of the front wheel pair and the rear wheel pair, it is possible to determine to which side of the robot the ICR is placed. To sum up, the criterion for choosing $\Sigma(\gamma)$ is stated in table 5.1. As stated in table 5.1 the choice of $\Sigma(\gamma)$ is free, if $\gamma_1 = \gamma_2 = 0$. In this case the trajectory of the robot will be a straight line parallel to the robot frame. Thus the distance from the center of the robot to the ICR will be the same, and thereby ensuring the same trajectory of the robot if $\Sigma_R(\gamma)$ or $\Sigma_L(\gamma)$ is used. Having defined $\Sigma(\gamma)$ the kinematic model can be written as:

$$\dot{q} = \begin{bmatrix} \dot{\xi} \\ \dot{\gamma} \\ \dot{\psi} \end{bmatrix} = S(q)u = \begin{bmatrix} R^T(\theta)\Sigma(\gamma) & 0 \\ 0 & I \\ J_2^{-1}J_1(\gamma)\Sigma(\gamma) & 0 \end{bmatrix} \begin{bmatrix} \eta \\ \zeta \end{bmatrix} \tag{5.16}$$

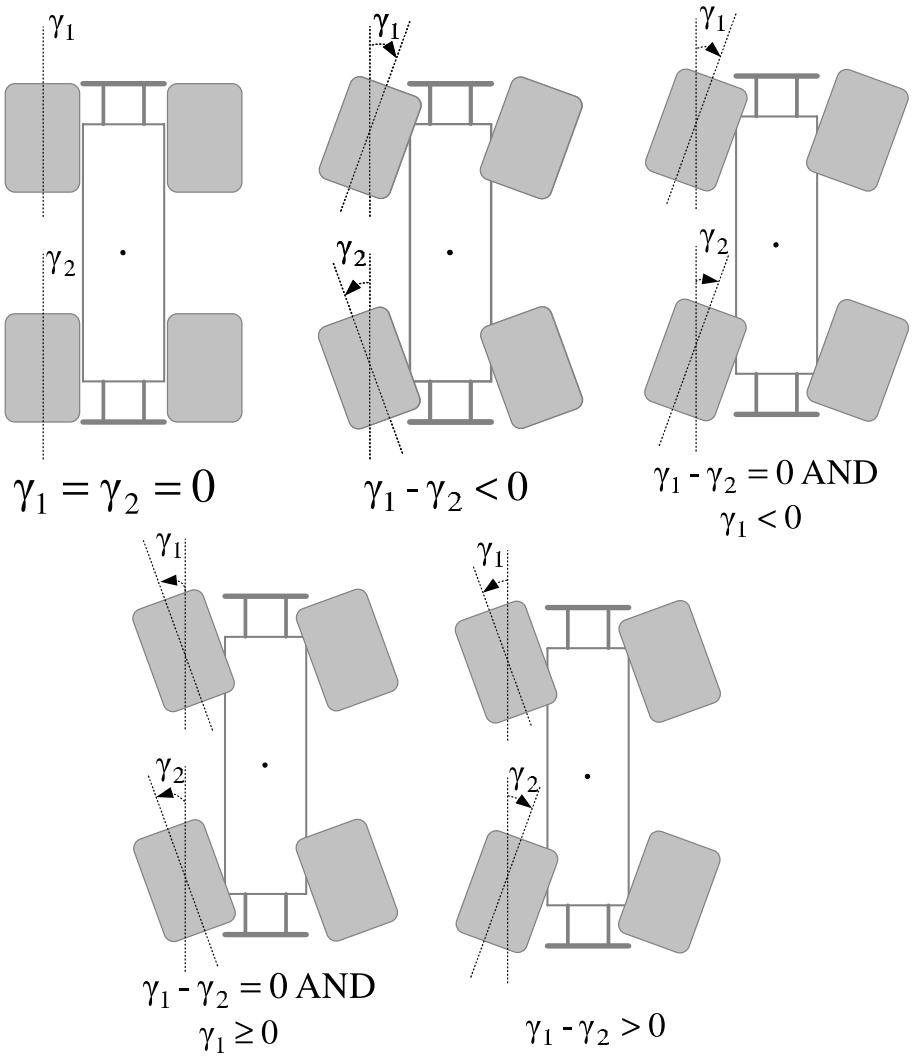

Figure 5.3: *The situations evaluated to choose the correct* $\Sigma(\gamma)$.

Criterion	$\Sigma(\gamma)$ used
$\gamma_1 = \gamma_2 = 0$	Does not matter.
$\gamma_1 - \gamma_2 < 0$	$\Sigma_R(\gamma)$
$\gamma_1 - \gamma_2 = 0$ AND $\gamma_1 \geq 0$	$\Sigma_L(\gamma)$
$\gamma_1 - \gamma_2 = 0$ AND $\gamma_1 < 0$	$\Sigma_R(\gamma)$
$\gamma_1 - \gamma_2 > 0$	$\Sigma_L(\gamma)$

Table 5.1: *The criterions used for choosing between* $\Sigma_L(\gamma)$ *and* $\Sigma_R(\gamma)$. *The criterions are only to be used in the simulations, since the problem only exists when plotting the trajectories.*

5.4 Posture Kinematic Model

The kinematic model can be reduced if only the posture ξ of the robot needs to be considered. This is e.g. the case when a controller needs to be designed. The reduced model is called the posture

kinematic model [Champion, 1996]. The kinematic model is given as:

$$\dot{q} = \begin{bmatrix} \dot{\xi} \\ \dot{\gamma} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} R^T(\theta)\Sigma(\gamma) & 0 \\ 0 & I_{4\times 4} \\ J_2^{-1}J_1(\gamma)\Sigma(\gamma) & 0 \end{bmatrix} u \tag{5.17}$$

In equation 5.17 u are the velocities $[\eta \ \zeta_1 \ \zeta_2 \ \zeta_3 \ \zeta_4]^T$. Due to the mechanics of the robot, the front wheels are always parallel and likewise are the back wheels. It is therefore only necessary to consider ζ_1 and ζ_2. The input u_{pk} to the posture kinematic model is therefore defined as:

$$u = [\eta \ \zeta_1 \ \zeta_2]^T \tag{5.18}$$

Another simplification is found in equation 5.17. Because $\dot{\psi}$ doesn't have any influence on the posture ξ in the kinematic model. The posture kinematic model is therefore:

$$\dot{q}_{pk} = \begin{bmatrix} \dot{\xi} \\ \dot{\gamma}_1 \\ \dot{\gamma}_2 \end{bmatrix} = S\left(q_{pk}\right) u_{pk} = \begin{bmatrix} R^T(\theta)\Sigma(\gamma) & 0 \\ 0 & I_{2\times 2} \end{bmatrix} \begin{bmatrix} \eta \\ \zeta_1 \\ \zeta_2 \end{bmatrix} \tag{5.19}$$

5.5 Verification of the Kinematic Model

Verification of the kinematic model is possible but letting all the steering angles be equal to zero:

$$\gamma_i = 0, \ i = 1 \ldots 4 \tag{5.20}$$

Substituting these angles into the expression for $\Sigma(\gamma)$ yields:

$$\Sigma(\gamma) = \begin{bmatrix} 2\ell \sin\left(\frac{\pi}{2} - \alpha_1\right) \\ 0 \\ 0 \end{bmatrix} \tag{5.21}$$

The change in posture $\dot{\xi}$

$$\dot{\xi} = R^T(\theta)\Sigma(\gamma)\eta(t) = \begin{bmatrix} 2\cos\theta\ell \sin\left(\frac{\pi}{2} - \alpha_1\right) \\ 2\sin\theta\ell \sin\left(\frac{\pi}{2} - \alpha_1\right) \\ 0 \end{bmatrix} \eta(t) \tag{5.22}$$

From this equation it is seen, that for $\theta = 0$, the change in posture is zero for y and θ, implying that the robot only moves along the x direction. Thus η can be interpreted as the linear velocity of the robot with the unit $\frac{m}{s}$.

5.6 Simulations

The posture of the robot is described by the coordinate vector ξ, and an expression for $\dot{\xi}$ was found in equation 5.5

$$\dot{\xi} = R^T(\theta)\Sigma(\gamma)\eta(t)$$

In MATLAB a simulation integrating $\dot{\xi}$ over time was implemented. The input to the simulation was changed over time as illustrated in figure 5.4. The resulting trajectory is seen in figure 5.5.

The trajectory is as expected, and it is noted that the problem with different radii in circles has been solved.

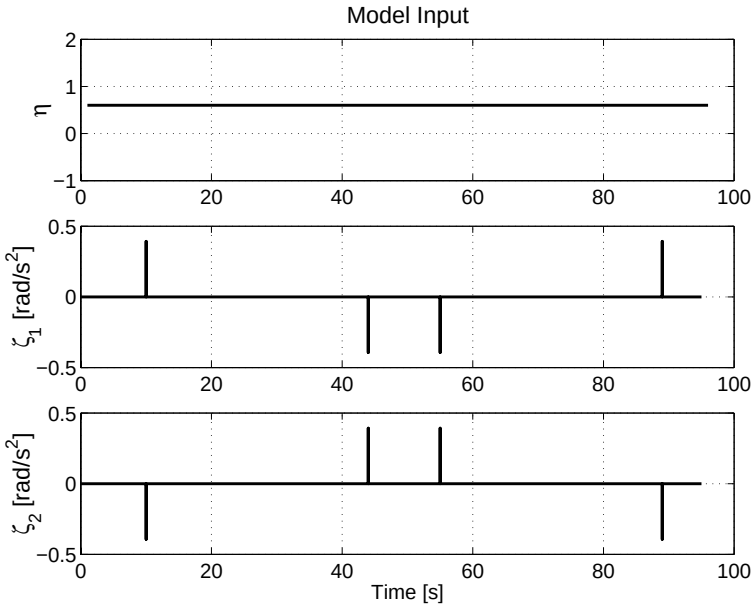

Figure 5.4: *The three inputs to the kinematic model in the simulation. In the top the linear velocity η is seen, and below the changes in steering angles ζ_1 and ζ_2 are plotted.*

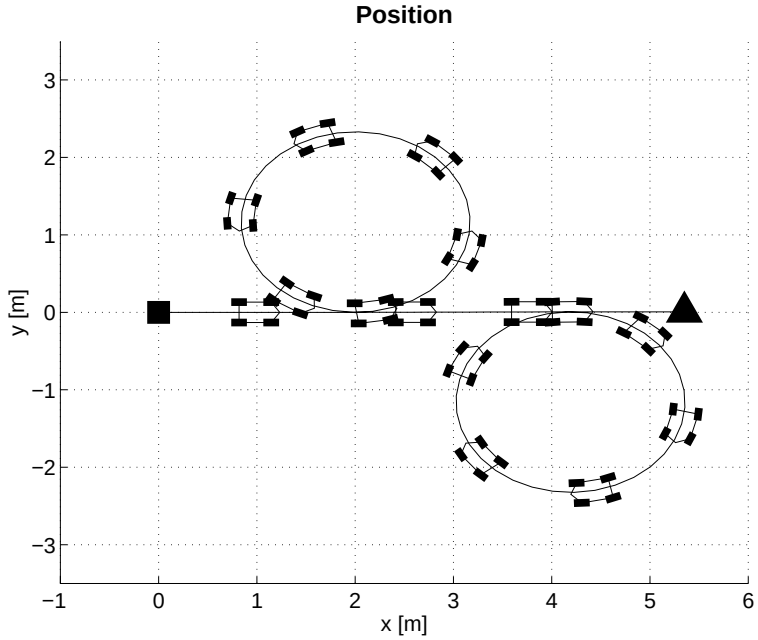

Figure 5.5: *The resulting posture of the robot with the inputs from 5.4 using only the kinematic model . The simulation is run for 95 seconds and reveals that the robot posture is as expected.*

5.7 Summary

The kinematic model derived in the previous can be summed up as

$$\dot{q} = S(q)u \tag{5.23}$$

$$\Downarrow$$

$$\begin{bmatrix} \dot{\xi} \\ \dot{\gamma} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} R^T(\theta)\Sigma(\gamma) & 0 \\ 0 & I \\ J_2^{-1}J_1(\gamma)\Sigma(\gamma) & 0 \end{bmatrix} \begin{bmatrix} \eta \\ \zeta \end{bmatrix} \tag{5.24}$$

The kinematic model derived expresses the change in the robots configuration (the generalized coordinate vector) as a function of the control input η and ζ. η is the linear velocity of the robot and $\zeta = [\zeta_1 \quad \zeta_2]^T$ is the change in the front and rear steering angles. η will eventually be applied by the DC-motor, and the changes in steering angles ζ_1 and ζ_2 will be applied using the servo motors. The matrix $\Sigma(\gamma)$ varies according to which side of the robot, the ICR is placed.

CHAPTER 6

Dynamical Model

In this section a dynamical model of the robot is derived. The dynamical model describes the relation between the torques τ from the motors and the resulting change in velocities $\dot{u}$. The relation between the torques and the resulting change in velocities isn't a simple straight forward calculation. The approach to the problem is to consider the energy of the robot. Applied torque from the motor will change the energy of the robot. Based on this change in energy, a resulting change in velocities can be found. The relation between energy, torques and change of velocities can be found through Lagrange equations. The Lagrangian $L(q, \dot{q})$ of the system is given as the difference between the kinetic energy $T(q, \dot{q})$ and the potential energy $W(q)$ of the system [Oriolo, 1995]:

$$L(q, \dot{q}) = T(q, \dot{q}) - W(q) \tag{6.1}$$

The dynamical model is derived from Lagrange equations with extension of Hamilton's principle to nonholonomic systems. The used Lagrange formalism is:

$$\frac{d}{dt}\left(\frac{\partial L(q, \dot{q})}{\partial \dot{q}}\right) - \frac{\partial L(q, \dot{q})}{\partial q} = A(q)\lambda + Q(q)\tau \tag{6.2}$$

Where q is the generalized coordinates and λ is undetermined Lagrange multipliers. $A(q)$ is the transpose of the constraint matrix and $Q(q)$ is a matrix, which maps the the external inputs τ into forces/torques performing work on q.

Because it is assumed that the robot is moving in a horizontal plane the potential energy remains constant. Since only changes in the energy of the system will have an effect and result in a change of velocities the constant potential energy can be neglected. The Lagrangian of the system is therefore equal to the kinetic energy. In order to reduce the calculations the kinetic energy is partioned into energy in the frame T_F and energy in the wheels T_W. In the following sections T_F and T_W is found as an expression of the state variables in q.

6.1 Kinetic Energy of the Frame

The frame of the robot is approximated with a rectangular rigid body. The kinetic energy of the frame is the sum of the energy from the linear and rotational movement around the center of mass

CM:

$$T_F = \frac{1}{2} M_F v^2 + \frac{1}{2} I_F \dot{\omega}^2 \tag{6.3}$$

$$\Downarrow$$

$$T_F = \frac{1}{2} M_F (\dot{x}_{CM}^2 + \dot{y}_{CM}^2) + \frac{1}{2} I_F \dot{\theta}^2 \tag{6.4}$$

Where v is the linear velocity for CM and ω is the angular velocity around CM. M_F is the mass of the frame and I_F is the moment of inertia for the frame in $[kg \cdot m^2]$. With the definitions given

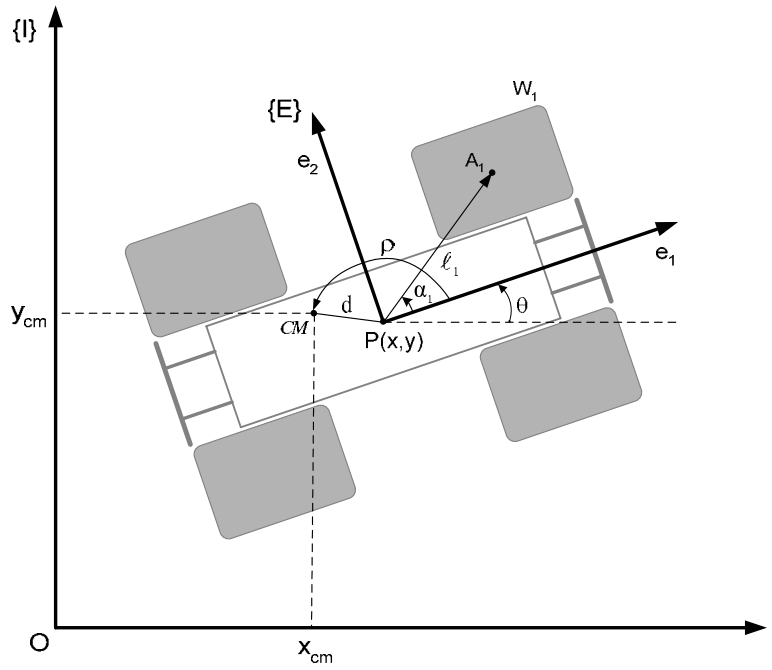

Figure 6.1: *Definitions used to find the coordinates for the center of mass CM.*

in figure 6.1 the CM with the coordinates (x_{CM}, y_{CM}) in the inertial frame is given as:

$$x_{CM} = x + d\cos(\theta + \rho) \tag{6.5}$$

$$y_{CM} = y + d\sin(\theta + \rho) \tag{6.6}$$

The derivative of equation 6.5 and 6.6 is:

$$\dot{x}_{CM} = \dot{x} - \dot{\theta} d \sin(\theta + \rho) \tag{6.7}$$

$$\dot{y}_{CM} = \dot{y} + \dot{\theta} d \cos(\theta + \rho) \tag{6.8}$$

Inserting equation 6.7 and 6.8 in 6.4 yields the following expression for the kinetic energy of the frame:

$$\begin{aligned} T_F = {} & \frac{1}{2} M_F \dot{x}^2 + \frac{1}{2} M_F \dot{y}^2 + \frac{1}{2} M_F \dot{\theta}^2 d^2 + \frac{1}{2} I_F \dot{\theta}^2 \\ & - \dot{x}\dot{\theta} d M_F \sin(\theta + \rho) + \dot{y}\dot{\theta} d M_F \cos(\theta + \rho) \end{aligned} \tag{6.9}$$

6.2 Kinetic Energy of a Wheel

It is assumed that the wheel is perfectly round. Furthermore the contact area between the wheel and the ground is assumed to be a point to minimize complexity. Let P_w be a point on the edge of the wheel as shown in figure 6.2. Let the point P_w have an infinitesimal small mass m_{Pw} which

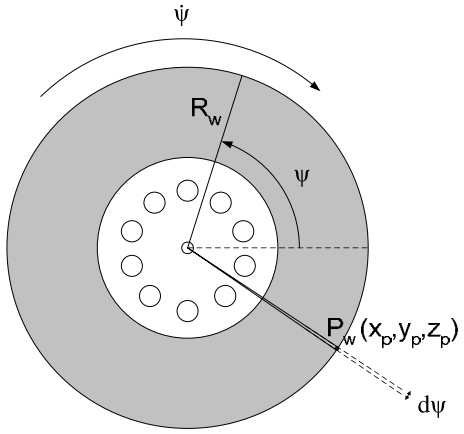

Figure 6.2: *A wheel on the robot seen from the side.*

spans $d\psi$ [rad] on the edge of the wheel. Assuming that the entire mass of the wheel is evenly distributed along the edge with an angular density $p\,[kg/rad]$, the mass m_{Pw} is:

$$m_{Pw} = p\,d\psi$$

The kinetic energy of P_w is:

$$T_w = \frac{1}{2} m_{Pw} v_{Pw}^2 = \frac{1}{2} p\,d\psi\, v_{Pw}^2$$

Where v_{Pw} is the linear velocity of P_w. The kinetic energy of a wheel T_W is found by integration of all the masses around the wheel:

$$\begin{aligned} T_W &= \frac{1}{2}\int_0^{2\pi} v_{pw}^2\, p\, d\psi \\ \Downarrow & \\ T_W &= \frac{1}{2}\int_0^{2\pi} \left(\sqrt{(\dot{x}_P^2 + \dot{y}_P^2 + \dot{z}_P^2)}\right)^2 p\, d\psi \\ \Downarrow & \\ T_W &= \frac{1}{2}\int_0^{2\pi} (\dot{x}_P^2 + \dot{y}_P^2 + \dot{z}_P^2) p\, d\psi \end{aligned} \tag{6.10}$$

The point P_w is described with respect to the inertial frame with the following equations:

$$x_P = x + l\cos(\alpha+\theta) + R_w\cos(\psi)\cos(\gamma+\theta) \tag{6.11}$$

$$y_P = y + l\sin(\alpha+\theta) + R_w\cos(\psi)\sin(\gamma+\theta) \tag{6.12}$$

$$z_P = R_w\sin(\psi) \tag{6.13}$$

The time derivatives of the coordinates are:

$$\begin{aligned}
\dot{x}_P &= \dot{x} - l\dot{\theta}\sin(\alpha+\theta) - R_w\dot{\psi}\sin(\psi)\cos(\gamma+\theta) \\
&\quad - R_w\dot{\gamma}\cos(\psi)\sin(\gamma+\theta) - R_w\dot{\theta}\cos(\psi)\sin(\gamma+\theta) && (6.14)\\
\dot{y}_P &= \dot{y} + l\dot{\theta}\cos(\alpha+\theta) + R_w\dot{\psi}\sin(\psi)\sin(\gamma+\theta) \\
&\quad + R_w\dot{\psi}\cos(\psi)\cos(\gamma+\theta) + R_w\dot{\theta}\cos(\psi)\cos(\gamma+\theta) && (6.15)\\
\dot{z}_P &= R_w\dot{\psi}\cos(\psi) && (6.16)
\end{aligned}$$

The next step is to find the square of equation 6.14, 6.15 and 6.16 and inserting in equation 6.10. The calculation of this is substantial and can be found in appendix F. The result found in appendix F is:

$$\begin{aligned}
T_W &= \frac{1}{2}\dot{x}^2 m + \frac{1}{2}\dot{y}^2 m + ml\dot{\theta}\big(\dot{y}cos(\alpha+\theta) - \dot{x}sin(\alpha+\theta)\big) \\
&\quad + \frac{1}{2}l^2\dot{\theta}^2 m + \frac{1}{2}I_W\dot{\theta}^2\dot{\psi}^2 + \frac{1}{4}I_W\dot{\gamma}^2 + \frac{1}{2}I_W\dot{\theta}\dot{\gamma} + \frac{1}{4}I_W\dot{\theta}^2 && (6.17)
\end{aligned}$$

Where the following two relations has been used:

$$\begin{aligned}
m &= 2\pi p && (6.18)\\
I_W &= mR_w^2 && (6.19)
\end{aligned}$$

m is mass of a wheel which is assumed to be equal for the four wheels. I_W is the moment of inertia for a wheel.

6.3 Total Kinetic Energy

The total kinetic energy for the robot is a sum of the energy from the frame and the energy from the four wheels. In the following equations it is assumed that the distance l is equal for the four wheels:

$$\begin{aligned}
T_{Total} &= T_F + \sum_{i=1}^{4} T_{Wi} && (6.20)\\
&= \frac{1}{2}\dot{x}^2(M_F+4m) + \frac{1}{2}\dot{y}^2(M_F+4m) + \frac{1}{2}\dot{\theta}^2(4l^2m + 2I_W + d^2M_F + I_F) \\
&\quad + \frac{1}{4}I_W\sum_{i=1}^{4}\dot{\gamma}_i^2 + \frac{1}{4}I_W\sum_{i=1}^{4}\dot{\psi}_i^2 + \frac{1}{2}I_W\dot{\theta}\sum_{i=1}^{4}\dot{\gamma}_i && (6.21)\\
&\quad - \dot{x}\dot{\theta}\left(2dM_F\sin(\theta+\rho) + ml\sum_{i=1}^{4}\sin(\alpha_i+\theta)\right) \\
&\quad + \dot{y}\dot{\theta}\left(2dM_F\cos(\theta+\rho) + ml\sum_{i=1}^{4}\cos(\alpha_i+\theta)\right)
\end{aligned}$$

Introducing the matrices M and I_{WM} as:

$$M = \begin{bmatrix} M_F + 4m & 0 & -2dM_F\sin(\rho) - ml\sum_{i=1}^{4}\sin(\alpha_i) \\ 0 & M_F + 4m & 2dM_F\cos(\rho) + ml\sum_{i=1}^{4}\cos(\alpha_i) \\ 0 & 0 & 4l^2m + 2I_W + d^2M_F + I_F \end{bmatrix}, \quad I_{WM} = \begin{bmatrix} I_W & 0 & 0 & 0 \\ 0 & I_W & 0 & 0 \\ 0 & 0 & I_W & 0 \\ 0 & 0 & 0 & I_W \end{bmatrix}$$

$$\dot{\psi}^T = [\dot{\psi}_1, \dot{\psi}_2, \dot{\psi}_3, \dot{\psi}_3], \quad I_{WV} = [I_W, I_W, I_W, I_W]$$

The total kinetic energy can then be written as:

$$T_{Total} = \frac{1}{2}\dot{\xi}^T R^T(\theta) M R(\theta)\dot{\xi} + \frac{1}{4}\dot{\gamma}^T I_{WM}\dot{\gamma} + \frac{1}{4}\dot{\psi}^T I_{WM}\dot{\psi} + \frac{1}{2}\dot{\theta} I_{WV}\dot{\gamma} \tag{6.22}$$

Or in a more compact form as:

$$T_{Total} = \frac{1}{2}\dot{q} D(q)\dot{q}^T \tag{6.23}$$

where $D(q)$ is called the inertia matrix and is defined as:

$$D(q) = \begin{bmatrix} & & & 0 & 0 & 0 & 0 & & & & \\ & \left[R^T(\theta) M R(\theta)\right] & & 0 & 0 & 0 & 0 & & 0_{3\times4} & & \\ & & & \frac{1}{2}I_W & \frac{1}{2}I_W & \frac{1}{2}I_W & \frac{1}{2}I_W & & & & \\ 0 & 0 & \frac{1}{2}I_W & \frac{1}{2}I_W & 0 & 0 & 0 & & & & \\ 0 & 0 & \frac{1}{2}I_W & 0 & \frac{1}{2}I_W & 0 & 0 & & 0_{4\times4} & & \\ 0 & 0 & \frac{1}{2}I_W & 0 & 0 & \frac{1}{2}I_W & 0 & & & & \\ 0 & 0 & \frac{1}{2}I_W & 0 & 0 & 0 & \frac{1}{2}I_W & & & & \\ & & & & & & & \frac{1}{2}I_W & 0 & 0 & 0 \\ & 0_{4\times3} & & & 0_{4\times4} & & & 0 & \frac{1}{2}I_W & 0 & 0 \\ & & & & & & & 0 & 0 & \frac{1}{2}I_W & 0 \\ & & & & & & & 0 & 0 & 0 & \frac{1}{2}I_W \end{bmatrix} \tag{6.24}$$

6.4 Removal of Lagrangian multipliers

Consider again used Lagrangian formalism from equation 6.2:

$$\frac{d}{dt}\left(\frac{\partial L(q,\dot{q})}{\partial \dot{q}}\right) - \frac{\partial L(q,\dot{q})}{\partial q} = A(q)\lambda + Q(q)\tau \tag{6.25}$$

In this section the Lagrangian multipliers λ are removed because they are undetermined and therefore unwanted in the model. Before the Lagrangian multipliers are removed the terms in equation 6.25 needs to be determined.

The matrix $A(q)$ is equal to the transpose of the constraint matrix Λ, which is known from

section 4.3:

$$\Lambda = A^T(q) = \begin{bmatrix} J_1(\gamma)R(\theta) & 0 & -J_2 \\ C_1(\gamma)R(\theta) & 0 & 0 \end{bmatrix} \tag{6.26}$$

$$\Downarrow$$

$$A(q) = \begin{bmatrix} R^T(\theta)J_1^T(\gamma) & R^T(\theta)C_1^T(\gamma) \\ 0 & 0 \\ -J_2^T & 0 \end{bmatrix} \tag{6.27}$$

The input τ are torques performing work on the generalized coordinates. The work is performed on γ and ψ. No forces are directly performing work on ξ and the matrix $Q(q)$ therefore becomes:

$$Q(q) = \begin{bmatrix} 0_{3\times 8} \\ I_{8\times 8} \end{bmatrix} \tag{6.28}$$

The Lagrangian of the system $L(q,\dot{q})$ was equal to the total kinetic energy T_{Total}. Unless otherwise is stated T_{Total} will from now on be refered to as T to reduce the length of the equations. The left side of equation 6.25 can be written as:

$$\frac{d}{dt}\left(\frac{\partial L(q,\dot{q})}{\partial \dot{q}}\right) - \frac{\partial L(q,\dot{q})}{\partial q} = \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}}\right) - \frac{\partial T}{\partial q} = \frac{d}{dt}\begin{bmatrix} \frac{\partial T}{\partial \dot{\xi}} \\ \frac{\partial T}{\partial \dot{\gamma}} \\ \frac{\partial T}{\partial \dot{\psi}} \end{bmatrix} - \begin{bmatrix} \frac{\partial T}{\partial \xi} \\ \frac{\partial T}{\partial \gamma} \\ \frac{\partial T}{\partial \psi} \end{bmatrix} \tag{6.29}$$

With equation 6.27 and 6.29 the Lagrangian formalism from equation 6.25 is can be written as:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\xi}}\right) - \frac{\partial T}{\partial \xi} = R^T(\theta)J_1^T(\gamma)\lambda_1 + R^T(\theta)C_1^T(\gamma)\lambda_2 \tag{6.30}$$

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\gamma}}\right) - \frac{\partial T}{\partial \gamma} = \tau_\gamma \tag{6.31}$$

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\psi}}\right) - \frac{\partial T}{\partial \psi} = -J_2^T\lambda_1 + \tau_\psi \tag{6.32}$$

Where λ_1 and λ_2 are the Lagrangian multipliers. A short hand notation $[T]$ for the terms on the left side of equation 6.30-6.32 is used in the following:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\xi}}\right) - \frac{\partial T}{\partial \xi} = [T]_\xi$$

In order to eliminate the Lagrangian multipliers equation 6.30 is multiplied with $\Sigma^T(\gamma)R(\theta)$ and equation 6.32 is multiplied with $\Sigma^T(\gamma)J_1^T(\gamma)(J_2^{-1})^T$. Thus results in:

$$\Sigma^T(\gamma)R(\theta)[T]_\xi = \Sigma^T(\gamma)J_1^T(\gamma)\lambda_1 + \Sigma^T(\gamma)C_1^T(\gamma)\lambda_2 \tag{6.33}$$

$$\Sigma^T(\gamma)J_1^T(\gamma)(J_2^{-1})^T[T]_\psi = -\Sigma^T(\gamma)J_1^T(\gamma)\lambda_1 + \Sigma^T(\gamma)J_1^T(\gamma)(J_2^{-1})^T\tau_\psi \tag{6.34}$$

Since $\Sigma(\gamma)$ is the null space of $C_1(\gamma)$ it is known that:

$$\Sigma(\gamma)C_1(\gamma) = 0 = \Sigma^T(\gamma)C_1^T(\gamma) \tag{6.35}$$

Thereby the second term on the right side of equation 6.33 becomes zero. Adding equation 6.33 and 6.34 removes λ_1. The three equations 6.30, 6.32 and 6.31 is reduced to:

$$\Sigma^T(\gamma)\big(R(\theta)[T]_\xi + J_1^T(\gamma)(J_2^{-1})^T[T]_\psi\big) = \Sigma^T(\gamma)J_1^T(\gamma)(J_2^{-1})^T\tau_\psi \tag{6.36}$$

$$[T]_\gamma = \tau_\gamma \tag{6.37}$$

In appendix F.2 $[T]_\xi$, $[T]_\psi$ and $[T]_\gamma$ is found to be:

$$[T]_\xi = \dot{P}\dot{\xi} + P\ddot{\xi} + \frac{1}{2}K_1 I_{WV}\ddot{\gamma} - K_1\dot{\xi}^T R^T(\theta)NR(\theta)\dot{\xi} \tag{6.38}$$

$$[T]_\psi = \frac{1}{2}I_{WM}\ddot{\psi} \tag{6.39}$$

$$[T]_\gamma = \frac{1}{2}I_{WM}\ddot{\gamma} + \frac{1}{2}\ddot{\theta}I_W K_2 \tag{6.40}$$

where K_1, K_2, N, and P are defined as:

$$K_1 = [0\ 0\ 1]^T \tag{6.41}$$

$$K_2 = [1\ 1\ 1\ 1]^T \tag{6.42}$$

$$N = \begin{bmatrix} 0 & 0 & -\frac{1}{2}\theta M_3 \\ 0 & 0 & \frac{1}{2}\theta M_2 \\ -\frac{1}{2}\theta M_3 & \frac{1}{2}\theta M_2 & O \end{bmatrix} \tag{6.43}$$

$$P = \begin{bmatrix} M_1 & 0 & \frac{1}{2}(M_2\cos(\theta) - M_3\sin(\theta)) \\ 0 & M_1 & \frac{1}{2}(M_2\sin(\theta) + M_3\cos(\theta)) \\ \frac{1}{2}(M_2\cos(\theta) - M_3\sin(\theta)) & \frac{1}{2}(M_2\sin(\theta) + M_3\cos(\theta)) & M_4 \end{bmatrix} \tag{6.44}$$

Equations 6.38, 6.39 and 6.40 are inserted into equations 6.36 and 6.37:

$$\Sigma^T(\gamma)\Big[R(\theta)\big(\dot{P}\dot{\xi} + P\ddot{\xi} + \frac{1}{2}K I_{WV}\ddot{\gamma} - K_1\dot{\xi}^T R^T(\theta)NR(\theta)\dot{\xi}\big) + J_1^T(\gamma)(J_2^{-1})^T\frac{1}{2}I_{WM}\ddot{\psi}\Big] = \Sigma^T(\gamma)J_1^T(\gamma)(J_2^{-1})^T\tau_\psi \tag{6.45}$$

$$\frac{1}{2}I_{WM}\ddot{\gamma} + \frac{1}{2}\ddot{\theta}I_W K_2 = \tau_\gamma \tag{6.46}$$

The last step is to substitute the terms $\ddot{\xi}$, $\ddot{\psi}$, $\ddot{\gamma}$ and $\dot{\xi}$ with terms containing the control input ζ to the kinematic model. Hereby obtaining state equations for the dynamical model which describe the relation between the torque inputs τ_ψ and τ_γ and the output ζ. The terms for $\ddot{\xi}$, $\ddot{\psi}$, $\ddot{\gamma}$ and $\dot{\xi}$ is

know from the kinematic model and are:

$$\dot{\xi} = R^T(\theta)\Sigma(\gamma)\eta \tag{6.47}$$

$$\dot{\psi} = J_2^{-1}J_1(\gamma)\Sigma(\gamma)\eta = E(\gamma)\Sigma(\gamma)\eta \tag{6.48}$$

$$\dot{\gamma} = \begin{bmatrix} \zeta_1 \\ \zeta_2 \end{bmatrix} = \zeta \tag{6.49}$$

$$\Downarrow$$

$$\ddot{\xi} = \dot{R}^T(\theta)\Sigma(\gamma)\eta + R^T(\theta)\dot{\Sigma}(\gamma)\eta + R^T(\theta)\Sigma(\gamma)\dot{\eta} \tag{6.50}$$

$$\ddot{\psi} = \dot{E}(\gamma)\Sigma(\gamma)\eta + E(\gamma)\dot{\Sigma}(\gamma)\eta + E(\gamma)\Sigma(\gamma)\dot{\eta} \tag{6.51}$$

$$\ddot{\gamma} = \dot{\zeta} \tag{6.52}$$

Substituting these expressions into equation 6.45 and 6.46 yields:

$$\begin{aligned}
&\Sigma^T(\gamma)\Big[R(\theta)\Big(\dot{P}R^T(\theta)\Sigma(\gamma)\eta + P[\dot{R}^T(\theta)\Sigma(\gamma)\eta + R^T(\theta)\dot{\Sigma}(\gamma)\eta + R^T(\theta)\Sigma(\gamma)\dot{\eta}] \\
&+\frac{1}{2}K_1 I_{WV}\zeta - K_1\eta\Sigma^T(\gamma)R(\gamma)R^T(\theta)NR(\theta)R^T(\theta)\Sigma(\gamma)\eta\Big) \\
&+E^T(\gamma)\frac{1}{2}I_{WM}[\dot{E}(\gamma)\Sigma(\gamma)\eta + E(\gamma)\dot{\Sigma}(\gamma)\eta + E(\gamma)\Sigma(\gamma)\dot{\eta}]\Big] \\
&= \Sigma^T(\gamma)E^T(\gamma)\tau_\psi
\end{aligned} \tag{6.53}$$

$$\frac{1}{2}I_{WM}\dot{\zeta} + \frac{1}{2}\ddot{\theta}I_W K_2 = \tau_\gamma \tag{6.54}$$

Equation 6.53 and 6.54 can be rewritten as:

$$H(\gamma)\dot{u} + f(\gamma, u) = F(\gamma)\tau \tag{6.55}$$

Where:

$$\begin{aligned}
\tau &= \begin{bmatrix} \tau_\psi \\ \tau_\gamma \end{bmatrix} \\
F(\gamma) &= \begin{bmatrix} \Sigma^T(\gamma)E^T(\gamma) & 0 \\ 0 & I_{4\times 4} \end{bmatrix} \\
f(\gamma, u) &= \begin{bmatrix} f_1(\gamma, u) \\ f_2(\gamma, u) \end{bmatrix} \\
f_1(\gamma, u) &= \Sigma^T(\gamma)\Big[\big[R(\theta)\dot{P}R^T(\theta) + R(\theta)P\dot{R}^T(\theta) + \\
&\quad -R(\theta)K_1\eta\Sigma^T(\gamma)N + \frac{1}{2}E^T(\gamma)I_{WM}E(\gamma)\big]\Sigma(\gamma) \\
&\quad \big[R(\theta)PR^T(\theta) + \Sigma^T(\gamma)R(\theta)\frac{1}{2}E^T(\gamma)I_{WM}E(\gamma)\big]\dot{\Sigma}(\gamma)\Big]\eta + \frac{1}{2}K_1 I_{WV}\zeta \\
f_2(\gamma, u) &= \frac{1}{2}\ddot{\theta}I_W K_2 \\
H(\gamma) &= \begin{bmatrix} H_1(\gamma) & 0 \\ 0 & \frac{1}{2}I_{WM} \end{bmatrix} \\
H_1(\gamma) &= \Sigma(\gamma)[R(\theta)PR^T(\theta) + \frac{1}{2}E^T(\gamma)I_{WM}E(\gamma)]\Sigma(\gamma)
\end{aligned}$$

With equation 6.55 a state space model for the robot can be written as:

$$H(\gamma)\dot{u} = -f(\gamma, u) + F(\gamma)\tau \quad (6.56)$$

$$\dot{q} = S(q)u \quad (6.57)$$

6.5 Simulations

In order to verify the dynamical model, the simulation of the kinematic model was expanded to include the dynamic model of the robot. The dynamic model takes torques for propulsion (τ_ψ) and steering (τ_{γ_1} and τ_{γ_2}) as inputs. For the propulsion the same torque is applied to each wheel, and for the steering torque is applied at the front and rear wheel pair. The inputs for the simulation are plotted in figure 6.3. At time 2 a torque is applied to propulsion, and at the same time torque is applied to the rear wheel pair steering. No torque is applied to steer the front wheel pair. The

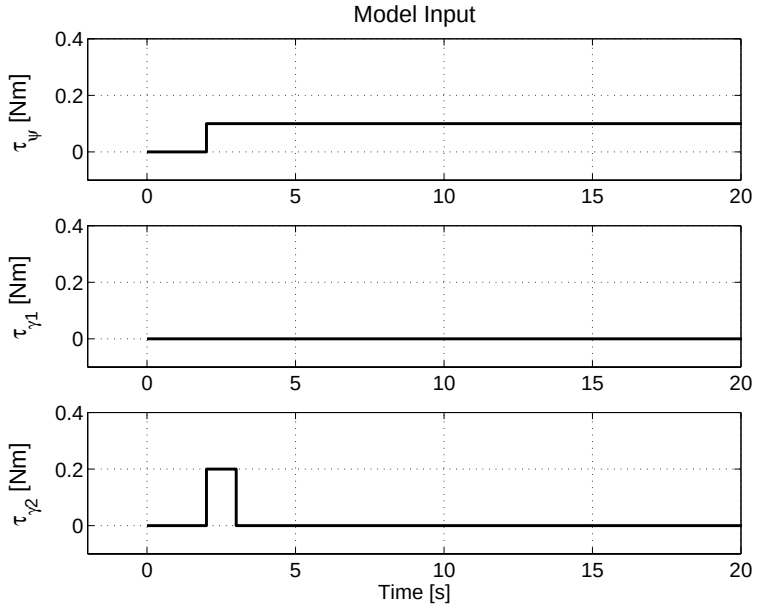

Figure 6.3: *The input used for the simulation of both the kinematic and dynamic model. In the top the torque applied to the wheels propulsion is seen. The second and third plots are the torques applied to the change of steering in the front and rear.*

resulting trajectory of the robot is available in figure 6.4. As expected the robot is turning, but after some time, the trajectory becomes a straight line. When the rear wheel pair is steered, the robot frame is rotating. The rotation of the robot frame makes the steering angle of the front wheel pair change (one can say that the robot frame is rotating above the front wheel pair). The changes in steering angles ends when the steering angles of the front and rear wheel pairs are parallel resulting in a straight line trajectory. The fact that the steering angle of the front wheel pair is changing because of the steering angle of the rear wheel pair is more obvious from figure 6.5. The plot shows the steering angle of the front and rear wheel pair. The front angle is in the left side, and the rear angle is in the right side of the plot. The rear steering angle is changing as the torque is applied, but the angle of the front pair is changing slower due to the momentum of the robot.

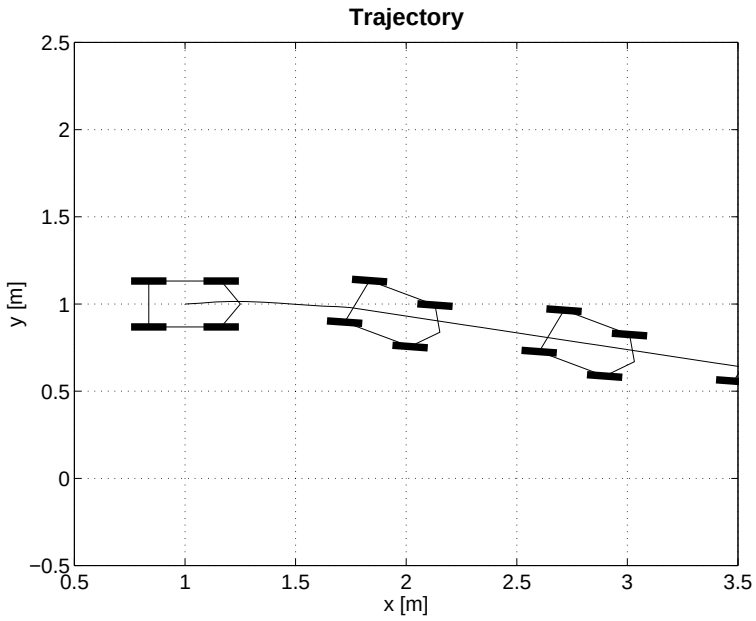

Figure 6.4: *The resulting trajectory from the kinematic and dynamic model using the input from figure 6.3. Since there is only applied steering to the rear wheels, the robot frame turns until the front and rear wheels are parallel. Note that no torque is applied to the front wheel steering.*

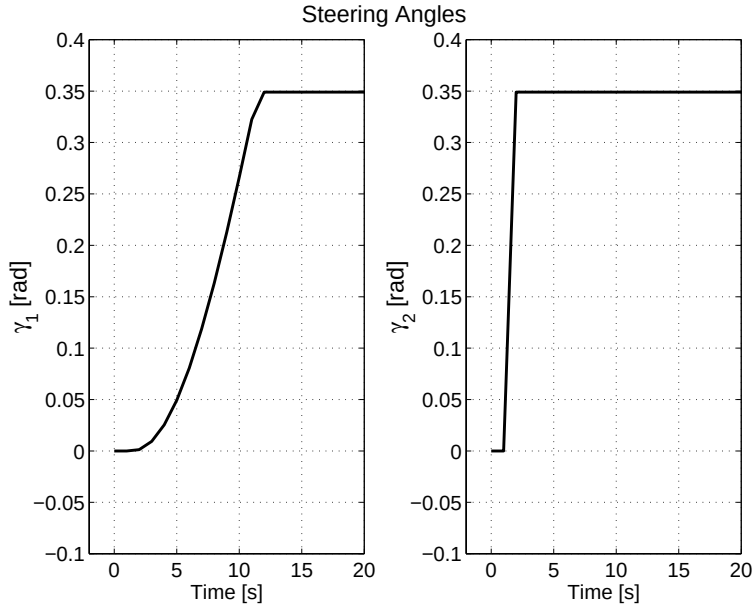

Figure 6.5: *The wheel angles for the front (left side of the plot) and rear (right side of the plot) wheel pair. The angle of the rear pair changes when the torque is applied, and the angle of the front pair changes over time afterwards.*

CHAPTER 7

DC-motor and Servo Model

7.1 DC-motor

In this section a model for the DC-motors including SSC-controller, speed-controller and gears are derived. Figure 7.1 shows the blocks included in this model. The first block in the left side of the

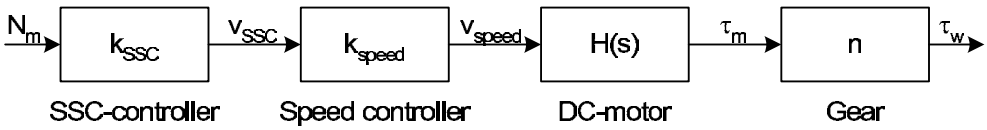

Figure 7.1: *Overview of the connections between the SSC-controller, speed-controller, DC-motor and gear.*

figure is the SSC-controller which converts a one byte signal N_m received from the computer to a PWM signal v_{SSC} which is transmitted to the speed-controller. The speed-controller converts the PWM signal v_{SSC} to a PWM powered signal v_{speed}. In this project the SSC and speed-controller are considered to be a linear gain between the value of a 8 bit binary signal N_m from the computer to the voltage v_{speed}.

First a model for the SCC-controller and the speed-controller is found. As mentioned earlier both the SSC and the speed-controller has PWM signals with finite amplitude as outputs. To make these blocks easier to model, they are considered as one black box with input N_m and output v_{speed}. It is assumed that the output can be considered as DC-voltages ranging from $-7.2\ V$ to $7.2\ V$, though they really are pulse width modulated. From now on we call this signal v'_{speed}. An experiment was performed with the SSC-controller and the speed-controller to determine the gain. Binary values was written to the SSC-controller to determine the resolution from full throttle backwards to full throttle forward. Table 7.1 shows the results of the test. Assuming linearity, the

N_m [8-bit signal]	v'_{speed} [V]	Comment
57	-7.2	Full throttle forward
127	0	-
197	7.2	Full throttle backward

Table 7.1: *Measurements of the relationship between the 8-bit signal to the servo and the voltage fed to the motor.*

model of the SSC-controller and the speed-controller is

$$N_m = v'_{speed}\kappa_m + \Delta \tag{7.1}$$

where N_m denotes the 8-bit value from the computer, v'_{speed} the output of the speed-controller, κ_m the adjustment factor and Δ the offset. The adjustments factor κ_m is calculated:

$$\kappa_m = \frac{(197 - 127)}{7.2} = 9.72 \tag{7.2}$$

and the offset is:

$$\Delta = 127 \tag{7.3}$$

Now the blocks SCC- and speed-controller in figure 7.1 can be replaced with the block seen in figure 7.2.

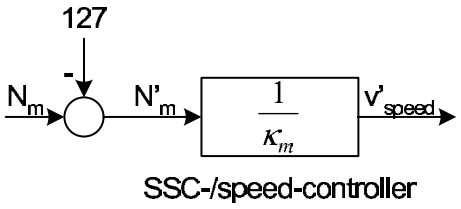

Figure 7.2: *The two blocks for SCC- and speed-controller in figure 7.1 is reduced to the one depicted above.*

Next an expression for the DC-motor is found. Figure 7.3 shows the frame of the TXT-1. In

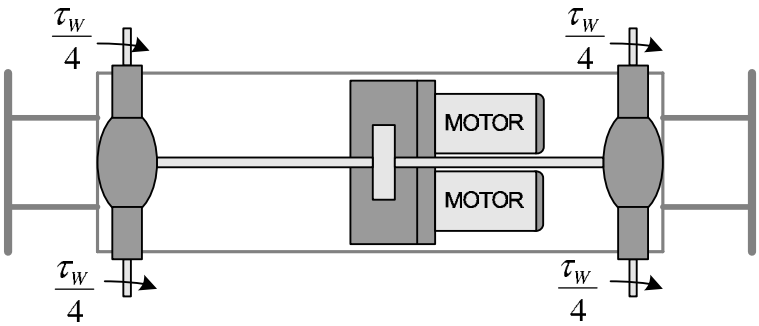

Figure 7.3: *The TXT-1 frame with DC-motors, cardan shaft and differential gears.*

the following model it is assumed that the wheels rotate with the same speed at all times, even when turning, though this is not the case in real life. This assumption is made to distribute the torque from the motors equally between the wheels. The input to the DC-motor is the voltage V_a, and the output is the resulting torque τ_w, which is equally distributed to front and rear wheel pair.

Figure 7.4 shows the electrical circuit of the motors and the gears. In the model the following assumptions are made:

- The two motors are modelled as one motor and the resulting torque τ_w from this motor is divided by two.
- Motor is linear, thus static friction is not considered.

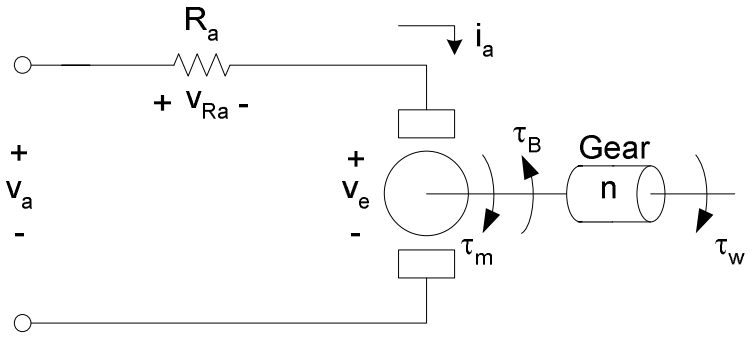

Figure 7.4: *Electrical and mechanical parts in the DC-motors*

- Inductance L_a is not considered.
- The torque τ_w is distributed equally to the wheels at all times.

The assumption about the inductance is made because the inductance is small, thus negligible. From figure 7.4 the following equations can be derived for the electrical part:

$$v_a - v_{Ra} - v_e = 0 \tag{7.4}$$

$$v_a - R_a i_a - k_e \omega_m = 0 \tag{7.5}$$

The torque put on the axle is proportional to the current i_a in the armature through a constant k_T. The torque τ_m is given by

$$\tau_m = i_a k_T \tag{7.6}$$

The linear friction provides a torque τ_B to oppose the axle rotation

$$\tau_B = B_T \omega_m \tag{7.7}$$

Thus, the resulting torque τ_w provided is given by equation 7.8. n specifies the ratio of the gears. τ_w can be written as

$$\tau_w = (\tau_m - \tau_B) n \tag{7.8}$$

Substituting equation 7.6 and 7.7 in equation 7.8 yields

$$\tau_w = (i_a k_t - B_T \omega_m) n \tag{7.9}$$

$$\Downarrow$$

$$\tau_w = \left(\frac{k_T}{R_s} (v_a - k_e \omega_m) - B_T \omega_m \right) n \tag{7.10}$$

Isolating v_a gives

$$v_a = k_e \omega_m + (\frac{\tau_w}{n} + B_T \omega_m) \frac{R_a}{k_T} \tag{7.11}$$

From equation 7.10 the block diagram in figure 7.5 can be drawn. In equation 7.10 and in

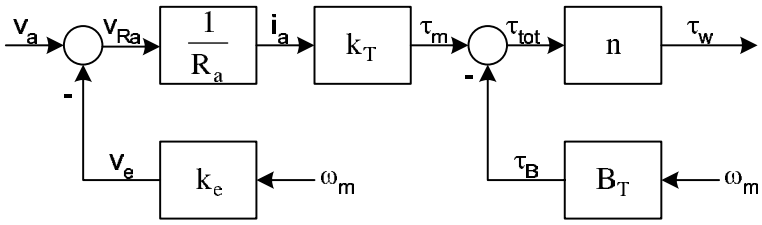

Figure 7.5: *Block diagram for the DC-motor*

figure 7.5, k_e and B_T are unknown constants. B_T is found in an experiment. This experiment is described in appendix A on page 111. Besides linear friction, the motor is considered without loss. This means that the electrical power equals the mechanical power. This assumption is expressed as follows:

$$u_a \cdot i_a = \tau_m \cdot \omega_m \tag{7.12}$$

Equation 7.12 can be written as:

$$k_e \cdot \omega_m \cdot i_a = k_T \cdot i_a \cdot \omega_m \tag{7.13}$$

$$\Downarrow$$

$$k_e = K_T \tag{7.14}$$

The last result in equation 7.14 shows that under the assumption no loss, the constants k_e equals k_T. k_T is given in the datasheet [CD, \Datablade\Johnson_HC683G.pdf]. Below all constants is shown in table 7.2.

Constant	Value
k_T	$3.9170 \cdot 10^{-3}$ Nm/A
k_e	$3.9170 \cdot 10^{-3}$ Nm/A
R_a	0.1340Ω
B_T	$18.288 \cdot 10^{-3}$ Nms/rad
n	34

Table 7.2: *Motor constants used in the model seen in figure* **??**

7.2 Servo

The SSC-controller, which will be controlling both the DC-motor and the servos for steering, accepts an 8-bit signal indicating the position of the servo as mentioned earlier. The position of the servo results in a steering angle for the wheel pair in question. The relationship between the angle and the 8-bit signal will be determined here.

Measurements of the steering angle of the front wheel pair as a function of the 8-bit signal was performed. The results are seen in table 7.3. Assuming linearity, the model reduces to a

8-bit signal	Steering angle [rad/s]
89	$-\frac{\pi}{9}$
127	0
168	$\frac{\pi}{9}$

Table 7.3: *Measurements of the relationship between the 8-bit signal to the servo and the steering angle for the front wheel pair.*

adjustment factor and offset, which can be expressed as:

$$N_f = \gamma_1 \kappa_s + \Delta \tag{7.15}$$

Where N_f denotes the 8-bit signal for the front servo, γ_1 the steering angle of the front wheel pair, κ_s the adjustment factor and Δ the offset. Accordingly the model of the rear wheel pair is similar:

$$N_r = \gamma_2 \kappa_s + \Delta \tag{7.16}$$

Using the assumption of linearity, the adjustment factor κ_s has been calculated:

$$\kappa_s = \frac{\pi}{360} \tag{7.17}$$

and the offset is:

$$\Delta = 127 \tag{7.18}$$

CHAPTER 8

Odometry Model

In order to keep track of the posture of the robot over time, a odometry model is needed. As we will see later, also the value of η and γ_1 must be determined using the odometry model.

Using inputs from the encoders placed at the two rear wheels, the odometry model shall be able to determine the posture of the robot at all times. The development and simulation of such a model will be the subject of the following.

We will use indices k and $k-1$ to distinguish between the state of variables at the present sampling instant (t_k) and the previous sampling instant (t_{k-1}), respectively. With this definition, the principle of updating the odometry can be expressed as:

$$\tilde{\xi}_k = \tilde{\xi}_{k-1} + \Delta\tilde{\xi}_k \tag{8.1}$$

Where

$$\tilde{\xi}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix}, \quad \tilde{\xi}_{k-1} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix}, \quad \Delta\tilde{\xi}_k = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{8.2}$$

The odometry ($\tilde{\xi}$) is maintained by summing up the change in posture for each time step. This leaves the main objective of the odometry model to be able to determine $\Delta\tilde{\xi}_k$.

In [Wang, 1988], a method for determining the change in posture of a two-wheeled robot drive system using only the measurements from the wheels encoders is suggested. Using this method, the posture of the middle point of the axle is described. The method can be expanded to be used here, by using the rear wheel pair (wheel 2 and 3), if they are fixed in orientation with respect to the robot frame. The method is ideal since the change in orientation θ is deducted from the encoder readings. If this wasn't possible, the change in orientation would have been evaluated using the steering angles γ. This would have been a problem, since no measurement of the steering angles are available. The only information available is the requested steering angle applied by the controller, but this is not precise due to friction of the tires.

8.1 Deriving the Model

In the following the odometry model is derived. First the change in orientation is determined, and afterwards the change in position is determined.

8.1.1 The Change in Orientation

The wheel-pair in question (the rear one) consists of wheels W_2 and W_3. The axle mid-point is called G. Encoders are placed on both wheels, thereby making measurements of the distance travelled by each wheel known. If both wheels are travelling the same distance, the problem of deriving the change in orientation $\Delta\theta$ and change in position Δx and Δy are simple. In that case there is no change in orientation, and the change in position is a matter of simple calculations. If the travelled distance for the two wheels are not the same, the problem becomes a bit more complex. Consider the situation illustrated in figure 8.1. The wheels are travelling different distances

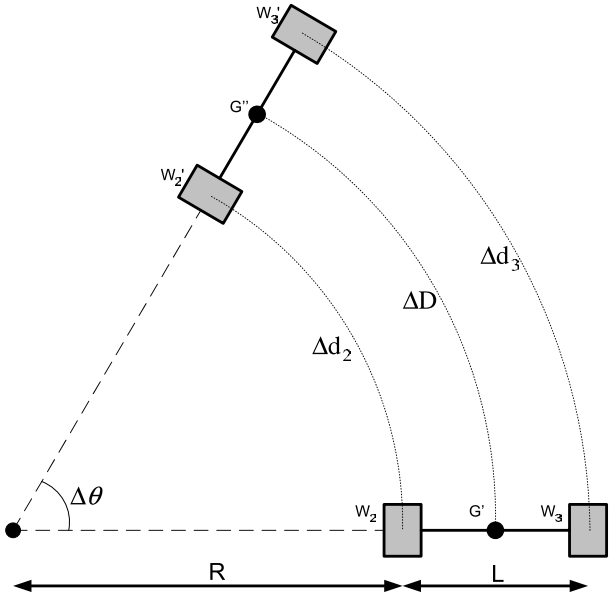

Figure 8.1: *The two-wheel drive system. The mid-axle point is moving from G' to G'', and the wheels 2 and 3 are travelling the distances Δd_2 and Δd_3 respectively.*

Δd_2 and Δd_3 resulting in a change in orientation $\Delta\theta$. For small angles $\Delta\theta$, the distances can be expressed as:

$$\Delta d_2 = R\Delta\theta \tag{8.3}$$

$$\Delta d_3 = (R + L)\Delta\theta \tag{8.4}$$

The distance travelled by the point G is found as the mean of the distances Δd_2 and Δd_3:

$$\Delta D = \frac{\Delta d_2 + \Delta d_3}{2} \tag{8.5}$$

since

$$\Delta D = \left(R + \frac{L}{2}\right)\Delta\theta = \frac{1}{2}(2R + L)\Delta\theta = \frac{1}{2}(\Delta d_2 + \Delta d_3) \tag{8.6}$$

The change in orientation is found by combining equations 8.3 and 8.4:

$$\Delta d_3 = R\Delta\theta + L\Delta\theta = \Delta d_2 + L\Delta\theta \Rightarrow \Delta\theta = \frac{\Delta d_3 - \Delta d_2}{L} \tag{8.7}$$

Thus, the change in orientation is found as the difference between the encoder measurements divided by the distance between the wheels, L. Since the calculation of the change in orientation

is based on the assumption that $\sin(\Delta\theta) = \Delta\theta$, the model is only valid for small angles $\Delta\theta$. It is therefore a requirement for the model to be accurate that the sampling time is small enough to ensure the assumption.

8.1.2 The Change in Position

Having established the change in orientation $\Delta\theta$, the next step is to determine the change in position for the robot. For that purpose, the illustation in figure 8.2 is used. At time $k-1$ the axle

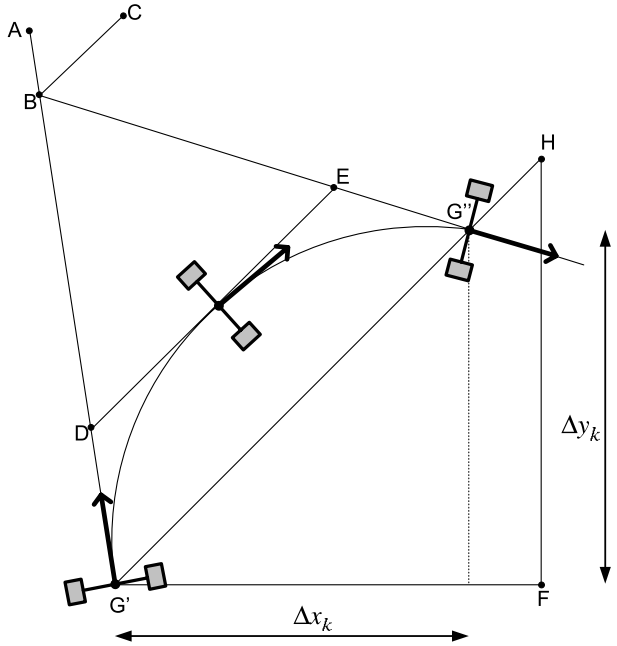

Figure 8.2: *The geometrical problem for determining the new position and orientation of the wheel pair. At time $k-1$ the wheel pair is at position G' oriented towards A. The wheel pair moves along the arc$G'G''$ towards the point G'' at time k.*

mid-point is at the point G' oriented towards the point A, thereby making $\theta_{k-1} = \angle AG'F$. The orientation at time k is given by $\theta_{k-1} + \Delta\theta_k$. The wheel pair takes a path from G' to G'' that has total arc length ΔD_k and a change in orientation of $\Delta\theta_k$. Such a path is illustrated in figure 8.2 as the arc between G' and G''. The change in orientation for the wheels is equal to the angle ABG':

$$\Delta\theta_k = \angle ABG'' \tag{8.8}$$

We seek the location of the wheel axle-midpoint at time k at location G'. For this the angle $HG'F$ and the length $|G'G''|$ must be determined. When they are determined, the position of G' can be calculated as:

$$G'' = \cos(\angle HG'F)|G'G''| \tag{8.9}$$

The angle $HG'F$ is found by inspecting the various angles. According to [Wang, 1988] it can be shown that the line BC, which is parallel to the line $G'G''$, bisects the angle ABG''. Thus:

$$\angle ABC = \angle AG'H = \frac{\Delta\theta_k}{2} \tag{8.10}$$

With this knowledge, the following is also satisfied:

$$\angle HG'F = \angle AG'F - \angle AG'H = \theta_{k-1} + \frac{\Delta\theta_k}{2} \tag{8.11}$$

The length $|G'G''|$ is unknown, since the encoder measurements represent the length $|\text{arc}G'G''|$. An expression for the relationship between the length $|G'G''|$ and the length $|\text{arc}G'G''|$ is therefore found in [Wang, 1988] under the assumption that the path $\text{arc}G'G''$ is circular:

$$\frac{|G'G''|}{|\text{arc}G'G''|} = \frac{2\left(\frac{L}{2}+R\right)\sin\left(\frac{\Delta\theta_k}{2}\right)}{\left(\frac{L}{2}+R\right)\Delta\theta_k} = \frac{\sin\left(\frac{\Delta\theta_k}{2}\right)}{\frac{\Delta\theta_k}{2}} \tag{8.12}$$

That is, the length $G'G''$ is:

$$|G'G''| = \frac{\sin\left(\frac{\Delta\theta_k}{2}\right)}{\frac{\Delta\theta_k}{2}}\Delta D_k \tag{8.13}$$

since $|\text{arc}G'G''| = \Delta D_k$. Inserting equations 8.13 and 8.11 into equation 8.9 then yields:

$$\Delta x = \frac{\sin\left(\frac{\Delta\theta_k}{2}\right)}{\frac{\Delta\theta_k}{2}}\Delta D_k \cos\left(\theta_{k-1} + \frac{\Delta\theta_k}{2}\right) \tag{8.14}$$

$$\Delta y = \frac{\sin\left(\frac{\Delta\theta_k}{2}\right)}{\frac{\Delta\theta_k}{2}}\Delta D_k \sin\left(\theta_{k-1} + \frac{\Delta\theta_k}{2}\right) \tag{8.15}$$

$$\tag{8.16}$$

Using this, regular expressions for the posture at time k based on the posture at time $k-1$ are:

$$x_k = x_{k-1} + \frac{\sin\left(\frac{\Delta\theta_k}{2}\right)}{\frac{\Delta\theta_k}{2}}\Delta D_k \cos\left(\theta_{k-1} + \frac{\Delta\theta_k}{2}\right) \tag{8.17}$$

$$y_k = y_{k-1} + \frac{\sin\left(\frac{\Delta\theta_k}{2}\right)}{\frac{\Delta\theta_k}{2}}\Delta D_k \sin\left(\theta_{k-1} + \frac{\Delta\theta_k}{2}\right) \tag{8.18}$$

$$\theta_k = \theta_{k-1} + \Delta\theta_k \tag{8.19}$$

The factor

$$\frac{\sin\left(\frac{\Delta\theta_k}{2}\right)}{\frac{\Delta\theta_k}{2}} \tag{8.20}$$

introduces a problem. For $\Delta\theta_k = 0$ the expressions for Δx_k and Δy_k in equations 8.14 and 8.15 are not defined. This means that straight motion is not supported, and this is not acceptable. As seen in figure 8.3, it is a property of equation 8.20 that

$$\lim_{\Delta\theta_k \to 0}\left[\frac{\sin\left(\frac{\Delta\theta_k}{2}\right)}{\frac{\Delta\theta_k}{2}}\right] = 1 \tag{8.21}$$

This means, that for small angles $\Delta\theta_k$, the factor will be negligible. Therefore the factor will be disregarded in this project if $\Delta\theta_k = 0$.

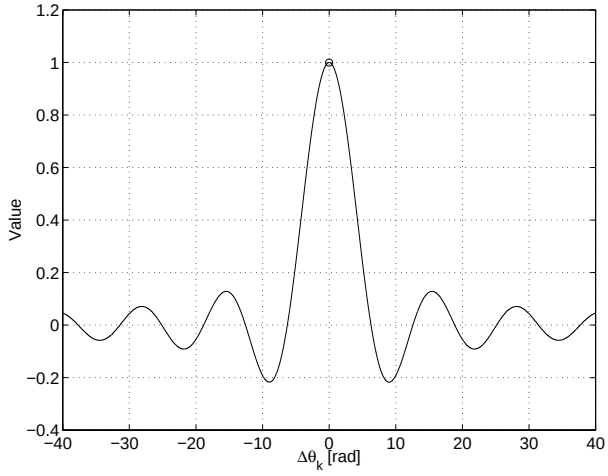

Figure 8.3: *Plot of the adjustment factor from equation 8.20 as a function of $\Delta\theta_k$. It is seen that the function converges to 1 as $\Delta\theta_k$ gets closer to zero. Note that the function is not defined in $\Delta\theta_k = 0$.*

8.1.3 Relationship between G and P

Having determined the change in posture for the point G the last step is to calculate the resulting change in posture for the robot center P. The situation is illustrated in figure 8.4. Denoting$\tilde{\xi}_P$ as

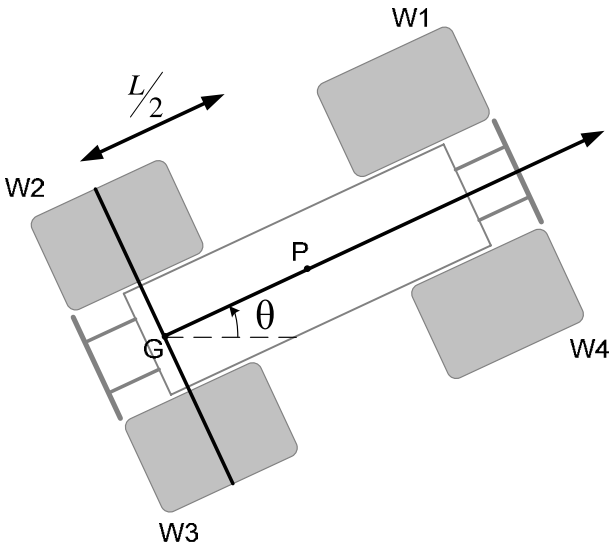

Figure 8.4: *The positions of the points G and P on the robot.*

the odometry with respect to the point P and $\tilde{\xi}_G$ with respect to G the following is true:

$$\tilde{\xi}_P = \tilde{\xi}_G + \begin{bmatrix} \frac{L}{2}\cos\theta \\ \frac{L}{2}\sin\theta \\ 0 \end{bmatrix} \tag{8.22}$$

8.2 Determining η and γ_1 Using the Odometry Model

Having found expressions for x, y, θ and their derivatives, the final step is to determine expressions for η, ζ_1 and ζ_2, since they must be known in order to able to implement the controllers designed later. The variables η, ζ_1 and ζ_2 must be known at each sampling instant for use in feedback. ζ_2 is zero if the orientation of the wheels are fixed with respect to the robot frame. The value of ζ_1 can be calculated, if the angle γ_1 is known for the present and previous sampling instant:

$$\zeta_1 = (\gamma_1)_k - (\gamma_1)_{k-1} \tag{8.23}$$

since

$$\zeta_1 = \dot{\gamma}_1 \tag{8.24}$$

The expressions for these are found in appendix F.4, and therefore only the results are stated here:

$$\eta = -\frac{\dot{x}}{\ell(-\cos\theta\cos(\alpha_1-\gamma_1)+\cos\theta\cos\gamma_1\cos\alpha_2+\sin\theta\sin\gamma_1\cos\alpha_2)} \tag{8.25}$$

$$\gamma_1 = \arctan\left\{\frac{-\ell\sin\theta(\cos\alpha_1-\cos\alpha_2)\dot{\theta}}{\dot{y}+\ell\sin\theta\sin\alpha_1\dot{\theta}-\cos\theta K}\right\} \tag{8.26}$$

8.3 Simulations

To be able to verify the developed odometry model, simulations were performed. Encoder-measurements were constructed ”at hand”, since it wasn’t possible to use simulated values from the kinematic model. Plots were made to reveal the results. The constructed input from the encoders are chosen, so the trajectory of the robot will contain three types of motion:

- Straight motion with both wheels travelling the same distance.
- Turning motion, with only one wheel moving.
- Turning motion with both wheels moving.

The simulation is run for 1000 seconds. It is the intention to simulate the trajectory illustrated in figure 8.5. The change in encoder measurements as a function of time has been found by simple

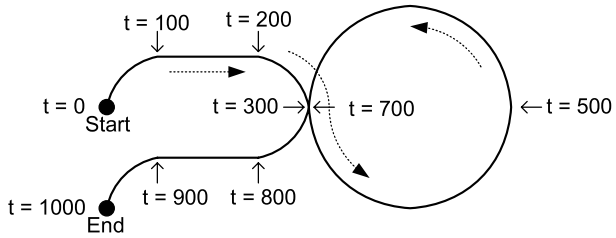

Figure 8.5: *The trajectory the robot will follow in the simulation.*

calculations ”in hand”, and are stated in table 8.1 and also shown in figure 8.6.

The corresponding outputs from the odometry model is seen in figure 8.7 (the position) and in figure 8.8 (the orientation). The robot position is as expected, with the robot moving from the

Time interval [s]	Direction	W2 increment per sample [m]	W3 increment per sample [m]
0-100	Right	0.00443	0
101-200	Straight	0.00443	0.00443
201-300	Right	0.00443	0
301-700	Left	0.00443	0.00886
701-800	Right	0.00443	0
801-900	Straight	0.00443	0.00443
901-1000	Left	0.00443	0

Table 8.1: *The sample intervals, type of motion and increments in encoder values used to construct the inputs used for the simulation of the odometry model.*

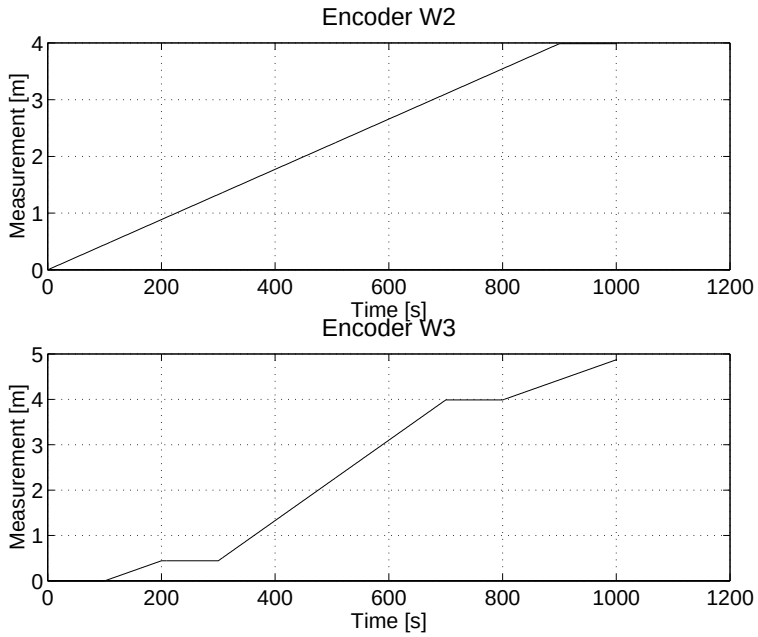

Figure 8.6: *The inputs from the encoders used for the simulation. At the top the encoder measurements over time from wheel 2, and below the measurements from wheel 3.*

starting point (the square) to the end point (the triangle) as indicated by the arrows. The orientation is also as expected. Since the angle is restricted:

$$-\pi \leq \theta \leq \pi \tag{8.27}$$

it sometimes changes from -180° to 180°.

8.4 Summary

In this section an odometry model was derived taking inspiration from [Wang, 1988]. The model describes the robot posture over time, using only the encoder measurements from the two rear wheels. The model was simulated using constructed encoder-measurements, and the results were as expected.

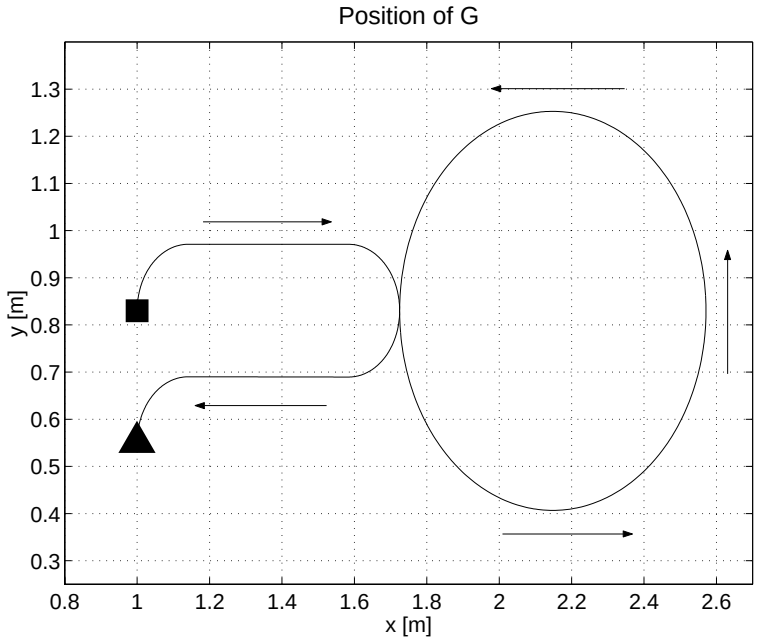

Figure 8.7: *The position of the point G as found by the odometry model. As expected the robot is moving along the solid line as indicated by the arrows starting at the square and ending at the triangle.*

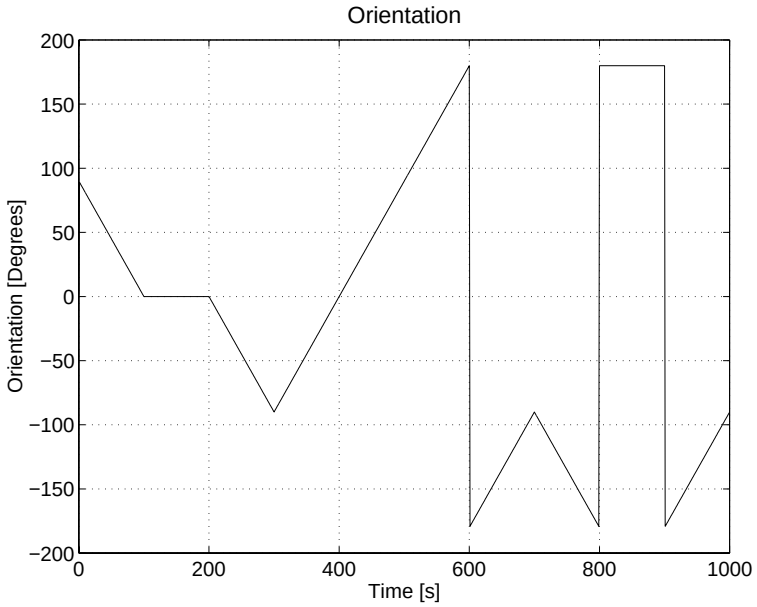

Figure 8.8: *The change of orientation during the simulations. Since the orientation is restricted to be in the interval $[-\pi; \pi]$ the angle on occasion changes from* -180° *to* 180°.

Part III

Control

CHAPTER 9

Control by Feedback Linearization

In this chapter the first of two control algorithms for trajectory tracking is designed. The control design is based on feedback linearization. To reduce the complexity of the control problem, a partial linearization of the dynamical model is performed in section 9.1. In section 9.2 the steps of feedback linearization for a general system are described briefly. In the section 9.3 a change of variables is performed on the nonlinear model for the robot, and the control laws are applied in section 9.4.

9.1 Partial Linearization

The purpose of this section is to perform a linearization of the dynamical model to ease the design of a controller. In chapter 4 a nonlinear state space model for the robot was found:

$$H(\gamma)\dot{u} = -f(\gamma, u) + F(\gamma)\tau \tag{9.1}$$

$$\dot{q} = S(q)u \tag{9.2}$$

Recapitulating that q is the vector of generalized coordinates defined as:

$$q = \begin{bmatrix} \dot{\xi} \\ \dot{\gamma} \\ \dot{\psi} \end{bmatrix} \tag{9.3}$$

And u is the input vector to the kinematic model:

$$u = [\eta \quad \zeta_1 \quad \zeta_2 \quad \zeta_3 \quad \zeta_4]^T \tag{9.4}$$

A common approach in the design for a controller to a nonholonomic wheeled robot is to perform a partial linearization of the dynamics in equation 9.1 as described in e.g. [Oriolo, 1995], [Champion, 1996] and [Dimon, 2002]. The idea is to determine τ based on a desired value for $\dot{u}$. That is, the output from a designed controller would be $\dot{u}$ instead of τ, hereby reducing the model to:

$$\dot{u} = v \tag{9.5}$$

$$\dot{q} = S(q)u \tag{9.6}$$

where v are the accelerations determined by the controller. An expression for τ based on the value of v can be found by rearranging equation 9.7 and substituting $\dot{u}$ with v:

$$H(\gamma)v = -f(\gamma, u) + F(\gamma)\tau \tag{9.7}$$

$$\Downarrow$$

$$\tau = F^{\dagger}(\gamma)\left(H(\gamma)v + f(\gamma, u)\right) \tag{9.8}$$

where $\dagger$ denotes a pseudo-inverse, since $F(\gamma)$ isn't a square matrix. The change in the structure of the control design is illustrated in figure 9.1.

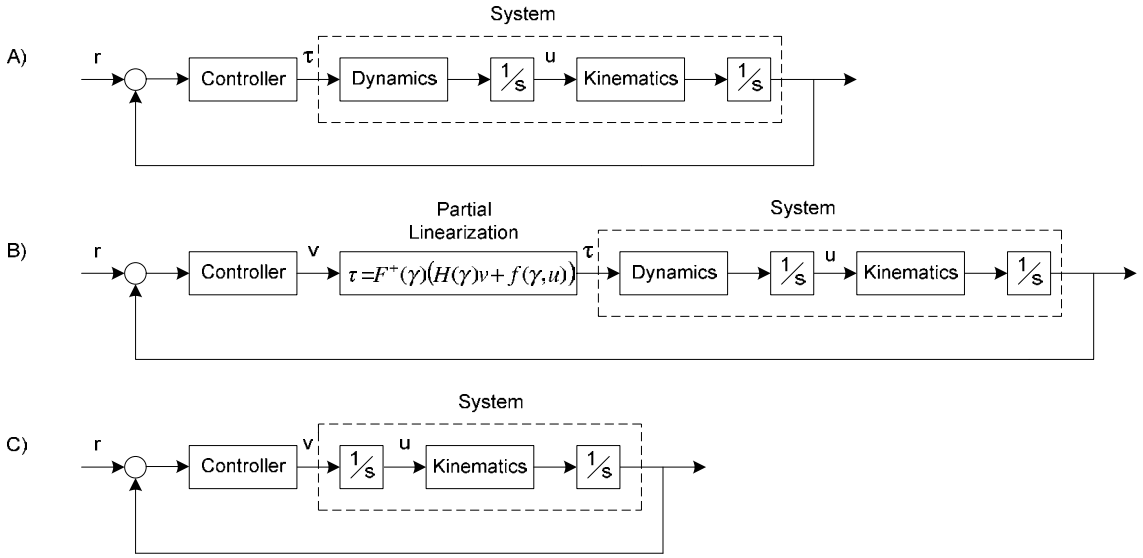

Figure 9.1: *A) The control problem before partial linearization is applied. B) How partial linearization is applied. C) The reduced system from the controllers point of view after having applied the partial linearization.*

9.2 General Equations for Feedback Linearization

The equations presented in this section are not specific for the robot. The purpose is to clarify the steps performed in section 9.3. Consider a general nonlinear system:

$$\dot{\chi} = f(\chi) + G(\chi)v \tag{9.9}$$

$$y = h(\chi) \tag{9.10}$$

with input v, system states χ and output y. With feedback linearization the question is whether there exists a control law:

$$v = \beta^{-1}(z)w + \alpha(z) \tag{9.11}$$

and a change of variables:

$$z = T(\chi) \tag{9.12}$$

which together transforms the nonlinear system into an equivalent linear system:

$$\dot{z} = Az + Bw \tag{9.13}$$

The linear system in equation 9.13 with input w can be stabilized with a linear controller if the system is controllable.

An important note. Once the change of variables in equation 9.12 has been performed on the system, it is required that the system can be written in form:

$$\dot{z} = Az + B\beta(z)\left(v - \alpha(z)\right) \tag{9.14}$$

in order for the feedback linearization to be applicable [Khalil, 2002].

9.3 Change of Variables

A change of variables is performed on the system to obtain the structure of equation 9.14. Using partial linearization and considering the posture kinematic model from section 5.4, the system model is reduced to:

$$\dot{u}_{pk} = v \tag{9.15}$$

$$\dot{q}_{pk} = S(q_{pk})u_{pk} \tag{9.16}$$

As a consequence of the posture kinematic model the vector v is reduced to:

$$v = [\dot{\eta} \quad \dot{\zeta}_1 \quad \dot{\zeta}_2]^T \tag{9.17}$$

Before a change of coordinates is applied, the system model is written into the same form as equation 9.9:

$$\dot{\chi} = \begin{bmatrix} \dot{u} \\ \dot{q}_{pk} \end{bmatrix} = \begin{bmatrix} 0 \\ S(q_pk)u_pk \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} v \tag{9.18}$$

With inspiration from [Micaelli, 1996] and [Dimon, 2002] a change of variables is introduced as:

$$z = T(\chi) = \begin{bmatrix} \xi \\ \dot{\xi} \end{bmatrix} \tag{9.19}$$

Remembering that ξ describes the posture of the robot:

$$\xi = [x \quad y \quad \theta]^T \tag{9.20}$$

The choice is a result of the fact that we wish to minimize the error between a time-varying trajectory $r(t)$:

$$r(t) = [\xi_{ref}(t) \quad \dot{\xi}_{ref}(t)]^T, \quad \forall\, t \tag{9.21}$$

and the system states $[\xi \ \dot{\xi}]^T$. With equation 9.19 the control design takes the structure given in figure 9.2:

With the known change of variables, the next step is to write the system in the form of equation 9.14 and thereby verifying that feedback linearization is applicable. The system written on the form of equation 9.14 is found with use of equation 5.5 where an expression for$\dot{\xi}$ is given:

$$\begin{aligned} \dot{z} &= \begin{bmatrix} \dot{\xi} \\ \ddot{\xi} \end{bmatrix} = \begin{bmatrix} \dot{\xi} \\ \frac{d}{dt}\left(\dot{\xi}\right) \end{bmatrix} = \begin{bmatrix} \dot{\xi} \\ \frac{d}{dt}\left(R^T(\theta)\Sigma(\gamma)\eta\right) \end{bmatrix} \\ &= A_s \begin{bmatrix} \xi \\ \dot{\xi} \end{bmatrix} + B_s \left(\frac{d}{dt}\left(R^T(\theta)\Sigma(\gamma)\eta\right)\right) \end{aligned} \tag{9.22}$$

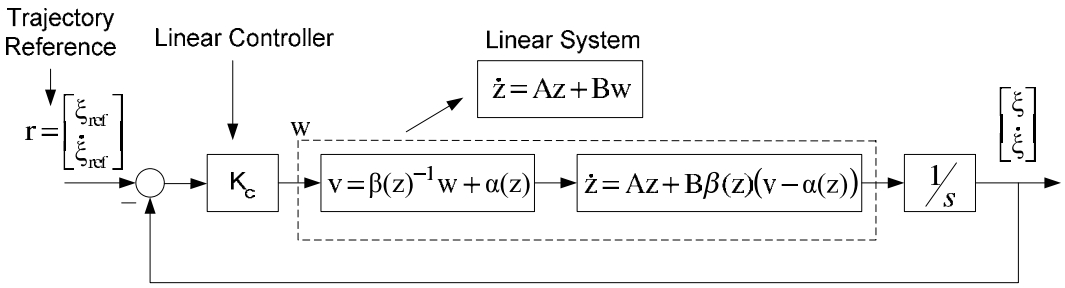

Figure 9.2: *The control law for v cancels the nonlinear term in the system. Thus making it possible to use a linear controller, which drives the output equal to the reference r.*

Where

$$A_s = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \quad \text{and} \quad B_s = \begin{bmatrix} 0_{3\times3} \\ I_{3\times3} \end{bmatrix} \tag{9.23}$$

Comparing 9.14 with 9.22 it becomes clear that:

$$\beta(z)\left(v - \alpha(z)\right) = \frac{d}{dt}\left(R^T(\theta)\Sigma(\gamma)\eta\right) \tag{9.24}$$

Hence for the two terms $\alpha(z)$ and $\beta(z)$ to be found, the term $\frac{d}{dt}\left(R^T(\theta)\Sigma(\gamma)\eta\right)$ must be calculated:

$$\frac{d}{dt}\left(R^T(\theta)\Sigma(\gamma)\eta\right) = \dot{R}^T(\theta)\Sigma(\gamma)\eta + R^T(\theta)\left(\Sigma(\gamma)\dot{\eta} + \dot{\Sigma}(\gamma)\eta\right) \tag{9.25}$$

The first term on the right hand side of equation 9.25 is independent of v and therefore contains α. The remaining two terms in the bracket on the right hand side of equation 9.25 must be the product of β and v. v is found in the first term in the bracket in the form of $\dot{\eta}$. The other two elements $\dot{\zeta}_1$ and $\dot{\zeta}_2$ in the vector v should therefore be found in the second term in the bracket of equation 9.25. Remembering that $\dot{\gamma} = \zeta$, expressions for $\alpha(z)$ and $\beta(z)$ is found:

$$\begin{aligned} -\beta(z)\alpha(z) &= \dot{R}^T(\theta)\Sigma(\gamma)\eta & (9.26)\\ \beta(z)v &= R^T(\theta)\left(\Sigma(\gamma)\dot{\eta} + \dot{\Sigma}(\gamma)\eta\right) \\ &= R^T(\theta)\beta_1(z)\acute{v} & (9.27) \end{aligned}$$

$$\Downarrow$$

$$\begin{aligned} \alpha(z) &= -\beta^{-1}(z)\dot{R}^T(\theta)\Sigma(\gamma)\eta & (9.28)\\ \beta(z) &= R^T(\theta)\beta_1(z), \quad \text{for} \quad v = \acute{v} & (9.29) \end{aligned}$$

Where:

$$\beta_1(z) = \begin{bmatrix} l_1 cos(\gamma_2)cos(\alpha_1-\gamma_1) & \big(l_1 cos(\gamma_2)sin(\alpha_1-\gamma_1) & \big(-l_1 sin(\gamma_2)cos(\alpha_1-\gamma_1) \\ -l_2 cos(\gamma_1)cos(-\alpha_2+\gamma_2) & l_2 sin(\gamma_1)cos(-\alpha_2+\gamma_2)\big)\eta & l_2 cos(\gamma_1)sin(-\alpha_2+\gamma_2)\big)\eta \\ \\ l_1 sin(\gamma_2)cos(\alpha_1-\gamma_1) & \big(l_1 sin(\gamma_2)sin(\alpha_1-\gamma_1) & \big(l_1 cos(\gamma_2)cos(\alpha_1-\gamma_1) \\ -l_2 sin(\gamma_1)cos(-\alpha_2+\gamma_2) & -l_2 cos(\gamma_1)cos(-\alpha_2+\gamma_2)\big)\eta & l_2 sin(\gamma_1)sin(-\alpha_2+\gamma_2)\big)\eta \\ \\ sin(\gamma_1-\gamma_2) & cos(\gamma_1-\gamma_2)\eta & -cos(\gamma_1-\gamma_2)\eta \end{bmatrix} \tag{9.30}$$

and $\acute{v} = [\dot{\eta} \quad \zeta_1 \quad \zeta_2]^T$ (9.31)

As seen from equation 9.29 a problem occurs since $v = [\dot{\eta} \ \dot{\zeta}_1 \ \dot{\zeta}_2]^T$ and $\acute{v} = [\dot{\eta} \quad \zeta_1 \quad \zeta_2]^T$ isn't equal. In other words the input v to the model in equation 9.7 isn't equal to the input $\acute{v}$ given from a designed control law in the form of equation 9.11. This raises the following question: Can the input v be determined from $\acute{v}$? If so, a control law for v can be found from the control law for $\acute{v}$. The answer to the question is yes, since v can be determined by differentiation of last elements in $\acute{v}$.

With $\alpha(z)$ and $\beta(z)$ as given in equation 9.28 and 9.29, the system can be written in the form of equation 9.14, which was the requirement for the use of feedback linearization.

9.4 Control Law

The next step is to apply the control law that cancels out the nonlinear terms in system. The control law is:

$$\acute{v} = \beta^{-1}(z)w + \alpha(z) \tag{9.32}$$

where w is the new input to the linear system. With this control law the system reduces to an equivalent controllable linear system:

$$\dot{z} = A_s z + B_s w \tag{9.33}$$

The fact that equation 9.33 is controllable can be seen from the controllability matrix ϑ.

$$\vartheta = \begin{bmatrix} B_s & A_s B_s .... A_s^{n-1} B_s \end{bmatrix} \tag{9.34}$$

where n = 6 is the order of the system in equation 9.33. For the system to be controllable ϑ must have full rank:

$$rank(\vartheta) = n. \tag{9.35}$$

Calculating ϑ in MATLAB with the function `ctrb(As;Bs)` shows that $rank(\vartheta) = 6$ and hence the system is controllable.

Equation 9.33 is a linear state space model. Such a system can be stabilized with a full state feedback control law [Emami-Naeini, 1994]:

$$w = K_c z \tag{9.36}$$

K_c is a 3×6 matrix of gain constants. The values of the gains are calculated on the basis of the system-matrices A_s, B_s and on the choice of pole placement with the MATLAB function `place()`. With the function `place()` the placement of the poles is free of choice. Though it is important to remember that the gains grow as the poles are moved further away from Origo. Large gains may cause the motors to saturate. The pole placement issue is treated in section 9.5.

The last step in the control design is the introduction of the reference r. To avoid a steady-state error in the system, the reference should be introduced as given in equation 9.37 [Emami-Naeini, 1994]:

$$w = -K_c z + (N_u + K_c N_x) r \tag{9.37}$$

The gain matrices N_u and N_x are found from their relation to the system-matrices:

$$\begin{bmatrix} A_s & B_s \\ C_s & 0 \end{bmatrix} \begin{bmatrix} N_x \\ N_u \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix} \tag{9.38}$$

$$\Downarrow$$

$$\begin{aligned} N_x &= C_s^{-1} \\ N_u &= -B_s^{-1} A_s C_s^{-1} \end{aligned} \tag{9.39}$$

$$\Downarrow$$

$$\begin{aligned} N_x &= I_{6\times 6} \\ N_u &= 0_{3\times 6} \end{aligned} \tag{9.40}$$

Hence reducing equation 9.37 to:

$$w = K_c (r - z) \tag{9.41}$$

Inserting equation 9.41 in equation 9.32 and the control law becomes:

$$\acute{v} = \beta^{-1}(z) K_c (r - z) + \alpha(z) \tag{9.42}$$

9.5 Pole Placement

Various combinations for the placement of the poles were tested in a MATLAB simulation. An important observation was that in the resulting matrix K_c, gains and zeros were always placed on the same positions, independent of the choice of pole placement. The positions of gains and zeros in K_c is shown in equation 9.43:

$$K_c = \begin{bmatrix} k_{11} & 0 & 0 & k_{14} & 0 & 0 \\ 0 & k_{22} & 0 & 0 & k_{25} & 0 \\ 0 & 0 & k_{33} & 0 & 0 & k_{36} \end{bmatrix} \tag{9.43}$$

Considering equation 9.41 is becomes clear that the gains in the first three columns of K_c determines the influence of the feedback of the error in the posture $\xi_{ref} - \xi$ since:

$$w = K_c(r - z) = K_c = \begin{bmatrix} \xi_{ref} - \xi \\ \dot{\xi}_{ref} - \dot{\xi} \end{bmatrix} \tag{9.44}$$

The last three columns determines the influence of the derivative of the error in the posture. Because the goal is to minimize the error of the posture and not the derivative, the gains in the first three columns must be dominant. With this observation a different approach was used to determine the placement of the poles. The the gains in K_c was chosen without consideration to the poles. Appropriate values for the gains was found by trial and error in a MATLAB simulation. An inverse calculation from gains to poles was then performed to ensure that the poles were stable. Appropriate values for the gains was found to be:

$$K_c = \begin{bmatrix} 0.05 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0.05 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0.05 & 0 & 0 & 0.001 \end{bmatrix} \tag{9.45}$$

To check and see if these gains result in stable poles a reversed calculation was made:

$$P_c = eig(A_s - B_s K_c) \tag{9.46}$$

$$= \begin{bmatrix} -0.0005 + 0.2236i \\ -0.0005 - 0.2236i \\ -0.0005 + 0.2236i \\ -0.0005 - 0.2236i \\ -0.0005 + 0.2236i \\ -0.0005 - 0.2236i \\ -0.0005 + 0.2236i \\ -0.0005 - 0.2236i \end{bmatrix} \tag{9.47}$$

Since all poles are in the left half side of the complex plane the poles are stable.

9.6 Simulation

To verify the control design a simulation was made in MATLAB. The trajectory $r(t)$ used in the simulation was defined as:

$$r(t) = [t, \quad t, \quad \frac{\pi}{4}, \quad 1, \quad 1, \quad 0]; \tag{9.48}$$

The constraints on the angle of the wheels wasn't considered in the control design. So, to ensure that the controller behaved as desired, no constraints was applied to the angle of the wheels in the first simulations. It was found that with the designed controller, the robot was able to follow a trajectory, if the steering angles in the front and rear isn't equal to $\pm\frac{\pi}{2}$ rad. In the occurrence of such a situation, a singularity is experienced in the kinematic model, causing the robot to stand still in the simulation, see figure 9.4. The singularity is more precisely found in the vector $\Sigma(\gamma)$

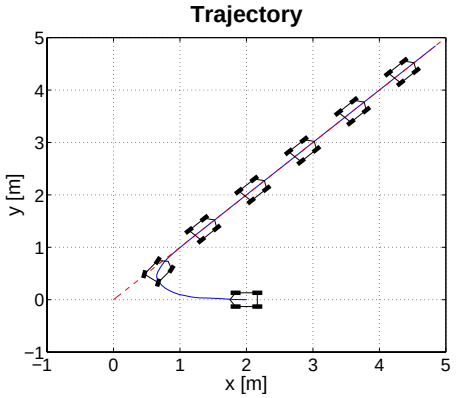

Figure 9.3: *The robot starts from a position in* $(-2, 0)$ *and with orientation* π *rad. From here the robot converges to and follows the trajectory as desired.*

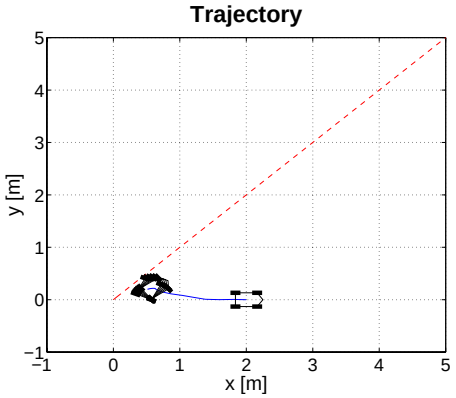

Figure 9.4: *The robot starts from a position in* $(-2, 0)$ *and with orientation* 0 *rad. This is a situation where the model fails because the front and rear steering angles are* $\frac{\pi}{2}$ *and hence a singularity is experienced.*

which becomes zero when $\gamma_1 = \gamma_2 = \pm\frac{\pi}{2}$. Hence the change in the posture $\dot{\xi}$ becomes zero:

$$\dot{\xi} = R^T(\theta)\Sigma(\gamma)\eta = 0 \quad \text{for } \gamma_1 = \gamma_2 = \pm\frac{\pi}{2} \tag{9.49}$$

A solution to the problem of the singularity will not be pursued. The reason this being, that the occurrence of the singularity is an impossibility in real life due to the constraints on the steering angles. However, introducing the constraints in the simulation doesn't come without complications. A situation with the same starting conditions as in figure 9.4, but with constraints is shown in figure 9.5. Here it becomes clear that the constraints can force the robot into a state where it is

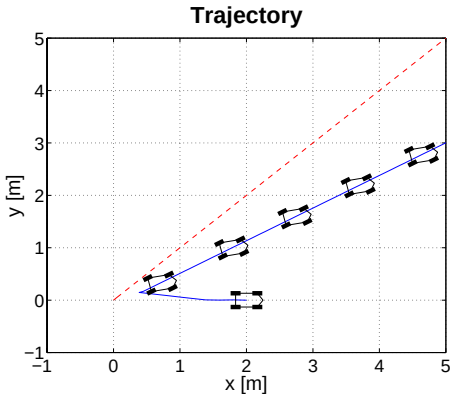

Figure 9.5: *Same situation as in figure 9.4, but with constraints on the wheels. Because of the constraints the robot doesn't experience any singularities but is still unable to follow the trajectory.*

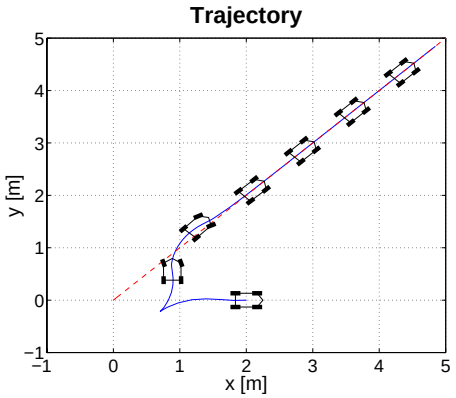

Figure 9.6: *A solution to the constraint problem. The front wheels are controlled and the rear wheels are locked.*

unable to change its orientation. The controller wishes to steer the front wheels in a larger angle than the rear wheels. But because of the constraints, the angle of the front and rear wheels becomes the same, and thereby causing the somewhat lateral movement pictured in figure 9.5. Simulations shows that the controller works fine once the orientation of the robot is aligned with the trajectory.

Or in other words, the problem is to ensure the right orientation of the robot without ending up in a situation where the steering angle of both front and rear wheels are in max. or min. One way to avoid such a situation is to lock the rear wheels and only control the front wheels. A simulation of this solution is shown in figure 9.6. Once the orientation is aligned with the trajectory, the control of the rear wheels can be reapplied. This kind of solution is a hybrid controller with two states. The design of a hybrid controller is therefore the subject of the following section.

9.7 Hybrid Control

In this section a hybrid automaton is derived. The definition for a hybrid automaton is found in appendix B. What the purpose of the hybrid automaton really comes down to, is a directed graph which shows, how the hybrid controller finds and tracks the trajectory. The derived hybrid automaton is therefore simplified in the sense that some of the elements of the seven tuple is described instead of defined mathematically. This is done to keep focus on the problem at hand rather than deriving mathematically correct definitions.

The continuous dynamics of the hybrid automaton is considered as the posture of the robot over time $\xi(t)$ as described in appendix B. In the following $\xi(t)$ will be denoted as ξ for simplicity. The differential equations describing the evolvement of ξ is equal to the closed loop controller for the robot. The controller in the close loop will change depending on the state of the hybrid automaton.

9.7.1 Finding the States

The tasks of the hybrid automaton can be divided in two. One is to bring the robot close to the trajectory, and the other is to track the trajectory once close to it. Close to the trajectory is defined as an error between the posture of the robot and any given posture on the trajectory which is less than a threshold Δ:

$$\Delta \geq |\xi - \xi_{ref}| \tag{9.50}$$

where $\Delta = [\Delta_x \ \ \Delta_y \ \ \Delta_\theta]^T$. When the robot is close to the trajectory the front and rear wheels are controlled as originally intended. Meaning that the change of steering angles for the wheels are given as $\dot{\gamma}_1 = \zeta_1, \dot{\gamma}_1 = \zeta_2$ where ζ_1 and ζ_2 is determined from equation 9.42. This is implemented in the hybrid automaton as a state l_1.

To bring the robot close to the trajectory several approaches to the control of the wheels were available. The problem to be solved here, is to avoid situations, where the front and rear wheels end up in the same angle and is unable to align the orientation of the robot with the trajectory. This problem was illustrated in figure 9.5. However, solutions involving design of new controllers are undesirable. The reason for this being that a new control design is time consuming. Instead a solution based on the existing control design is seeked.

One way to solve the problem is to lock the rear wheels and control the front wheels ($\dot{\gamma}_1 = \zeta_1, \dot{\gamma}_2 = 0$) as it was illustrated in figure 9.6. Another is to use the same control signal for the front and rear wheels but with opposite sign ($\dot{\gamma}_1 = \zeta_1, \dot{\gamma}_2 = -\zeta_1$) thereby giving the robot the ability

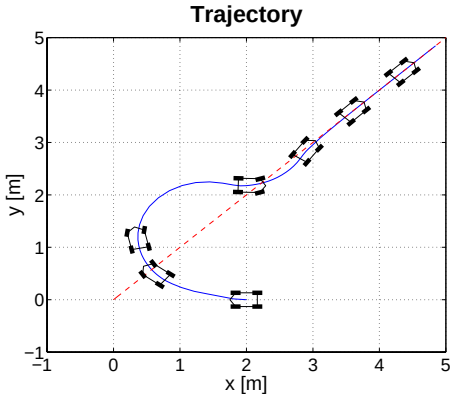

Figure 9.7: *Control is only applied to the front wheels and the rear wheels are locked.*

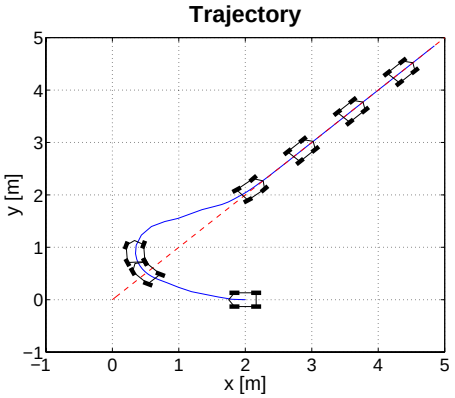

Figure 9.8: *Control is applied to both front and rear wheels. The rear wheels are controlled with the same signal as the front wheels but with opposite sign.*

to make sharper turns, see figure 9.8. Based on results from simulations it was chosen to use the latter solution. In the hybrid automaton this solution is implemented as a state $\frac{1}{2}$.

Though the performance in the ability to move close to trajectory is increased with the state $\frac{1}{2}$, situations may still occur, where the robot is unable to align the orientation of the robot with the trajectory. Situations where the robot isn't aligned can occur if the error in orientation $\theta - \theta_{ref}$ is greater than $\frac{\pi}{4}$. Here the robot may end up in a situation where the orientation of the robot is perpendicular to the point where it should converge towards. From here the robot is unable to determine, if it should move back or forth and the velocity η becomes zero. The problem is perhaps best illustrated, when the robot tries to converge to a static posture, see figure 9.9 and 9.10. Figure

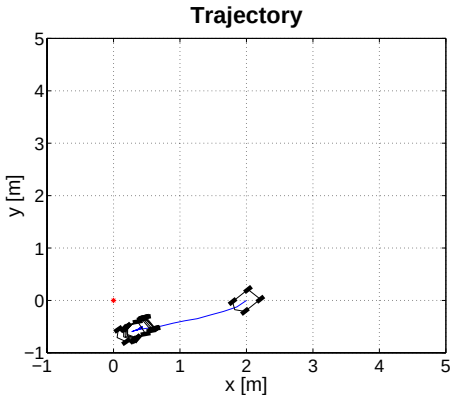

Figure 9.9: *From its starting posture in* $\xi = (2, 0, \frac{\pi}{4})$ *the robot tries to reach the posture* $(0, 0, \frac{\pi}{4})$.

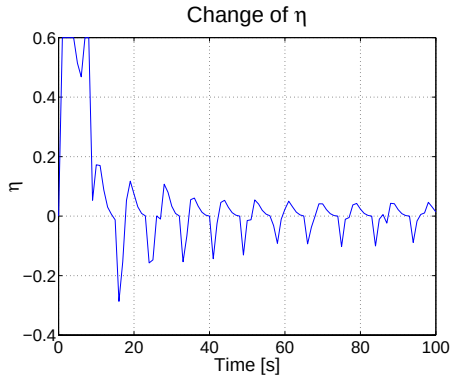

Figure 9.10: *As the orientation of the robot becomes perpendicular to the reference point, the value of* η *becomes zero.*

9.9 is actually a good example of why a different controller is needed for posture stabilization. The problem in trajectory tracking arises, if the orientation of the robot is perpendicular to the reference point (x_{ref}, y_{ref}), and the trajectory evolves in such a way that this situations remains. For a feasible trajectory, this situation can be detected by examination of the error in the orientation $\theta - \theta_{ref}$. If a feasible trajectory evolves in a way, where the orientation of a robot remains perpen-

dicular to the reference point (x_{ref}, y_{ref}) then $\theta - \theta_{ref}$ must be $\pm\frac{\pi}{2}$. A solution to the problem is found by forcing a constant value for η on the system, if $\theta - \theta_{ref}$ is greater than a threshold. The threshold is set to $\pm\frac{\pi}{4}$. The reason why the threshold isn't closer to $\pm\frac{\pi}{2}$ is because the robot slows down as it reaches $\pm\frac{\pi}{2}$ which is undesirable. With this solution the task of bringing the robot close to the trajectory is divided into two states *l*2 and *l*3. The difference between the two states being that η is a constant in the state *l*3.

9.7.2 The Directed Graph

The transition between the states in the hybrid automaton is made on the basis of the threshold Δ and the error in the orientation $\theta - \theta_{ref}$ which therefore are used in the invariant equations and as guards. A simplified directed graph for the hybrid automaton is shown in figure 9.11. In the directed graph the differential equation for the evolvement of $\xi(t)$ isn't presented. To keep focus, only the change in the controller for each state is presented. With the hybrid automaton the robot

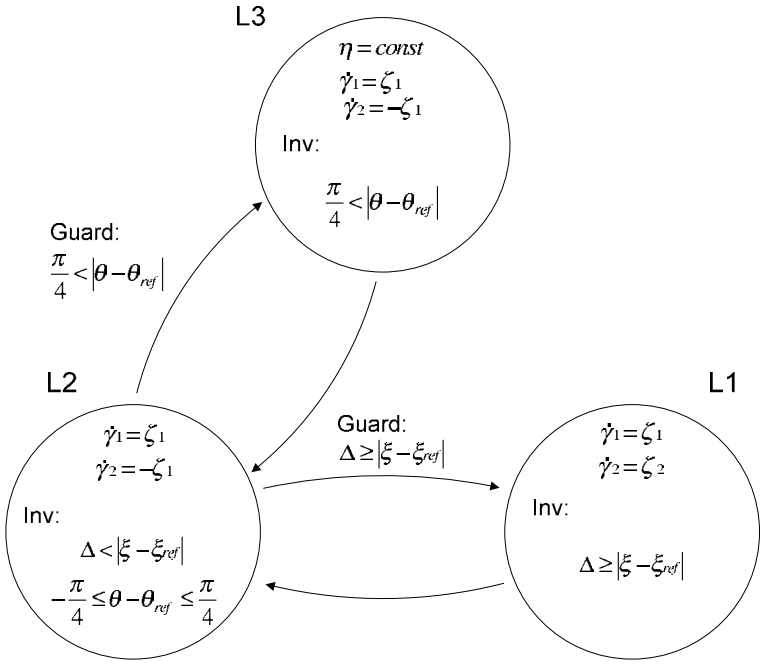

Figure 9.11: *Directed graph for the hybrid automaton.*

is able to move close to and follow the trajectory from any given start orientation. In figure 9.12, 9.13, 9.14 and 9.15 examples are shown where the robot moves close to and follows a trajectory. The robot has the same starting position (0,2) in the four figures, but different orientations. The performance of the controller with a more challenging trajectory is seen in figure 9.16, where the trajectory is a sinus-curve defined as:

$$r(t) = [t, sin(t) + 4, -cos(t), 1, -cos(t), -sin(t)]^T \tag{9.51}$$

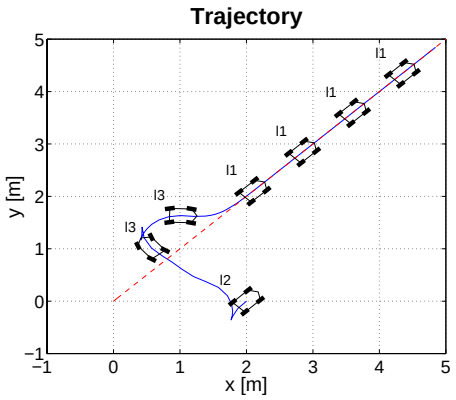

Figure 9.12: *The robot starts with an orientation of $\frac{\pi}{4}$ rad.*

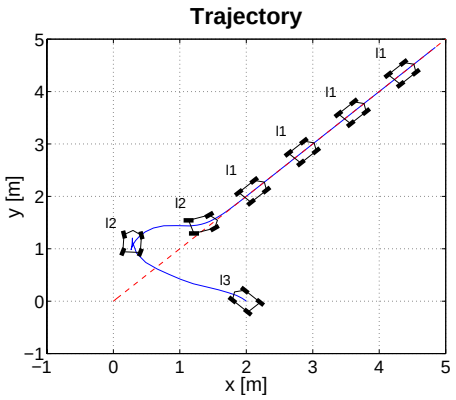

Figure 9.13: *The robot starts with an orientation of $\frac{3\pi}{4}$ rad.*

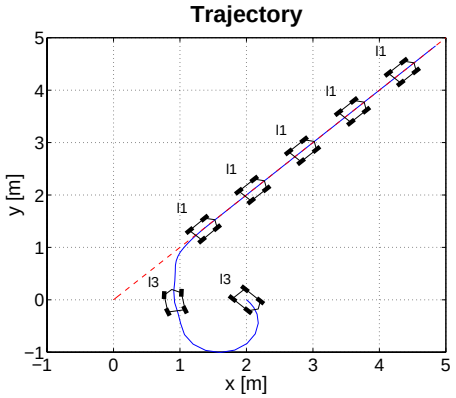

Figure 9.14: *The robot starts with an orientation of $\frac{-\pi}{4}$ rad.*

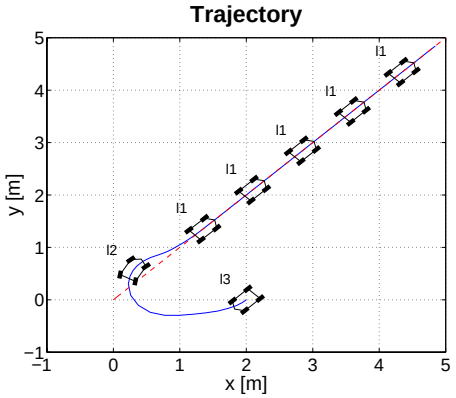

Figure 9.15: *The robot starts with an orientation of $\frac{-3\pi}{4}$ rad.*

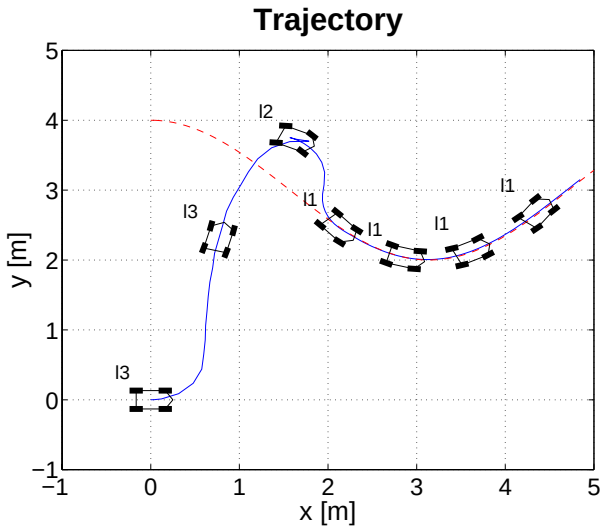

Figure 9.16: *From the starting posture $\xi = (0, 0, 0)$ the robot moves to the sinus-shaped trajectory and tracks it.*

CHAPTER 10

Passivity-Based Control

In this chapter a passivity-based controller to trajectory tracking is designed. The performance of a passivity-based controller to this kind of application is of interest since it has, to the authors knowledge, only been examined to a limited extend. The passivity-based controller designed in this chapter is furthermore of interest, because it offers an alternative to the partial linearization of the dynamics, which was described in section 9.1. In section 10.1 the dynamics of the system is rewritten, so that passivity-based control (PBC) can be applied. A controller for the dynamics is derived in section 10.2 and a controller for the kinematics is derived in section 10.3.

10.1 Rewriting the dynamics

The dynamics of an Euler-Lagrange system can be expressed as [Nijmeijer, 1998]:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{10.1}$$

$D(q)$ is a symmetric inertia matrix. g(q) accounts for gravity forces and $C(q,\dot{q})\dot{q}$ is centrifugal or Coriolis forces. In chapter 4 the dynamics of the robot was modeled as an Euler-Lagrange system. Hence the dynamics of the robot can through suitable factorization be expressed as equation 10.1. This is desirable because systems in the form of 10.1 can be controlled to follow a trajectory through passivity-based control as described in e.g. [Nijmeijer, 1998] and [Khalil, 2002]. Writing the dynamics in the form of equation 10.1 is therefore the subject of this section.

The symmetric inertia matrix $D(q)$ was found in section 6.3. From [Oriolo, 1995] it is known that the dynamics of a WMR can be written as:

$$D(q)\ddot{q} + n(q,\dot{q}) = A\lambda + Q(q)\tau \tag{10.2}$$

Where $n(q,\dot{q})$ is:

$$n(q,\dot{q}) = \dot{D}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}\left(\dot{q}^T D(q)\dot{q}\right)\right) \tag{10.3}$$

In fact equation 10.2 is the same as 6.2. To recapitulate D(q) is a matrix of inertia. D(q) is

symmetric positive definite and is given as:

$$D(q) = \begin{bmatrix} & & & 0 & 0 & 0 & 0 & & & & \\ & [R^T(\theta)MR(\theta)] & & 0 & 0 & 0 & 0 & & 0_{3\times 4} & & \\ & & & \frac{1}{2}I_W & \frac{1}{2}I_W & \frac{1}{2}I_W & \frac{1}{2}I_W & & & & \\ 0 & 0 & \frac{1}{2}I_W & \frac{1}{2}I_W & 0 & 0 & 0 & & & & \\ 0 & 0 & \frac{1}{2}I_W & 0 & \frac{1}{2}I_W & 0 & 0 & & 0_{4\times 4} & & \\ 0 & 0 & \frac{1}{2}I_W & 0 & 0 & \frac{1}{2}I_W & 0 & & & & \\ 0 & 0 & \frac{1}{2}I_W & 0 & 0 & 0 & \frac{1}{2}I_W & & & & \\ & & & & & & & \frac{1}{2}I_W & 0 & 0 & 0 \\ & 0_{4\times 3} & & & 0_{4\times 4} & & & 0 & \frac{1}{2}I_W & 0 & 0 \\ & & & & & & & 0 & 0 & \frac{1}{2}I_W & 0 \\ & & & & & & & 0 & 0 & 0 & \frac{1}{2}I_W \end{bmatrix} \tag{10.4}$$

A is the transpose of the constraint matrix Λ, λ are Lagrangian multipliers and $Q(q)$ is a matrix that maps the inputs τ into forces/torques performing work on $\dot{q}$. $Q(q)$ is defined as:

$$Q(q) = \begin{bmatrix} 0_{3\times 8} \\ I_{8\times 8} \end{bmatrix} \tag{10.5}$$

To remove the Lagrangian multipliers from equation 10.2, the property of the constraint matrix is used. From equation G.9 it is known that:

$$\Lambda\dot{q} = 0 \tag{10.6}$$

$$\Downarrow$$

$$A^T\dot{q} = 0 \tag{10.7}$$

Substituting $\dot{q}$ with the kinematic equation:

$$\dot{q} = S(q)u \tag{10.8}$$

yields:

$$A^T S(q)u = 0, \quad \forall\ u \tag{10.9}$$

$$\Downarrow$$

$$A^T S(q) = 0 \tag{10.10}$$

$$\Downarrow$$

$$S^T(q)A = 0 \tag{10.11}$$

From equation 10.11 it is clear that the Lagrangian multipliers in equation 10.2 can be removed by multiplication of $S^T(q)$:

$$S^T(q)\left[D(q)\ddot{q} + n(q,\dot{q})\right] = \acute{\tau} \tag{10.12}$$

where

$$\acute{\tau} = S^T(q)Q(q)\tau \tag{10.13}$$

Equation 10.12 resembles the desired form of equation 10.1, but a problem remains. The dimension of the input must be equal to the dimension of the output before PBC can be applied. The dimensions of the matrices and vectors in equation 10.12 are listed in table 10.1. Since the input

Vector/Matrix	q	$S(q)$	$D(q)$	$Q(q)$	τ	u	$\acute{\tau}$
Dimension	11×1	11×5	11×11	11×8	8×1	5×1	5×1

Table 10.1: *Dimensions for vectors and matrices in equation 10.12.*

$\acute{\tau}$ is a 5×1 vector and the output q is a 11×1 vector PBC cannot be applied. Inspired by the approach in [Sagatun, 1991] the vectors $\dot{q}$ and $\ddot{q}$ is substituted with $S(q)u$ and $\dot{S}(q)u + S(q)\dot{u}$ as given from equation 10.8. This results in the following system equation with input $\acute{\tau}$ and output u [Oriolo, 1995]:

$$D^*(q)\dot{u} - m(q,u) = \acute{\tau} \tag{10.14}$$

With:

$$D^*(q) = S^T(q)D(q)S(q) \tag{10.15}$$

$$m(q,u) = S^T(q)D(q)\dot{S}(q)u + S^T(q)n(q,S(q)u) \tag{10.16}$$

The matrix D in equation **??** was a symmetric positive definite matrix and hence so should $D^*(q)$ be. That $D^*(q)$ actually is symmetric positive definite can be seen from the following. A symmetric positive matrix like D can be written as:

$$D = R^T R \tag{10.17}$$

where R is a upper triangular matrix with positive diagonal entries [Lay, 1997]. Substituting equation 10.17 in equation 10.15 yields:

$$D^*(q) = S^T(q)R^T RS(q) \tag{10.18}$$

$$\Downarrow$$

$$D^*(q) = (RS(q))^T (RS(q)) \tag{10.19}$$

Hence the matrix $D^*(q)$ is positive definite.

Since the input and the output of the system in equation 10.14 has the same dimension and is of the form of equation 10.1, a passivity-based controller can be applied. It is however important to note that the output has changed from q to u, hereby dividing the control design into a controller for the dynamics and a controller for the kinematics. Or in other words, if a trajectory controller is applied to equation 10.14 only the dynamics of the system is controlled as illustrated in figure 10.1. The design of a controller to the dynamics is the subject of the following section and the design of a controller to the kinematics is found in section 10.3.

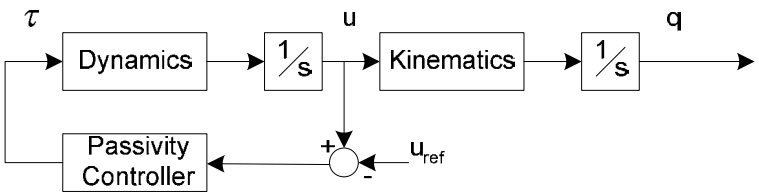

Figure 10.1: *The control structure for the dynamics of the robot.*

10.2 Controller Dynamics

The control design in this section is based on [Khalil, 2002]. We wish to track a time varying reference u_r. To ease the control design the reference u_r will for now be assumed to be constant. Once the control design is completed, the performance of the controller will be tested against a time-varying reference. The reference is tracked if the error vector e is minimized:

$$e = u - u_r \tag{10.20}$$

where:

$$u = [\eta \quad \zeta_1 \quad \zeta_2 \quad \zeta_3 \quad \zeta_4]^T \tag{10.21}$$

$$u_r = [\eta_{ref} \quad \zeta_{1ref} \quad \zeta_{2ref} \quad \zeta_{3ref} \quad \zeta_{4ref}]^T \tag{10.22}$$

Since u_r is assumed to be constant, the derivative of e can be substituted with $\dot{q}$ in equation 10.14, and hence satisfying the equation:

$$D^*(q)\dot{e} - m(q,u) = \acute{\tau} \tag{10.23}$$

The goal is to stabilize the system at $(e = 0, \dot{e} = 0)$. Let the input $\acute{\tau}$ be given as:

$$\acute{\tau} = -m(q,u) - K_p e + v \tag{10.24}$$

v is a new input to the system and K_p is a positive definite symmetric matrix. Substituting equation 10.24 in equation 10.23 leeds to:

$$D^*(q)\dot{e} - m(q,u) = -m(q,u) - K_p e + v \tag{10.25}$$

$$\Downarrow$$

$$D^*(q)\dot{e} + K_p e = v \tag{10.26}$$

Introducing the storage function V as:

$$V = \frac{1}{2} e^T K_p e \tag{10.27}$$

$$\Downarrow$$

$$\dot{V} = \dot{e}^T K_p e \tag{10.28}$$

From equation 10.26 an expression for $\dot{e}^T K_p e$ is found as:

$$D^*(q)\dot{e} + K_p e = v \tag{10.29}$$

$$\Downarrow$$

$$K_p e = v - D^*(q)\dot{e} \tag{10.30}$$

$$\Downarrow$$

$$\dot{e}^T K_p e = \dot{e}^T v - \dot{e}^T D^*(q)\dot{e} \tag{10.31}$$

Inserting equation 10.31 in equation 10.28 yields:

$$\dot{V} = \dot{e}^T v - \dot{e}^T D^*(q)\dot{e} \leqq \dot{e}^T v \tag{10.32}$$

Thus the system with input v and output $\dot{e}$ is passive with V as storage function. If the system is zero-state observable it can be globally stabilized by the control $v = -\phi(\dot{e})$. $\phi(\dot{e})$ is any function that satisfies:

$$\phi(0) = 0 \tag{10.33}$$

$$\dot{e}^T \phi(\dot{e}) > 0, \quad \forall\ \dot{e} \tag{10.34}$$

One choice for $\phi(\dot{e})$ is $v = -K_d\dot{e}$ where K_d is a positive definite symmetric matrix [Khalil, 2002]. A control control law for the dynamics is therefore:

$$\acute{\tau} = -m(q,u) - K_p e - K_d\dot{e} \tag{10.35}$$

$$\Downarrow$$

$$\tau = \left(S^T(q)Q(q)\right)^\dagger \left(-m(q,u) - K_p e - K_d\dot{e}\right) \tag{10.36}$$

where $\dagger$ denotes a pseudo-inverse. Equation 10.36 is actually an alternative to the partial linearization described in section 9.1. This solution is however considered to be more robust towards disturbances and modeling errors in the dynamics. The reason for this being that the partial linearization can be considered as an open loop controller which is entirely dependent on the model. The control law in equation 10.36 is a closed loop controller and should therefore be more robust towards disturbances and modeling errors. A requirement for the PBC solution is that the states η, ζ_1 and ζ_2 are measurable.

Through simulations appropriate values for the matrices K_p, K_d was found to be:

$$K_p = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0.01 \end{bmatrix} \tag{10.37}$$

$$K_d = 0.003 \cdot I_{5\times 5} \tag{10.38}$$

10.3 Controller Kinematics

The controller for the kinematics must track the time-varying reference $r(t)$:

$$r(t) = [\xi_{ref}(t) \quad \dot{\xi}_{ref}(t)]^T \tag{10.39}$$

Since the goal is to control the posture of the robot, the kinematic model of interest is again the posture kinematic model from section 5.4. Before PBC can be applied to the posture kinematic model, a change of coordinates $z = T(q_{pk})$ is necessary for two reasons. One is that the input and the output for the system must have the same dimension. The other reason is that a reference for the output $\dot{\gamma}$ in equation 5.19 cannot be found directly from $r(t)$. The same change of coordinates as in section 9.3 is applied resulting in the system:

$$\dot{z} = Az + B\beta(z)\left(\acute{v} - \alpha(z)\right) \tag{10.40}$$

where $z = [\xi \quad \dot{\xi}]^T$. The transformation of coordinates solves the second problem because a reference for the output z is found directly from $r(t)$. To solve the first problem with different dimension for the input and output, equation 10.40 is written as:

$$\dot{z} - A_s z + B_s\beta(z)\alpha(z) = B_s\beta(z)\acute{v} \tag{10.41}$$

$$\Downarrow$$

$$\dot{z} - A_s z + B_s\beta(z)\alpha(z) = \acute{u} \tag{10.42}$$

where $\acute{u} = B_s\beta(z)\acute{v}$ is considered as a the input to the system. Recapitulating the dimension of equation 10.41 in table 10.2. From table 10.2 and equation 10.42 it is seen that the input $\acute{u}$

Vector/Matrix	z	A_s	B_s	$\beta(z)$	$\alpha(z)$	$\acute{v}$
Dimension	6×1	6×6	6×3	3×3	3×1	3×1

Table 10.2: *Dimensions for vectors and matrices in equation 10.41.*

and the output z has the same dimension. If a passivity-based control law for $\acute{u}$ can be found, the inverse transformation back to $u = [\eta \quad \zeta_1 \quad \zeta_2]$ is found in two steps. First $\acute{v}$ is found as $\acute{v} = (B_s\beta(z))^{\dagger}\,\acute{u}$. From $\acute{v} = [\dot{\eta} \quad \zeta_1 \quad \zeta_2]^T$ the input u to the kinematic model can be determined by integration of $\dot{\eta}$.

Based on the same design procedure as used for the dynamics, a controller is designed for the kinematics. Because the design procedure is the same as given in section 10.2, the description in this section is less detailed. First an error vector is defined as:

$$e_{Kin} = z - r \tag{10.43}$$

A control law for $\acute{u}$ is introduced as:

$$\acute{u} = -A_s z + B_s\beta(z)\alpha(z) - K_{pKin}e_{Kin} + v \tag{10.44}$$

Inserting equation 10.44 in equation 10.42 and substituting z with e_{Kin}:

$$\dot{e}_{Kin} = -K_{pKin} e_{Kin} + v \tag{10.45}$$

$$\Downarrow$$

$$\dot{e}_{Kin}^T K_{pKin} e_{Kin} = \dot{e}_{Kin}^T v - \dot{e}_{Kin}^T \dot{e}_{Kin} \tag{10.46}$$

Consider the storage function V:

$$V = \frac{1}{2} e_{Kin}^T K_{pKin} e_{Kin} \tag{10.47}$$

$$\Downarrow$$

$$\dot{V} = \dot{e}_{Kin}^T K_{pKin} e_{Kin} \tag{10.48}$$

Inserting equation 10.46 in equation 10.48 yields:

$$\dot{V} = \dot{e}_{Kin}^T v - \dot{e}_{Kin}^T \dot{e}_{Kin} \leqq \dot{e}_{Kin}^T v \tag{10.49}$$

Hence the system with input v and output $\dot{e}_{Kin}$ is passive with the storage function V. A control law for v is chosen as $v = -K_{dKin} \dot{e}_{Kin}$ where K_{dKin} is a positive definite symmetric matrix. Transforming $\acute{u}$ back to $\acute{v}$ the control law becomes:

$$\acute{v} = (B_s \beta(z))^\dagger \left(-A_s z + B_s \beta(z) \alpha(z) - K_{pKin} e_{Kin} - K_{dKin} \dot{e}_{Kin}\right) \tag{10.50}$$

where † denotes a pseudo-inverse. With the kinematic controller the PBC-structure for the robot is shown in figure 10.2:

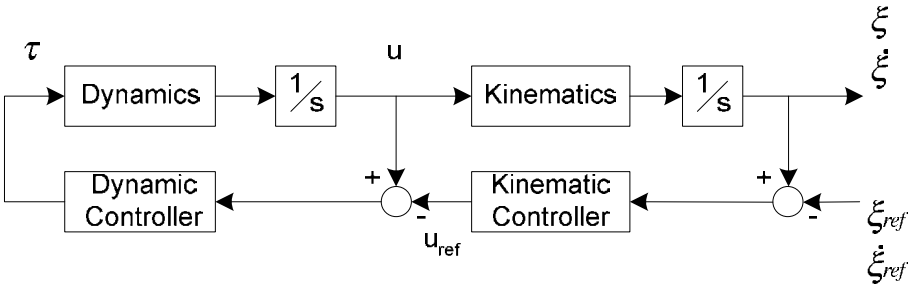

Figure 10.2: *The structure for the passivity-based control design.*

10.4 Determining K_{pKin}

According to the control design the requirement to K_{pKin} was that it had to be a symmetric positive definite matrix. In section 10.2 the equivalent matrix K_p was chosen as a diagonal matrix with positive entries. The choice of K_{pKin} needs to be a little different. Let's take a closer look at equation 10.50:

$$\acute{v} = (B_s \beta(z))^\dagger \left(-A_s z + B_s \beta(z) \alpha(z) - K_{pKin} e_{Kin} - K_{dKin} \dot{e}_{Kin}\right) \tag{10.51}$$

where the matrices and vectors are given as:

$$e_{Kin} = z - r = \begin{bmatrix} \xi \\ \dot{\xi} \end{bmatrix} - \begin{bmatrix} \xi_{ref} \\ \dot{\xi}_{ref} \end{bmatrix}, \quad 6 \times 1 \tag{10.52}$$

$$\beta(z) = R^T(\theta)\beta_1(z), \quad 3 \times 3 \tag{10.53}$$

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10.54}$$

$$\tag{10.55}$$

$$\beta_1(z) = \begin{bmatrix} l_1 cos(\gamma_2) cos(\alpha_1 - \gamma_1) & \big(l_1 cos(\gamma_2) sin(\alpha_1 - \gamma_1) & \big(-l_1 sin(\gamma_2) cos(\alpha_1 - \gamma_1) \\ -l_2 cos(\gamma_1) cos(-\alpha_2 + \gamma_2) & l_2 sin(\gamma_1) cos(-\alpha_2 + \gamma_2)\big)\eta & l_2 cos(\gamma_1) sin(-\alpha_2 + \gamma_2)\big)\eta \\ \\ l_1 sin(\gamma_2) cos(\alpha_1 - \gamma_1) & \big(l_1 sin(\gamma_2) sin(\alpha_1 - \gamma_1) & \big(l_1 cos(\gamma_2) cos(\alpha_1 - \gamma_1) \\ -l_2 sin(\gamma_1) cos(-\alpha_2 + \gamma_2) & -l_2 cos(\gamma_1) cos(-\alpha_2 + \gamma_2)\big)\eta & l_2 sin(\gamma_1) sin(-\alpha_2 + \gamma_2)\big)\eta \\ \\ sin(\gamma_1 - \gamma_2) & cos(\gamma_1 - \gamma_2)\eta & -cos(\gamma_1 - \gamma_2)\eta \end{bmatrix} \tag{10.56}$$

$$A_s = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \quad \text{and} \quad B_s = \begin{bmatrix} 0_{3\times3} \\ I_{3\times3} \end{bmatrix} \tag{10.57}$$

Calculating the pseudo-inverse in equation 10.51:

$$(B_s\beta(z))^\dagger = \left(\begin{bmatrix} 0_{3\times3} \\ I_{3\times3} \end{bmatrix} \beta(z) \right)^\dagger \tag{10.58}$$

$$= \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \beta(z) & & \end{bmatrix} \right)^\dagger \tag{10.59}$$

$$= \begin{bmatrix} 0 & 0 & 0 & \\ 0 & 0 & 0 & \beta^{-1}(z) \\ 0 & 0 & 0 & \end{bmatrix} \tag{10.60}$$

With equation 10.60 the control law for the kinematics in equation 10.51 can be reduced. The first term on the right hand side of equation 10.51 is:

$$-(B_s\beta(z))^\dagger A_s z = \begin{bmatrix} 0_{3\times3} & \beta^{-1}(z) \end{bmatrix} \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} z = 0 \tag{10.61}$$

The second term on the right hand side of equation 10.51 is:

$$(B_s\beta(z))^{\dagger}B_s\beta(z)\alpha(z) = \begin{bmatrix}0_{3\times 3} & \beta^{-1}(z)\end{bmatrix}\begin{bmatrix}0_{3\times 3}\\ \beta(z)\end{bmatrix}\alpha(z) \tag{10.62}$$

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\alpha(z) = \alpha(z) \tag{10.63}$$

Hence equation 10.51 is reduced to:

$$\acute{v} = (B_s\beta(z))^{\dagger}(-K_{pKin}e_{kin} - K_{dKin}\dot{e}_{kin}) + \alpha(z) \tag{10.64}$$

The two matrices K_{pKin} and K_{dKin} are open design parameters, but they must be symmetric positive definite matrices. In equation 10.51 it is the term $(B_s\beta(z))^{\dagger}K_{pKin}e_{Kin}$, which contains the feedback of the error in the posture. K_{pKin} is a 6×6 matrix. From equation equation 10.60 it becomes clear that the gains in the first three rows of K_{pKin} are without influence on the system because they are multiplied with zeros. Hence for the error e_{Kin} to have influence the gains in K_{pKin} must be placed in the last three rows. The influence of the e_{kin} can be divided into the influence of $\xi - \xi_{ref}$ and $\dot{\xi} - \dot{\xi}_{ref}$ as illustrated in figure 10.3 The appropriate values for gains

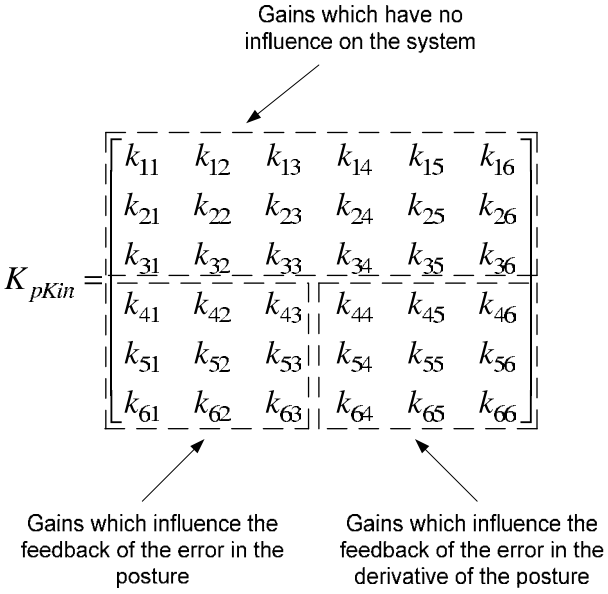

Figure 10.3: *Because K_{pKin} is multiplied with $(B_s\beta(z))^{\dagger}$, the gains in the first three rows of K_{pKin} is without influence on the system. The influence of the feedback error in the posture $(\dot{\xi} - \dot{\xi}_{ref})$ is found in the first three columns and the derivative of the posture $(\dot{\xi} - \dot{\xi}_{ref})$ in the last three rows.*

in K_{pKin} are found through trial and error in simulation. Though they have no influence in the simulation, gains are also placed in the first three rows are to keep K_{pKin} symmetric and positive

definite and consistent with the design. The matrix K_{pKin} is defined as:

$$K_{pKin} = \begin{bmatrix} 0 & 0 & 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.4 \\ 0.4 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0.4 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0.4 & 0 & 0 & 0.001 \end{bmatrix} \tag{10.65}$$

The gains in the matrix K_{dKin} is placed in the same way as K_{qKin} for the exact same reasons. The matrix K_{dKin} is defined as:

$$K_{dKin} = \begin{bmatrix} 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 \\ 0.01 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0.01 & & 0 & 0.001 \end{bmatrix} \tag{10.66}$$

10.5 Simulation

To verify if the derived control laws are valid, simulations are made using MATLAB. First the control law for the dynamics is tested. In the control design, it was assumed that the reference was constant. An interesting thing to see in a simulation is therefore, if the controller will be able to track a time varying reference. When the controller is implemented on the robot, the time varying reference will be constant in between two samples. It is therefore the time needed for the controller to reach a reference, which is crucial to the performance of the controller. In figure 10.4 and 10.5 the result of an applied step to the reference is shown. The two figures show that it takes

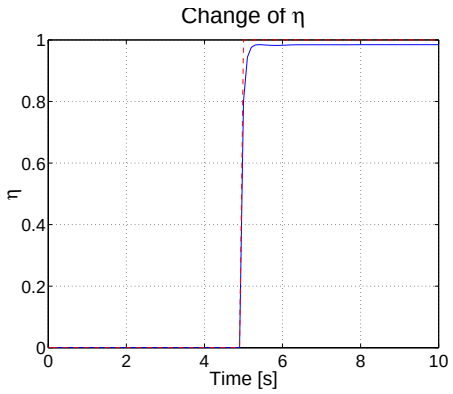

Figure 10.4: *The controller for the dynamics tracks a step in the reference (the dashed line) for η.*

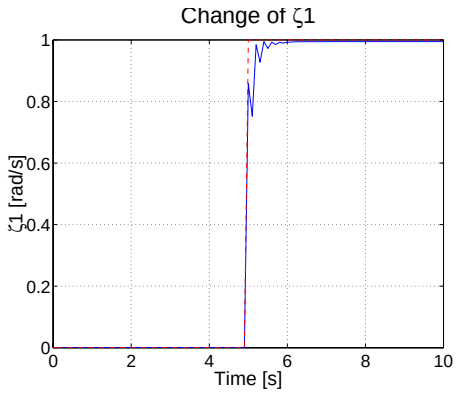

Figure 10.5: *The controller for the dynamics tracks a step in the reference for ζ_1. The plot for ζ_2 is equal to the one for ζ_1 and is therefore not shown.*

the controller under a second to find the reference. With a reference that changes for each sample and a sampling frequency of 10 Hz, the controller may not be able to track the reference and cause

instability. It is however important to note that with a feasible trajectory the reference u_{ref} will change in smaller steps than the one applied in the simulation in figure 10.4 and 10.5. In reality there is a limit to how fast the wheels can be turned. Hence the values of u must be subject to constraints. The following assumption is made about the elements in u:

$$u = \begin{bmatrix} \eta \\ \zeta_1 \\ \zeta_2 \end{bmatrix}, \quad \text{where} \quad \begin{array}{c} -0.6 < \eta < 0.6 \\ -0.5 < \zeta_1 < 0.5\,[rad/s] \\ -0.5 < \zeta_2 < 0.5\,[rad/s] \end{array} \tag{10.67}$$

These constraints are applied to u_{ref} in the following simulations. The constraints defines some max. and min. values for u_{ref}, but doesn't constrain how fast it may change. Once the kinematics is included in the simulation it will be easier to tell how u_{ref} changes. For now it is assumed that the controller for the dynamics is fast enough to track a time varying reference and then return to the topic once the kinematics have been included.

In the following simulations both dynamics and kinematics are included. In the simulations the time varying reference for the trajectory is again:

$$r(t) = [t, \quad t, \quad \frac{\pi}{4}, \quad 1, \quad 1, \quad 0]; \tag{10.68}$$

In the first simulation, no constraints were applied to the angle of the wheels. The result is shown in figure 10.6. Two things can be concluded from this simulation. One is that the control design

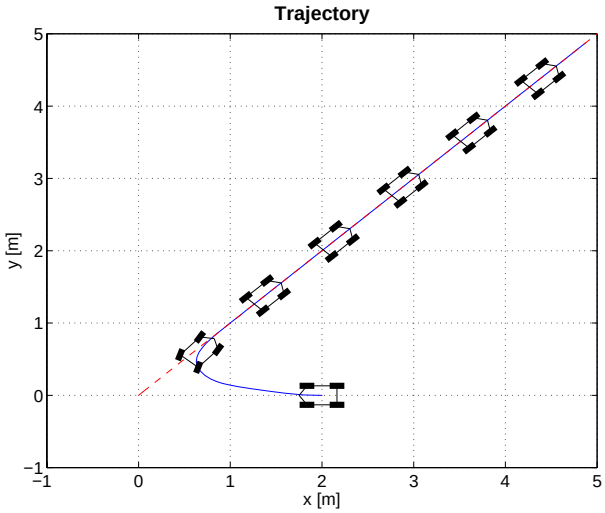

Figure 10.6: *From a start position in (0,2) and orientation pi the robot moves to and tracks a trajectory.*

works as intended, since the robot moves to and tracks the trajectory. The second is that the controller for the dynamics doesn't cause instability. The change in the reference u_{ref} and the controlled u for the movement of the robot in figure 10.6 is shown in figure 10.7. The figure clearly shows that the controller for the dynamics is able to track the reference in this situation, since u and u_{ref} are close to identical. Based on the results from this simulation, it is concluded that the controller for the dynamics is fast enough to track the time varying reference.

In the next step the constraints for the angle of the wheels were applied to the simulation. Not surprisingly the simulations showed that the passivity-based controller experienced the same

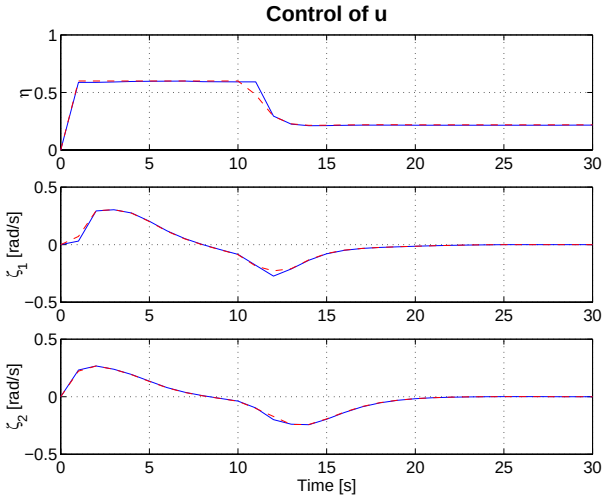

Figure 10.7: *The figure shows how u and u_{ref} changes. Only the first 30 sec. is shown in the figure because the values remain constant here after. The solid line is u and the dashed u_{ref}.*

problems as control by feedback linearization. To summarize, the problems with the constraints was related to the ability to orientate the robot with the trajectory. The angles for the front and rear wheels both ended up in max. or min. which resulted in a lateral movement and thereby preventing the robot from aligning the orientation with the trajectory.The problems are solved with the use of the hybrid automaton derived in section 9.7. The directed graph for the hybrid automaton is shown in figure 10.8. With the hybrid automaton, the robot is able to move close to and track

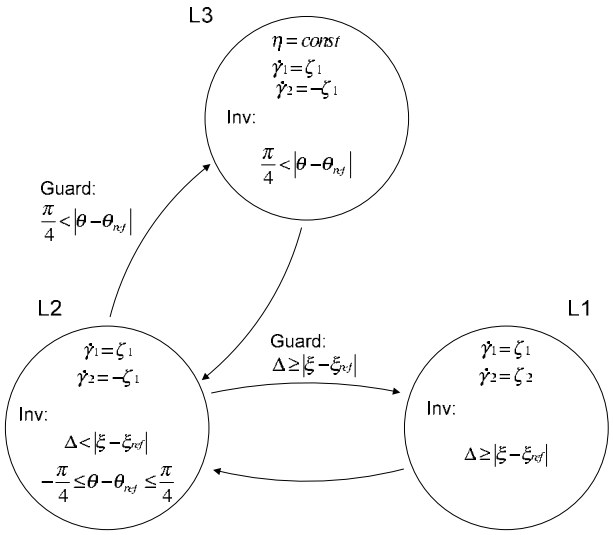

Figure 10.8: *Directed graph for the hybrid automaton.*

a trajectory. Examples of this, where the robot starts from the same position but with different orientations are shown in figure 10.9, 10.10, 10.11 and 10.12. With these results, it is concluded that the controller works as intended. As with the control by feedback linearization a test of the controller with the sinus-trajectory is shown in figure 10.13:

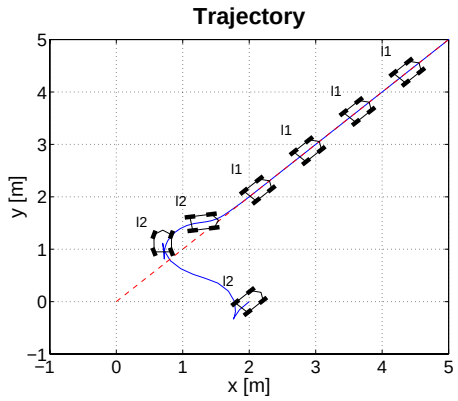

Figure 10.9: *The robot starts with an orientation of $\frac{\pi}{4}$ rad.*

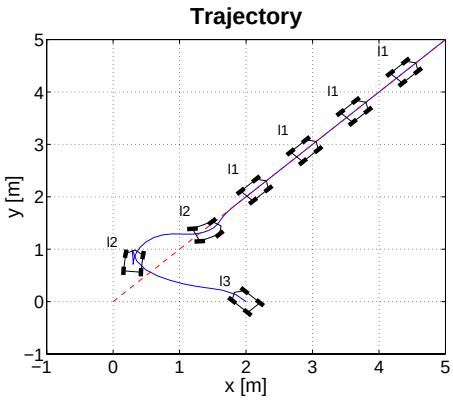

Figure 10.10: *The robot starts with an orientation of $\frac{3\pi}{4}$ rad.*

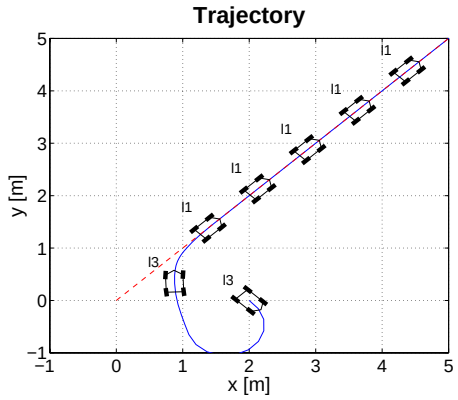

Figure 10.11: *The robot starts with an orientation of $\frac{-\pi}{4}$ rad.*

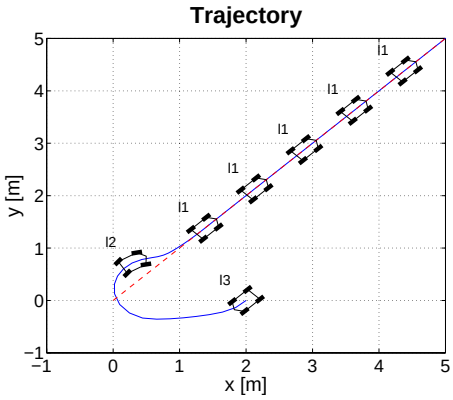

Figure 10.12: *The robot starts with an orientation of $\frac{-3\pi}{4}$ rad.*

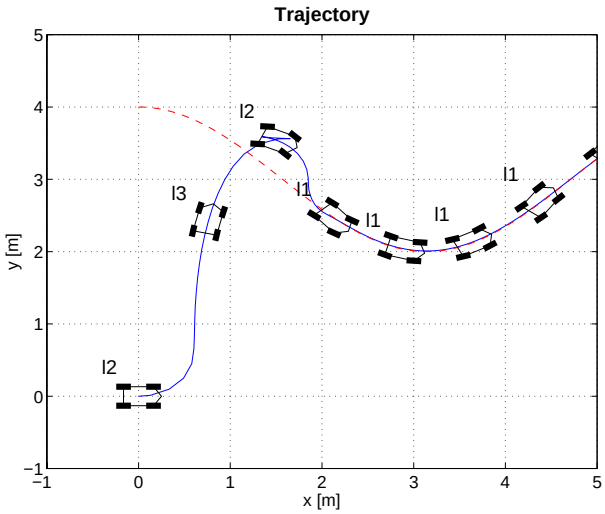

Figure 10.13: *Starting from a position some distance from the trajectory the robot converges to the trajectory and tracks it.*

CHAPTER 11

Comparison of Controllers

Having designed two different controllers for trajectory tracking, it is interesting to compare their performaces. If one compares the plotted simulations from section 9.7 with those in section 10.5 the plots look almost identical. From the plots nothing can be said as to which controller performs better. This is perhaps not so surprising, if the control laws for the kinematics are compared. The kinematic control law for control by feedback linearization and the control law for the passivity-based controller are:

$$\acute{v} = \beta^{-1}(z)K_c(z-r) + \alpha(z) \tag{11.1}$$

$$\acute{v} = ((B_s\beta(z))^{\dagger}(-K_{pKin}e_{Kin} - K_{dKin}\dot{e}_{Kin}) + \alpha(z) \tag{11.2}$$

In section 10.4 it was found that only the three last rows in K_{pKin} had influence on the feedback of the error $e_{Kin} = (r - z)$. These three rows can be found as the matrix K_c in equation 11.1. The two terms:

$$\beta^{-1}(z)K_c(z-r) \quad \text{and} \tag{11.3}$$

$$-((B_s\beta(z))^{\dagger}K_{pKin}e_{Kin} \tag{11.4}$$

are therefore the same.

With this observation the only difference between the two equations is the differential term in equation 11.2. With the chosen value for K_{dKin}, the influence of that term becomes minimal and the two equations are therefore almost identical. This is the reason why the plots for the two controllers are so very similar.

To compare the performance of the two controllers it is necessary to look at the dynamics where greater differences are found. found. In figure 11.1 and 11.2 the change of the output ζ_1 from the dynamics is shown for the two controllers. The output is generated from the simulations where the robot tracks the sinus-shaped trajectory shown in figure 9.16 and 10.13. The difference between the two controllers is best seen as the robot converges to the trajectory which is done in first 50 sec. Here after the plots are close to identical. As the robot converges to the trajectory the hybrid controller is in one of the two states l_2 and l_3. In these states $\zeta_2 = -\zeta_1$ which is why only ζ_1 is considered in the figures. Comparing figure 11.1 and 11.2 it is observed that the output ζ_1 for the passivity-based controller is smooth. A smooth change in ζ_1 is desirable since it results in a smooth change in the angle of the wheel. The output ζ_1 from the partial linearization in the controller using feedback linearization is less smooth. Here ζ_1 changes quickly and in large steps. If implented the servos would be worked unecessarily hard and use more power than with the

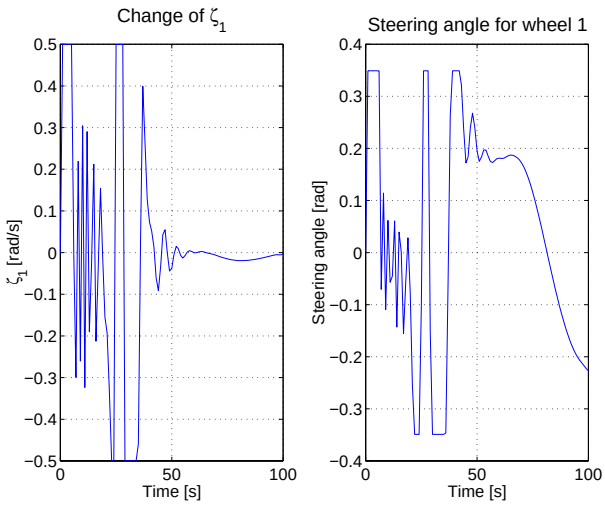

Figure 11.1: *In the left plot the output ζ_1 from the dynamics for the controller using feedback linearization is shown. On the right the resulting steering angle for wheel 1.*

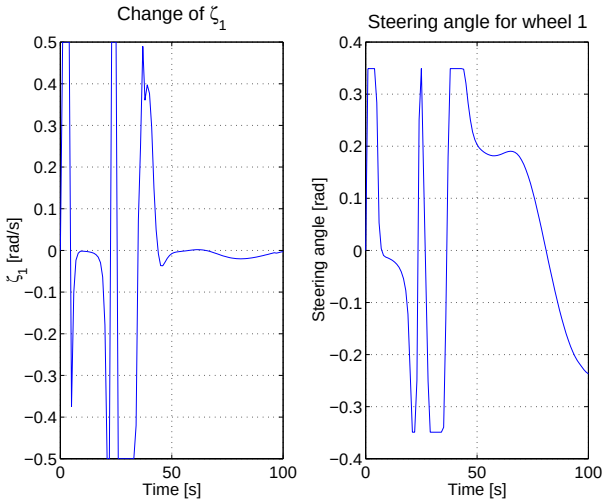

Figure 11.2: *Same plots as in figure 11.2 for the passivity-based dynamics controller.*

passivity-based controller. Furthermore the large changes in ζ_1 from the partial lineariztion may cause the controller to become unstable after implementation because the servos can't turn the wheels fast enough. A problem which isn't experienced in simulations because the servos isn't incluede in the model, and neither is the friction between the wheels and the surface. Hence the passivity-based dynamics controller have a clear advantage over the partial linearization used in the control by feedback linearization.

In section 10.2 it was argued that the passivity-based dynamics controller was more robust towards modelling error and disturbances than the partial linearization. To verify this statement a simulation where noise is applied to the dynamics is performed. The noise is applied as a random factor between ± 0.3 which is added to the output from the dynamics $\dot{u}$. The resulting trajectory is shown in figure 11.3 and 11.4. The figures shows that the passivity-based dynamics controller is able to suppress the disturbances better than the partial linearization. In the figures this is seen

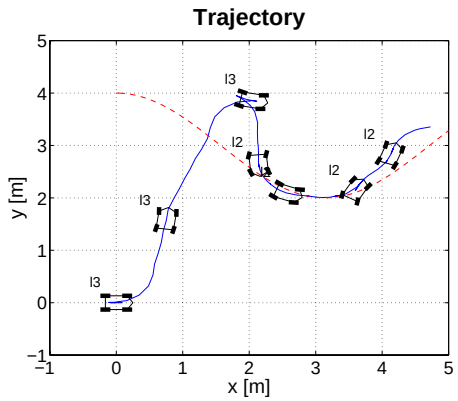

Figure 11.3: *Resulting trajectory for the control by feedback linearization when noise is added to the dynamics.*

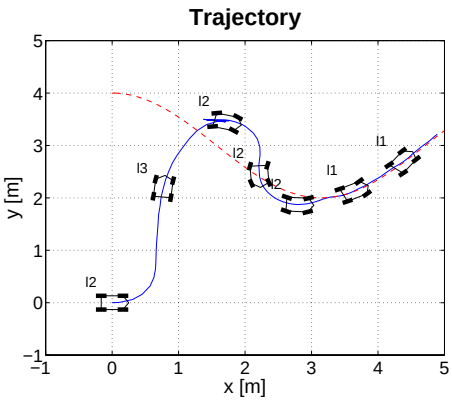

Figure 11.4: *Same situation as in figure 11.3 with the passivity-based controller.*

not only by the trajectory which is more smooth but also by the states. Once on the trajectory the hybrid controller remains in the state $\not{l}_1$ for the passivity-based controller while the control by feedback linearization struggles to stay on the trajectory and remains in the state $\not{l}_2$.

11.1 Summary

Two controllers for trajectory tracking have been designed. One based on feedback linearization and the other on passivity-based control. The control problem was in both designs divided into control of the dynamics and control of the kinematics. The control problems were approached in different ways. The controller based on feedback linearization used partial linearization to reduce the control problem to the kinematics. The other designed a controller separately for the dynamics and the kinematics.

Though the controllers where designed separately and through different design methods, the resulting two controllers for the kinematics turned out almost identical.

Better performance was observed by the passivity-based dynamical controller than with the partial linearization when the robot converges to a trajectory. Simulations revealed that the output from the passivity-based dynamical controller had a more smooth character, than the partial linearization. The output from the partial linearization changed quickly and with large steps which was undesirable since it puts strain on the servos and may cause instability. Furthermore the passivity-based dynamical controller showed better performance when disturbances was added to the dynamics of the system.

Part IV

Implementation and Test

CHAPTER 12

Implementation Framework

To be able to implement the designed controllers, a framework is needed for the onboard computer on the robot. The framework will contain the controller, the odometry model, the pathgenerator and the communication with the servos, encoders and DC-motor. Furthermore a speed-controller for the DC-motor will be implemented here. The design and implementation of the framework will be the subject of this chapter.

As mentioned earlier, it is the intention that the robot in future projects will be cooperating with a number of other robots. To ensure that it will be possible to reuse the elements of this project in future works, it is important that the code is well-structured and documented. This will be ensured by splitting up the code in smaller modules, which are easier to get an overview of.

Inspiration for the design of the framework has been taken from [Lima, 1999] and [Taylor, 2002]. [Lima, 1999] suggests a ”blackboard” for keeping shared variables. The blackboard can be extended to be distributed on more than one robot, thus making it ideal for the future developments of this project. [Taylor, 2002] introduces the idea of having multiple control modes available, an idea which is suitable for this project.

The outline of the chapter is: First the requirements of the framework are listed, and the choice of operating system is adressed. After the tasks to be performed has be determined, a design consisting of several modules (containing the code for a specific task) is suggested. Some more detailed implementation issues for each of the modules are placed in appendix D.

All the developed code are placed at [CD, \txt1\] along with a Makefile for easy compilation. A users guide for the compilation and the (simple) user-interface can be found in appendix C.

12.1 Requirements

The requirements for the framework are to make implementations of the following possible:

- Controllers.
- Odometry model.
- Pathgenerator.
- DC-motor speed controller.

Apart from the implementation of the four mentioned above, a clock must also be implemented, since the trajectories generated by the pathgenerator are time-dependent. Without a clock, it will not be possible to determine when the reference for the controller has to be changed.

Having the controller etc. implemented using the framework, a user-interface to e.g. start and stop the execution is needed. In this user-interface a mode to steer the robot manually using the keyboard is also implemented, and datalogging is made possible. The implemented user-interface is called `txt1`, and the source code is found in [CD, \txt1\txt1.c]. A users guide for the user-interface is found in appendix C. For further documentation of the source code for the user-interface the reader should refer to that file.

We define the main sampling frequency as the frequency the controller is executed with. The odometry model is also updated with the main sampling frequency. The DC-motor speed controller must be executed with a higher frequency than the controller. To guarantee this, it is a requirement, that both the main sampling frequency and the frequency for the DC-motor speed controller are fixed and known. The pathgenerator will be executed at startup, and then again every time the trajectory end-point is reached. Thus the pathgenerator will not be executed with a fixed frequency.

12.2 Choosing an Operating System

Based on prior experiences with both MS Windows and Linux it was decided by the project group that Linux would be the best choice for the implementation on the robot. Linux is preferred since it is Open Source and thereby makes software-development on the lowest levels of the operating system much easier than on e.g. MS Windows. Furthermore the documentation and support for Linux are extensive through various sources such as Internet sites and newsgroups.

As stated in the requirements to the framework, the main sampling frequency has to be constant and known. It is a task for the operating system to ensure that certain processes can be scheduled with a fixed frequency, but this feature is not supported by Linux nor MS Windows. In other words Linux doesn't support hard real-time implementations which is needed for this application.

A possible solution is to use the RTLinux kernel, and that solution has been chosen in this project. The RTLinux kernel is described shortly in the following subsection.

12.2.1 About RTLinux

The RTLinux kernel is an extension to Linux that makes hard real-time applications feasible. The RTLinux kernel accomplishes real-time performances by removing sources of unpredictability (such as disk handling and execution of demanding processes). The RTLinux kernel can be considered to be placed between the standard Linux kernel and the hardware. The standard Linux kernel thereby sees the RTLinux kernel as the actual hardware. In the RTLinux kernel it is possible to introduce and set priorities to each task, and correct timing for the processes can be achieved by using the scheduling algorithms, priorities and setting the frequency of execution. The RTLinux kernel assigns lowest priority to the standard Linux kernel, thereby only executing standard Linux when no real-time applications needs to be executed [Divakaran, 2002]. The organization of the Linux and RTLinux kernels and processes are illustrated in figure 12.1. Since RTLinux can be somewhat a challenge to install, a detailed installation description applicable for the onboard computer on the robot is offered in appendix E. In appendix D.1 there is a short description of the

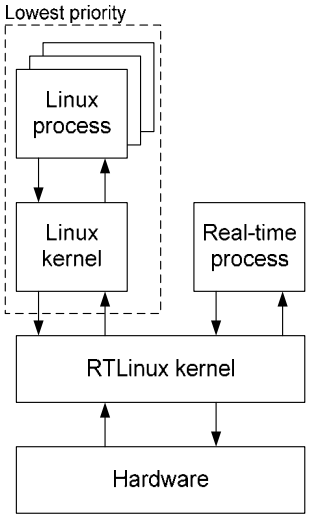

Figure 12.1: *The coexistence of the Linux and RTLinux kernels and processes. The RTLinux kernel is running the Linux kernel with the lowest priority thus ensuring real-time scheduling for the RTLinux process.*

implementation of real-time threads.

12.3 Tasks

With basis in the requirements from section 12.1, the tasks to be accomplished by the framework is elaborated in the following. Two threads are needed: One to execute the main loop containing the controllers for trajectory tracking and one for the DC-motor speed controller. In the following, we will call the thread with the controller for trajectory tracking the “main control thread” and the thread with the DC-motor speed controller the “DC-motor control thread”. As mentioned earlier the DC-motor control thread is run with a frequency higher than the main sampling frequency. In the following, the tasks of each of the two threads will be described.

12.3.1 DC-Motor Control Thread

The task of the DC-motor control thread is to regulate the velocity of the DC-motor to a desired reference value. The reference is accessible through a shared variable. The tasks of the thread are:

1. Retrieve encoder-values and the current reference velocity.
2. Calculate new control signal to be applied to the DC-motor.
3. Apply the control signal to the DC-motor.
4. Make encoder-values available for other modules.

The tasks for DC-motor control thread are illustrated in figure 12.2.

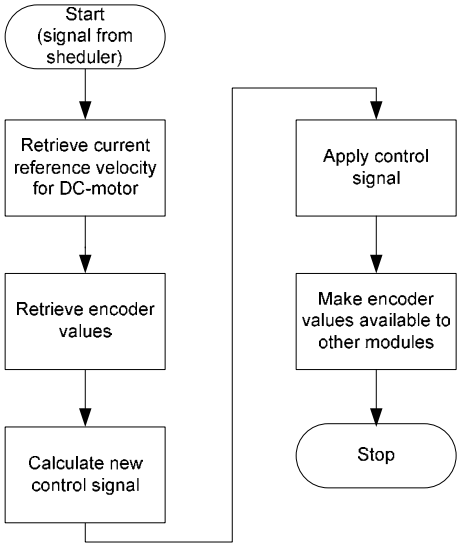

Figure 12.2: *The tasks to be performed by the DC-motor control thread.*

12.3.2 Main Control Thread

As mentioned several time earlier, the posture of the robot must be known for the controller to work. The posture is found by use of the odometry model, which uses the encoder values as input. These has been retrieved from the encoders by the DC-motor control thread and are accessible for the odometry model.

The reference for the robot is the next posture on the trajectory to be tracked. This reference changes with time, since the trajectory is time-varying. It is therefore necessary to have an internal clock to keep track of time, and to compare the value of the internal clock to the value of the time when the robot was supposed to arrive at the next posture in the trajectory. If the value of the internal clock is higher than the expected arrival time for the next posture, the reference should be changed to the next posture in the trajectory. Based on the odometry and the reference, a control signal is calculated by the controller. Afterwards the signal is be applied to the servos and DC-motor. In other words, the tasks to be performed to implement the controller and pathgenerator are:

1. Update the estimation of the position using the odometry model.
2. Update the reference using the pathgenerator.
3. Calculate new control signal.
4. Apply control signals to the servos and the DC-motor.
5. Maintenance of the internal clock.

The tasks listed here are to be executed with a fixed frequency using the RTLinux scheduler, and they are illustrated in figure 12.3.

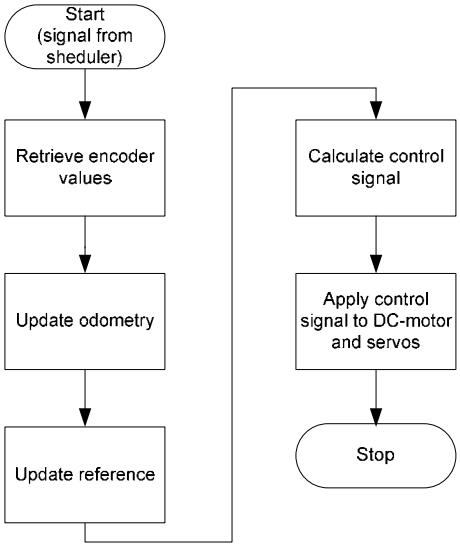

Figure 12.3: *The tasks to be executed by the main control thread at each time-step. The execution is initiated by the RTLinux scheduler to ensure a fixed frequency of execution.*

12.4 Modules

To ensure well-structured sourcecode, the tasks can be classified according to the nature of the task. The classification results in eight groups of tasks. Each of these seven groups of tasks will be implemented in RTLinux as a kernel module, so from now on we will call them modules.

`Blackboard` Since the modules needs to share various variables, an environment for shared variables is also needed. We will call this environment the Blackboard.

`Supervisor` To update the internal clock and generate and update trajectories a supervisor is implemented.

`Control` The implementation of the control algorithms and the criterions to choose between them.

`Odometry` The implementation of the odometry model.

Hardware Abstraction Layer (`HAL`) The hardware abstraction layer, which makes communication with the hardware possible. The only module which is dependent on the robot platform.

`Pathgen` The pathgenerator.

For implementation issues two more modules are needed:

`Main` Schedules the execution of the other modules. Is executed by RTLinux with a fixed frequency (the main sample frequency), and then in turn executes the functions of the other modules. Also contains the DC-motor speed controller.

`Functions` Various functions needed by the other modules, e.g. support for matrix operations.

The modules will be implemented in C as kernel modules, as is standard procedure when developing kernel level applications for Linux. In appendix D.2 the modules are specified with descriptions of the functions of each modules and the blackboard-variables maintained for each module.

12.5 Summary

To be able to implement the controllers for trajectory tracking and the DC-motor speed controller, an implementation framework for the onboard computer is needed. An implementation consisting of eight kernel modules was suggested and implemented. The implementation was performed using RTLinux, which is an extension to a standard Linux kernel, which allows hard real-time applications. The implementation of the framework on the onboard computer was succesful, but since none of the controllers were implemented, the framework was never tested fully.

CHAPTER 13

Tests

As mentioned earlier, the project group encountered some practical problems during the project period. Due to the problems, the original intention about implementing and testing both the controller based in feddback linearization and the passivity-based controller wasn't satisfied. Therefore no tests were performed on the controllers. The implementation model on the other hand was implemented, and the test are documented in the following.

13.1 Odometry

To test the odometry, three tests were performed

1. Straight motion.
2. Turning motion.
3. Travelling along the boundary of a square.

The first test is a simple test, to check that the odometry is working properly when moving along a straight line without turning. Although simple, the test will reveal if the measurements from the encoders are accurate. The second test is about turning motion. The robot will be moved in a curve, to check the orientation. As a final the test, the robot is moved along a two by two meter square. That is, the starting and the ending point is the same. This is done to verify that the odometry will reveal that the robot is returning to the starting point af travelling a distance.

13.1.1 How the Test were Carried Out

The tests were performed without the use of the DC-motor. Instead the robot was pulled by hand. The robot was pulled the robot by hand, because it is difficult to steer the robot as indended using the manual control mode.

13.1.2 Results

The result of the first test is seen in figure 13.1. As expeced the robot is moving along a almost straight line from the starting to ending point, making the test succesful.

The result of the second test, where the robot is moved so the orientation changes 90° is illustrated in figure 13.2 and 13.3. The test reveals an inaccuracy of the odometry. As seen in figure 13.2 the orientation according to the odometry is only approx. 85°, which is 5° wrong.

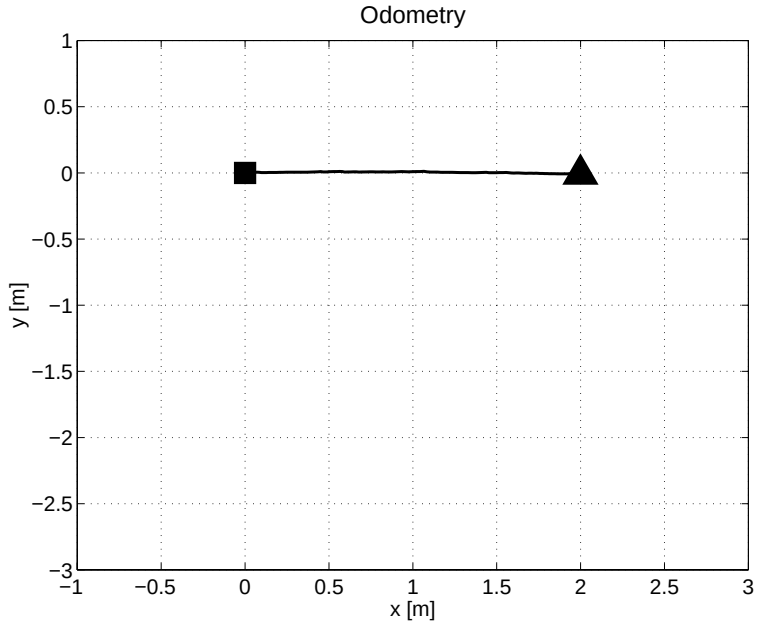

Figure 13.1: *The straight motion test. The robot was moved from* $(0, 0)$ *to* $(2, 0)$ *as indicated by the square and triangle.*

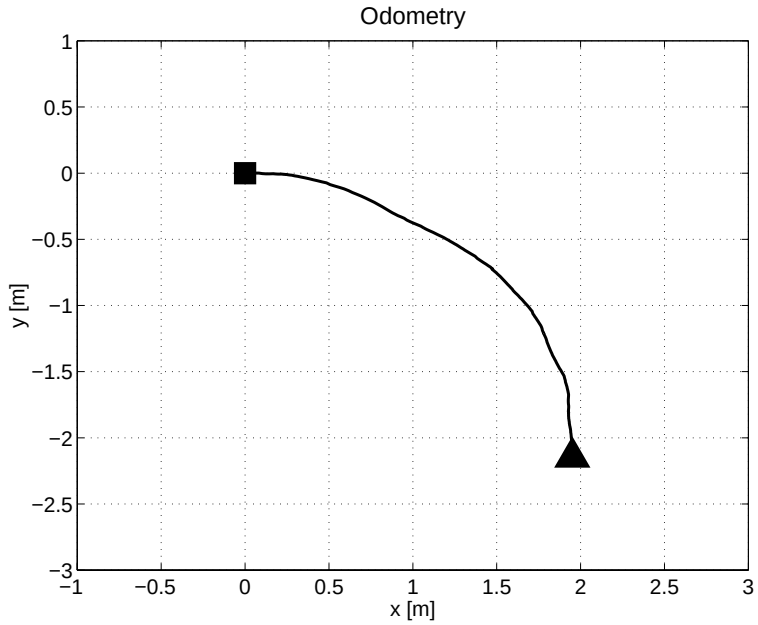

Figure 13.2: *The result of the second test, where the orientation of the robot is changed. The robot is moved from* $(0, 0)$ *to* $(2, -2)$*, but as seen the odometry is inaccurate. According to the odometry the position is* $(1.95, -2.18)$.

Since the calculations of the changes in positions is based on the change in orientation, the error propagates to the position as seen in figure 13.2. The robot is moved from the point $(0, 0)$ to $(2, -2)$. According to the odometry, the final position is $(1.95, -2.18)$, meaning an error of 5 cm. in x and a significant error of 18 cm. in y.

One of the reasons to the error is found by inspecting the change in position in more detail. The tests reveals that the encoder measurements from wheel 2 are inaccurate. The inaccuracies

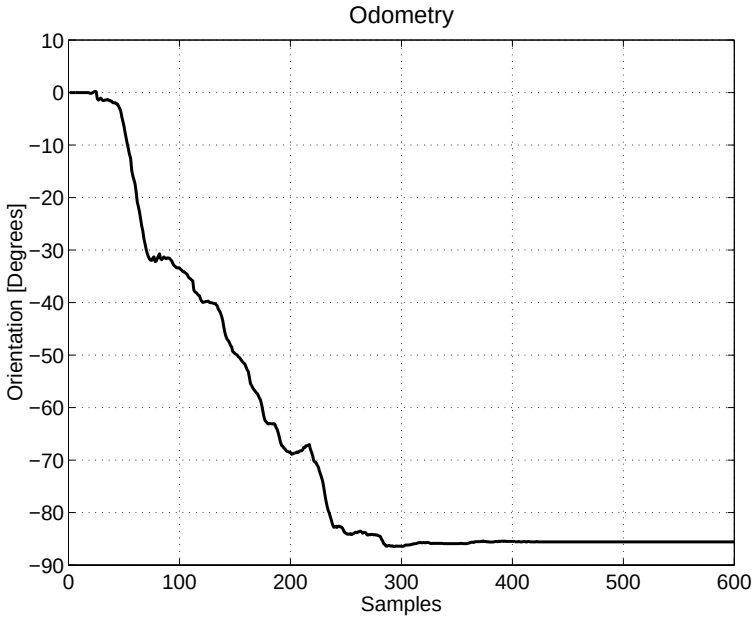

Figure 13.3: .

are seen in figure 13.4 and 13.5 as spikes. In figure 13.4 the spikes are at $x \approx 0.5, 1.0, 1.5$ and in figure 13.5 they are at $x \approx 0.25, 0.75, 1.25, 1.75$. The difference between the two plots is the starting wheel angle of the wheel. A mark was made on the tire using chalk. In the left plot, the mark was placed closest to the ground. In the plot at the right, the mark was placed away from the ground. The starting angle of the wheel has been turned half a revolution from the plot in figure 13.4 to the plot in figure 13.5. Since the circumference of the wheel is 0.5 m. the plots thereby indicates, that something is wrong with the reflective wheel of the encoder on wheel 2. By inspecting the reflective wheel, it was revealed that some of the reflective and non-reflective pattern was damaged. This caused the encoder to make a "wrong count" once in every revolution. The final test is the "square-test", where the robot is pulled along a two by two meter square. In the

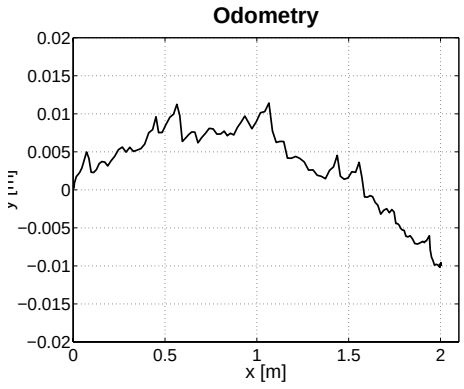

Figure 13.4: *The position of the robot with the chalk-mark on the tire closest to the ground.*

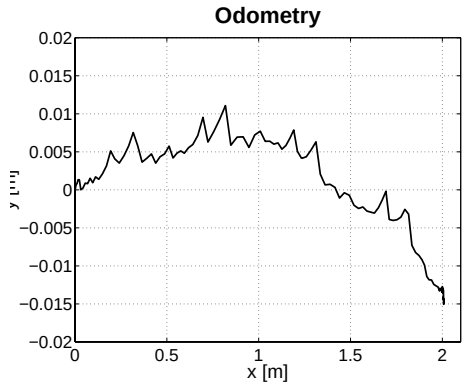

Figure 13.5: *The position of the robot with the chalk-mark away from the ground.*

corners of the square the robot is moved back and forth a number of times in order to get the right change in orientation of $90°$. With the inaccuracies found in the above, it wasn't expected that the

test would be succesful. And as seen i figure 13.6 this wasn't the case either. The odometry was expected to reveal that the robot returned to the starting point in $(0, 0)$ after covering the boundary of the square. This is not fulfilled due to the error stated in the test of the orientation.

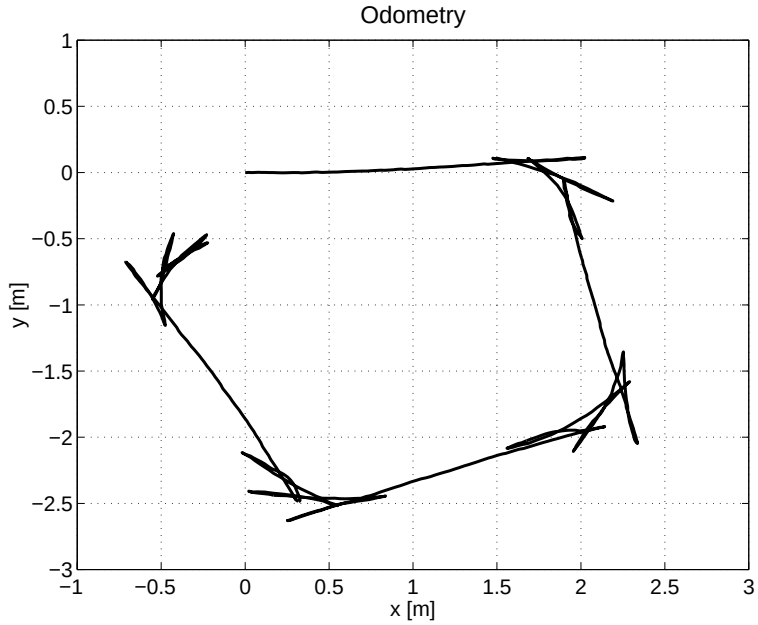

Figure 13.6: *The final test, where the robot is moved along a two by two meter square. The robot was moved back and forth a number of times in the corners in order to turn the robot* $90°$. *The test fails, since the odometry doesn't show that the robot returns to its starting position in* $(0, 0)$.

13.1.3 Summary

In this section the odometry for the robot was tested using three different tests. The tests revealed that the implementation contains an error. The reflective wheel used with the encoder in wheel 2 is damaged, and therefore introduces an error to the odometry. At the time this is written, no new reflective wheel is accessible to the project group. But it is expected that replacing the reflective wheel will correct the errors and make the odometry work satisfactory.

Part V

Conclusion

CHAPTER 14

Conclusion

With the intent to be part of a larger project for cooperative robots a wheeled mobile robot was to be build. Previous to this project it was decided that the frame of the new robot was to be build on a TXT1-Extreme Truck from Tamayia. The problem specification for the project was:

How can a Tamiya TXT-1 Xtreme Truck be modified in such a way that it is able to generate and track a trajectory between two given points?

The problem specification was divided into four areas. These areas are hardware modifications and design, modelling, control and delopment of an implementation framework.

Hardware Modification and Design

Problems posed in this are were design and implementation of an input interface to collect data from sensors and an output interface to control of actuators. The sensors which was used to position the robot was limited to two wheel encoders. The input interface was based on a PIC16F877 microcontroller which translated the raw signals from the encoders into a wheel angle. From the microcontroller the wheel angle was sent to the onboard PC using the RS-232 standard. The output interface consisted of a SSC serial controller and a speed controller. The SSC-serial controller translated the control signal sent by RS-232 from the onboard PC to a puls width modulated signal. The power of the puls width modulated signal to the motors for propulsion was amplified through the speed-controller. The SSC serial controller and the speed controller were both perched.

Modelling

In the area modelling three models for the robot was derived: A kinematic-, a dynamic- and a odometry model. The kinematic model was derived from the pure rolling without slipping-constraint where as the dynamic model was derived through Lagrange equations. To satisfy the rolling without slipping-constraint it was assumed that the wheels could be turned and controlled independently in both the kinematic and dynamic model. From simulations in MATLAB it was concluded that the kinematic and dynamic model behaved as intended.

In the derivation of the odometry model it was necessary to assume that the rear wheel pair was locked. If this assumption wasn't made it would be a requirement to know the steering angles of the wheels. The measurements of the steering angles are however at this stage not possible on the robot. The derived odometry model was verified in MATLAB.

Control

Two controllers for trajectory tracking was designed. One controller was based on feedback linearization and the other used passivity-based control. The control problem was in both controllers divide into control of the kinematics and control of the dynamics. The controller based on feedback linearization used partial linearization to reduce the control problem to the kinematics of the robot. The other designed a passivity-based controller separately for the kinematics and dynamics. Simulations revealed that due to constraints on the steering angles, both controllers had problems aligning the robot with the trajectory. The problems were overcome by introducing a three state hybrid automaton.

The controllers were never tested on the robot because of hardware troubles in the odometry. Therefore the two controllers performance were compared based on result from simulations. Though designed separately the control law for the kinematic ended up almost identical for the two controllers. The comparison of performance was therefore limited to the dynamics. Better performance was observed by the passivity-based controller for the dynamics when the robot converged to the trajectory from some posture away from it. Here the control signals to the servos changed smoothly. The partial linearization for the dynamics changed the control signals more rapidly which causes greater strain on the servos than the passivity-based controller when implemented. Furthermore the passivity-based controller for the dynamics was more robust towards disturbances and modeling errors.

Implementation Framework

In the last area a software framework for the onboard PC was suggested. The framework was implemented as 8 kernel modus. The implementation was performed using RT-linux.

Future Work

If the project was continued a natural starting point would be the establishment of correct odometry positions. It would however be preferable if the determination of the position wasn't solely depending on odometry, since odometry unavoidedly will become wrong as time passes. The incooperation of other sensors for positioning could therefore be a topic for future work on the robot. Once a reliable position can be determined, design and test of controllers can be performed. Future work for the control design could be expanded to incooperate posture stabilization, obstical detection and avoidance. Furthermore the design of a more advanced path planner would be preferable.

It is the intention that the robot shall participate in a project concerning cooperative robots. Another area for the future work is therefore the communication between the robots and the determination of the tasks for the robots to perform.

Part VI

Appendix

APPENDIX A

Linear friction

The purpose of this appendix is to find a transfer function for the motors through an experiment. A step is applied to the motors, and the angular velocity of one of the wheels is logged. Based on the model for the motors in section 7.1, a more detailed model is derived here. This extended model is given a step and the transfer function is tuned to fit the logged data from the motors. From the fitted transfer function the constant B_T is found.

Only the speed of the motors can be measured due to the optical encoders placed near the wheels. Unfortunately, speed (ω_w) is not a part of the model derived in section 7.1 on page 47. Based on figure 7.5 on page 50, an extension including the relationship between torque (τ_m) and speed at the wheels (ω_w) is needed.

A.1 Extended model

In figure 7.1 on page 47 the following relations are valid

$$\tau_m = i_a k_T \tag{A.1}$$

$$\tau_B = B_T \omega_m \tag{A.2}$$

$$\tau_{tot} = \tau_m - \tau_B \tag{A.3}$$

The torque τ_{tot} results in an acceleration given by

$$\alpha_m = \frac{\tau_{tot}}{J_{tot}} \tag{A.4}$$

where J_{tot} is the total moment of enertia in motors, gears, cardan shafts and wheels. Inserting and rearranging yields:

$$\dot{\omega}_m J_{tot} = k_T i_a - B_T \omega_m \tag{A.5}$$

Using the result in equation A.5 the block diagram in figure A.1 can be drawn. This diagram is an extension from the one in figure 7.5 on page 50. The following equation is the transfer function for the motor without gear seen in figure A.1

$$H(s) = \frac{\omega_m}{v_a} = \frac{k_T}{J_{tot} R_a s + R_a B_T + k_T k_e} \tag{A.6}$$

Adding the gear, SSC-controller and speed controller yields the following transfer function

$$T(s) = \frac{\omega_w}{N'_m} = \frac{\frac{k_T}{\kappa_m n}}{J_{tot} R_a s + R_a B_T + k_T k_e} \tag{A.7}$$

Equation A.7 is a first order system and this transfer function is used when applying the step.

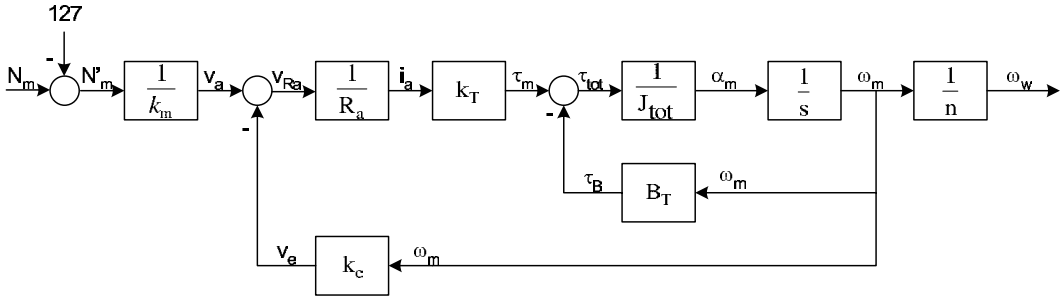

Figure A.1: *Block diagram for the extended DC-motor model*

A.2 Motor test

The test was performed without load, i.e. the wheels were not in contact with the ground at any time. To be able to get at steady count from the wheel encoders, wheel 2 was fixed, while the other wheels rotated freely. Only the values from the encoder at wheel 3 was used in the test.

The step applied to the motors had a amplitude of $N_m = 30$, and the speed of wheel 3 was logged. Figure A.2 shows the result of this test.

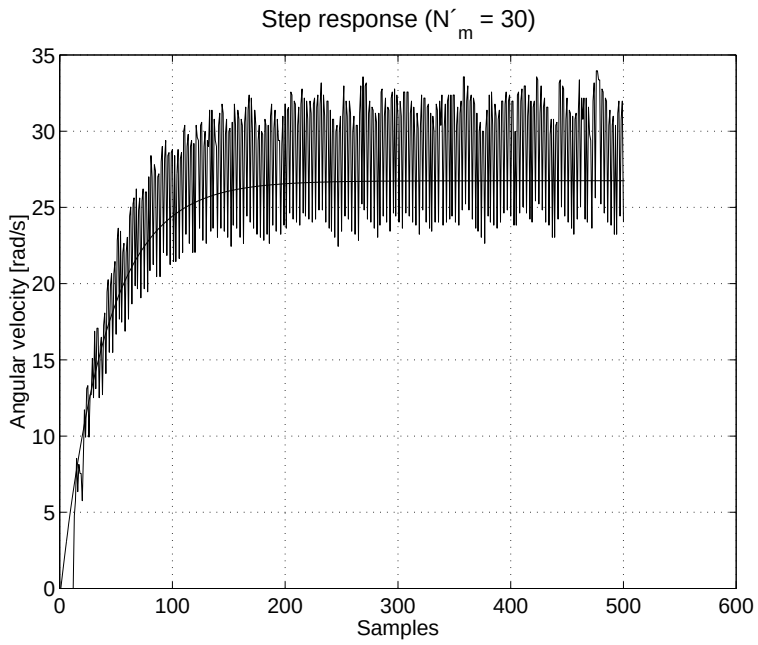

Figure A.2: *The step response for the DC-motors.*

A.3 Simulation

A step was applied to the transfer function in equation A.7. The step had the amplitude $N_m = 30$. The transfer function was tuned to fit the logged data from the real motors. The step response of the fitted transfer function is plotted as a smooth solid line in figure A.2. The fitted transfer

function for the motor is written below:

$$T(s) = \frac{\omega_w}{N'_m} = \frac{2.24 \cdot 10^{-3}}{1.018 \cdot 10^{-3}s + 2.512 \cdot 10^{-3}} \tag{A.8}$$

From this equation the constant of linear friction B_T is calculated.

$$B_T = 18.288 \cdot 10^{-3} Nms/rad \tag{A.9}$$

The result above is used in the model in section 7.1.

APPENDIX B

Hybrid Automaton

In this appendix general definitions for a hybrid automaton is presented. Depending on the used litterature the definitions for a hybrid automaton varies. The definitions presented here is in accordance with those given in [van der Schaft, 1999]. The definitions are explained with basis in the control of the robot. It is important to note that the hybrid automaton derived in this appendix only is intended as an example to aide in the understanding of the presented definitions.

A hybrid system consists of two subsystems which influence each other. The dynamics for one subsystem evolve in continuous time and the other in accordance to a discrete event system. The movement of the robot over time $\xi(t)$ is considered as the dynamics of the hybrid system, and the discrete event system is considered as various states the robot can be in. $\xi(t)$ will be denoted as ξ for simplicity. In the following sections definitions for the continuous subsystem and the discrete event subsystem are given and afterwards put together as a hybrid automaton.

B.1 The Continuous Subsystem

The dynamics of the continuous subsystem is described by differential equations. Since the movement of the robot over time is considered as the dynamics, the differential equations are given by the control law for the robot in closed loop with the system. Let these differential equations be denoted as $F(\xi, \dot{\xi}, \omega) = 0$, where ξ is the posture of the robot. ω are some continuous external variables, which may influence the system $F(\xi, \dot{\xi}, \omega) = 0$, but evolves in accordance to a different system. For the robot ω can be considered as the reference for the controller, which is equal to the trajectory $r(t)$. ξ belongs to a space denoted X, which is a three-dimensional subspace of R^3. Likewise ω belongs to a space denoted W.

B.2 The Discrete Subsystem

The discrete subsystem is described by the triple (L, A, E). L is a finite set of states $(l_1, l_2, \ldots, l_n)$ the robot can be in. A is a finite set of symbols $(a, b, c, d, ..)$ which serves as labels to a transition between to states. A transition between two states is called an event. E contains a set of rules for each event. A more precise definition for E will be given in section B.3. Lets assume that the robot can be in one of the three following states:

Find Trajectory In this state the robot moves from a posture away from the trajectory to a posture on the trajectory.

Track Trajectory In this state the robot is close to or on the trajectory and continues to track it.

Posture Stabilization The robot is close to its end destination and is moving into its final posture.

A resulting state diagram for the robot with the definitions above is shown in figure B.1.

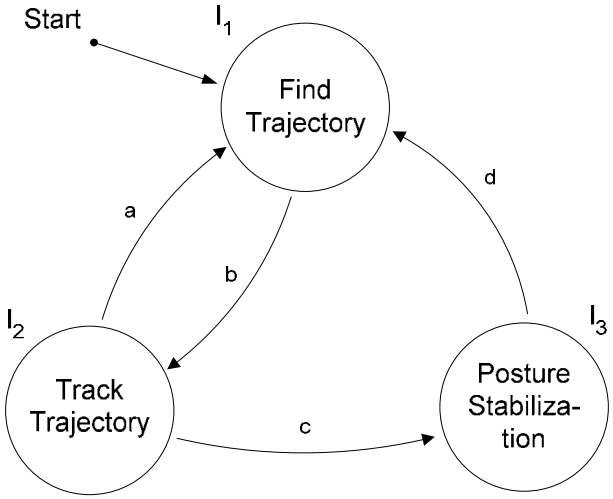

Figure B.1: *An example of a state diagram for a robot moving from one point to another. The system starts in the state l_1 as indicated with the arrow from the black dot.*

B.3 Defining the Hybrid Automaton

A hybrid automaton is defined by a seven tuple $(L, X, W, A, E, Inv, Act)$. The elements L, X, W, and A are as previously described. The hybrid automaton combines the two subsystems. Each state l_i has its own set of differential equations $F_{li}(\xi, \dot{\xi}, r) = 0$. The continuous variable ξ will then evolve according to the differential equations of the current state of the hybrid automaton. Act assigns a sets of differential equations $F_{li}(\xi, \dot{\xi}, r) = 0$ to the state l_i.

In a state, ξ may be limited to a subspace of X. Limits in a state are described by invariant equations denoted by Inv in the hybrid automaton. A more mathematically notation for Inv is therefore $Inv(l) \subset X$ for all l in L. If the hybrid automaton is in the state l_i then $\xi \in Inv(l_i)$ must be satisfied. If ξ reaches the boundary for $Inv(l_i)$, an event is triggered, making the automaton switch into a state where $\xi \in Inv(l_i)$ is satisfied. For the robot the invariant equations is defined by some thresholds for the posture. E.g. an invariant equation for the state l_3 could be:

$$Inv(l_3): \quad \Delta_{final} > |\xi - \xi_{ref}| \tag{B.1}$$

where Δ_{final} is a threshold that determines an area around the final posture, where the posture stabilization successfully can be applied.

Once a transition is forced on the hybrid automaton because of a violation of an invariant equation, the hybrid automaton may have to make a choice between several events. This choice is based on the last element in the seven tuple which is E. E is a finite set of events. Each event is subject to some rules defined by a five tuple $(l, a, Guard_{ll'}, Jump_{ll'}, l')$. The event is a

transition between the two states $l, l' \in L$ and has the label $a \in A$. $Guard_{ll'}$ is a subspace of X, $Guard_{ll'} \subset X$. For an event to occur $\xi \in Guard_{ll'}$ must be satisfied. If e.g. the robot is forced out of the state "Track Trajectory" the guards for the events with label a and c decides the next state for the hybrid automaton, (see figure B.1). A guard for the event a could be that the difference between the reference and the posture must have exceeded a given threshold $\Delta_{trajectory}$. $Jump_{ll'}$ describes a jump in the continuous state from ξ to ξ' which takes place in the transition from l to l'.

A hybrid automaton can be illustrated through a so called directed graph. A directed graph for the robot with the states described in section B.2 is seen in figure B.2.

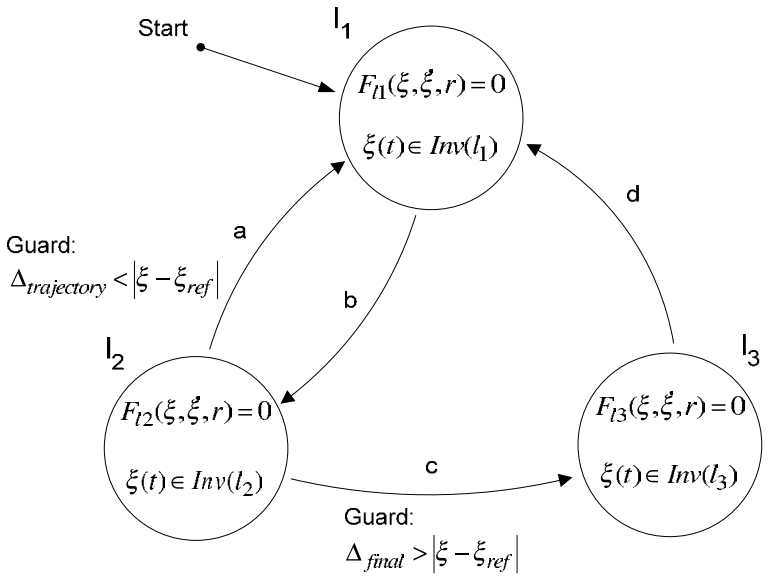

Figure B.2: *An example of a directed graph for a hybrid automaton. The figure shows how the definitions are placed on the graph.*

A event can occur in one of two ways. One is as mentioned previously that an invariant equation is violated. This is called an internal event. The other way is an external event. An external event is when the hybrid automaton is told to change state by another system. Such a system could be a supervisor for the hybrid automaton. The system outside the hybrid automaton addresses which event there is to take place through the matching label $a \in A$. Common for the internal and the external events are that the guards to an event always must be satisfied.

APPENDIX C

Users Guide for the User-Interface

In this appendix a short users guide for the user interface is given. The guide should make the reader able to compile and execute the user interface enclosed on the cd [CD, \txt1\].

C.1 Getting Started: How To Compile

Compiling the code is simple, since there is a Makefile to take care of all the compiling and linking. It is noted that the compiling will only work, if RTLinux is set up properly.

1. Copy the source-codes from the CD to a directory of your own choice.
2. In that directory, do `make`
3. If everything went well, the modules and the user interface is now compiled. Lots of messages appears on the screen. Some of them are warnings - they don't matter!

C.2 Using the User-Interface

Having compiled the code, the user-interface is available in the file `txt1`. The interface is executed by putting

```
$ ./txt1
```

With no parameters usage description is displayed on the screen. Several parameters applies. The parameters are:

start Starts the execution of the kernel modules (downloads the ASCII-file and starts the controllers).

stop Stops the kernel modules and thereby the controllers.

manual Starts the robot in manual mode. The robot is now controlled by the keyboard.

status Tells if the kernel modules are started or stopped.

reset Reset the status from above to 'stopped'. Used if the system has crashed, leaving the user interface to think that it is running although this isn't the case.

help Displays detailed help with description of possible parameters.

When running in manual mode, the controls of the robot are as shown in figure C.1. Logging of

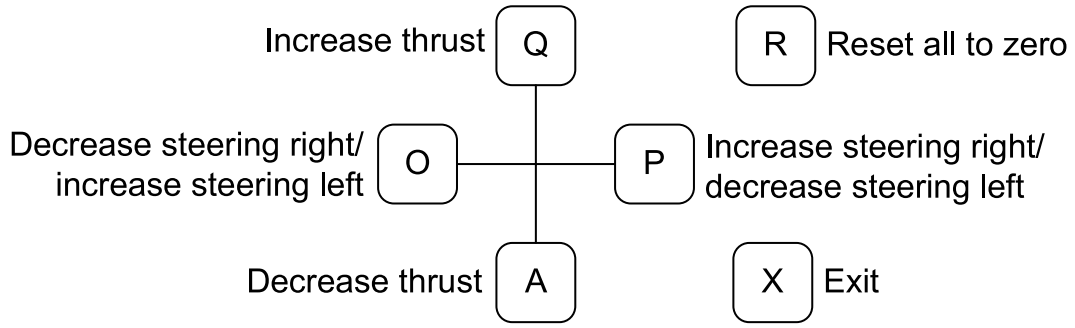

Figure C.1: *The keyboard controls used for steering the robot when manual mode is activated.*

data is also possible. Logging is activated applying the extra parameter `log` to either `start` or `manual`. Example:

```
$ ./txt1 start log
```

Executing this command will start logging of data as defined in the function `write_log` in `main.c` and in `read_log.c`.

APPENDIX D

Real-time Threads and Module Specifications

D.1 Using Real-time Threads in RTLinux

RTLinux code is implemented as Linux kernel modules as is standard when developing kernel level applications for Linux. The difference between standard Linux kernel modules and RTLinux kernel modules is the use of real-time threads. Standard Linux kernel modules doesn't support such threads. Linux kernel modules must contain the two functions `init_module` and `cleanup_module`. These functions are invoked when the kernel modules is inserted and removed from the kernel using the `insmod` and `rmmod`, respectively. `init_module` initializes the module, and `cleanup_module` deallocates any ressources used by the module. When developing RTLinux applications `init_module` is used to setup and start a real-time thread using RTLinux functions. In `cleanup_module` the real-time thread is stopped and the ressources are deallocated. The use of real-time threads in kernel modules are best illustrated by the reoccuring "Hello World" example stated in table D.1. The thread is set up in `init_module` using the function `pthread_create`. The function `start_routine` is assigned to the thread, meaning that this is the function, that will be run in the thread. The function `start_routine` consists of three parts: Initialization, main and deallocation. In the initialization part of the function, the priority of the thread is set to 1, and with the use of the function `pthread_make_periodic_np` the RTLinux scheduler is told to execute the function `start_routine` (identified by the function `pthread_self()`) once every 500000000 nanoseconds. The thread main function is a `while(1)`-loop. Every time the function `pthread_wait_np` is executed, the execution of the code is halted until the next time the scheduler executes the thread. And when the scheduler signals the thread to execute, execution is restarted from the line after the function `pthread_wait_np`. The result is that the code in the `while(1)`-loop is only run once for every time the scheduler signals the thread to run.

D.2 Module Specifications

In the following a description of each of the eight implemented modules is given. For more detailed information (the definitions of the used datastructures and the prototypes of the functions in C syntax) the reader is referred to the `.h`-files in [CD, \txt1\].

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>

pthread_t thread;
void *start_routine(void *arg) {

   /* Thread initialization */
   struct sched_param p;
   p.sched_priority = 1;
   pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);
   pthread_make_periodic_np (pthread_self(), gethrtime(), 500000000);

   /* Thread loop. Is run for every sample */
   while (1) {
     pthread_wait_np();
     rtl_printf("Hello World!\n");
   }

   /* Deallocation of thread */
   return 0;
}

int init_module(void) {
   return pthread_create (&thread, NULL, start_routine, 0);
}

void cleanup_module(void) {
   pthread_cancel (thread);
   pthread_join (thread, NULL);
}
```

Table D.1: *A simple Hello World-example using a real-time thread in RTLinux.*

D.2.1 Blackboard

Specifications for the Blackboard-module are:

- Contain shared variables.
- Shared variables can be `int` or `float`.
- Can read initializing settings from an ASCII-file.

To avoid typecasting of variables it is found neccesary to be able to contain variables of the types `int` and `float`. To be able to make changes to the setup without having to compile sourcecode, the blackboard will be initialized with values from an ASCII-file called `txt1.conf`. Each vari-

able is assigned a name and a value. By convention the names of the variables are written using a dot-notation with the group first and the element of the group last:

```
group.element
```

Recommended groups are: `odometry`, `encoder`, `control` and `misc` (for miscellaneous variables). As an example the `x` value of the `odometry` will be called `odometry.x`. Since no check is performed on the variable names in the implementation, the choice of name is in fact free. The naming conventions are only to be considered as recommendations.

The functions to be implemented in `blackboard` can be seen in table D.2. The functions `bbSetInt` and `bbSetFloat` sets a variable on the blackboard. If the variable already exists, the value is corrected to the new one. If the variable doesn't exist a new is created. `bbPrintAll` lists all the variable names and values of the variables currently on the blackboard. This function is intended for debugging purposes. The ASCII-file download that is use to initialize the blackboard

Function Name	Description
`bbGetInt()`	Returns the value of a specified variable blackboard.
`bbSetInt()`	Sets the value of a specified variable on the blackboard.
`bbGetFloat()`	Similar to `bbGetInt()`.
`bbSetFloat()`	Similar to `bbSetInt()`.
`bbInit()`	Initializes the blackboard.
`bbDestroy()`	Dealloacted the blackboard.
`bbPrintAll()`	Prints all names and values of the variables on the blackboard.

Table D.2: *The functions to be implemented in the module* `blackboard`.

with variables, is implemented using another thread. This is done to ensure that the controllers aren't started until all the variables has been downloadded from the ASCII file. For further details the reader is refered to [CD, *backslash*txt1\main.c].

D.2.2 Supervisor

The supervisor, which is executed at every sample along with the main control loop, is resposible for three things:

- Maintenance of the internal clock.
- Changing the reference of the controllers (the next posture to track) if necessary.
- Determine which controller to be used.

The internal clock is maintained by adding the time between each sample to at variable at each sample. Having maintained the clock, the reference is changed if necessary. A change is necessary if the arrival-time of the current reference is exceeded. When no more postures are in the trajectory

and the robot is at the final posture, a new trajectory is determined. Which controller to use is also determined by the supervisor using information from the blackboard. The choice is saved as an integer in the blackboard variable `control.id`. Each of the implemented control algorithms therefore needs to have an id-number assigned.

The tasks are performed using the function `supUpdate()`. This function uses the functions in the other modules to complete the tasks. Furthermore the functions `supInit` and `supDestroy` are implemented. The functions are listed in table D.3.

Function Name	Description
`supUpdate()`	Performs the tasks of the supervisor.
`supInit()`	Initializes the supervisor. (If needed).
`supDestroy()`	Dealloacted the supervisor. (If needed).

Table D.3: *The functions to be implemented in the module* `supervisor`.

D.2.3 Control

The requirements of the `Control` module is to:

- Make several control algorithms available for use.
- Choose the appropriate control algorithm based on specific criterions.
- Retrieve the current posture and change in posture from the blackboard.
- Calculate new control signals.
- Store the control signals on the blackboard.

Three functions are implemented in `Control`. The functions are listed in table D.4. `ctrlUpdate` is the actual implementation of the control algorithm. The function calculates and stores the new control signals on the blackboard. The function `ctrlUpdate()` both chooses the correct control algorithm and calculates the new control signals using the chosen control algorithm. The choice of control algorithm is based on criterions similar to guards in a hybrid automaton. The blackboard variables to be maintained by the `Control` module are shown in table D.5.

Function	Description
`ctrlUpdate()`	Calculates new control signals and stores them on the blackboard.
`ctrlInit()`	For controller initialization. (If needed).
`ctrlDestroy()`	Deallocation of the controller. (If needed).

Table D.4: *The functions implemented in the module* `Control`.

Name	Type	Description
`control.thrust`	`float`	The requested velocity for the DC-motor. [m/s].
`control.steering.front`	`float`	The requested steering angle for the front wheel pair. [rad].
`control.steering.rear`	`float`	As above, for the rear wheel pair. [rad].

Table D.5: *The blackboard variables maintained by* `control`.

D.2.4 Odometry

The Odometry-module are required to:

- Contain the implementation of the odometry model.
- Maintain blackboard variables containing the current odometry readings of x, y and θ and their time-derivatives along with η and γ_1.

The list of functions to be implemented in `Odometry` is found in table D.6. The odometry

Function	Description
`odUpdate()`	Estimation of the robot posture and η and γ_1 using the odometry model and stores the values on the blackboard.
`odInit()`	For odometry model initialization. (If needed).
`odDestroy()`	For odometry model deallocation. (If needed).

Table D.6: *Functions implemented in the module* `Odometry`.

model is implemented in the function `odUpdate`, which estimates the posture of the robot using the odometry model. The calculated values are stored on the blackboard for the controller to use. The blackboard variables to be maintained by the `odometry` module are shown in table D.7.

Name	Type	Description
`odometry.x`	`int`	The position in x calculated by the odometry model.
`odometry.y`	`int`	The position in y calculated by the odometry model.
`odometry.theta`	`float`	The orientation theta calculated by the odometry model.
`odometry.dotx`	`int`	The change in x.
`odometry.doty`	`int`	The change in y.
`odometry.dottheta`	`float`	The change in the angle theta.
`odometry.eta`	`float`	The value of η calculated using the odometry model.
`odometry.gamma1`	`float`	The steering angle γ_1 found by odometry.

Table D.7: *The blackboard variables maintained by* `Odometry`.

D.2.5 Hardware Abstraction Layer (HAL)

The HAL module will contain the functions able to communicate with the servo and DC-motor controller and the encoder-interface (PIC). The functions to be implemented are listed in table D.8. `halReadEncoders` reads the values of the encoders placed at each front wheel. The

Function	Description
`halActuate()`	Applies the control signals to the servos and the DC-motor.
`halReadEncoders()`	Reads the values of the encoders, and places them on the blackboard.
`halInit()`	Initialization of the module. (If needed).
`halDestroy()`	Deallocation of the module. (If needed).

Table D.8: *Functions implemented in the module HAL.*

change in value of each of the encoders with respect to the previous reading are stored on the blackboard along with encoder values. The other way around, the function `halActuate` applies the values of the blackboard variables from table D.5 to the servos and the DC-motor. The used blackboard variables are seen in table D.9.

Name	Type	Description
`encoder.w2`	`int`	The current encoder value of wheel 2.
`encoder.w3`	`int`	The current encoder value of wheel 3.
`encoder.delta.w2`	`int`	The change in the current encoder value with respect to the previous reading for wheel 2.
`encoder.delta.w3`	`int`	As above, for wheel 3.

Table D.9: *The blackboard variables maintained by `hal`.*

D.2.6 Pathgen

The `Pathgen` module is responsible of the generation of trajectories for the robot track. The

Function	Description
`pgGenerate()`	Generates the trajectory to be followed.
`pgInit()`	Initialization of the module. (If needed).
`pgDestroy()`	Deallocation of the module. (If needed).
`pgUpdate()`	Evaluates if the current reference posture is correct. Replaces the current reference if necessary.

Table D.10: *Functions implemented in the module `Pathgen`.*

function `pgGenerate` generates a path from a startpoint to an endpoint. At the present implemetation the generated path is a straigth line between the two points. The found trajectory is saved in a variable to be used by the supervisor when setting the reference for the robot.

D.2.7 Main

The module `main` is the main loop of the implementation. The execution progress illustrated in figure 12.3 will be implemented here. No functions will be available for other modules in this module.

D.2.8 Functions

The module `functions` are considered as a toolbox, consisting of functions that are (or may be) needed by more than two of the other modules. The list of functions to be implemented in the module are in table D.11.

Function	Description
`rtl_print_float()`	Prints a float on screen.
`print_matrix()`	Prints the contents of a matrix on screen
`init_matrix()`	Initializes a matrix to the value zero for all elements.
`add_matrices()`	Adds two matrices.
`subtract_vectors()`	Subtracts two vectors.
`transpose_matrix()`	Transposes a matrix.
`mult_matrices()`	Multiplies two matrices.
`scale_matrix()`	Multiplies a scalar to all elements of a matrix.

Table D.11: *Functions implemented in the module* `Functions`.

APPENDIX E

RTLinux Installation Guide

This appendix describes howto install RTLinux 3.2-pre2 on a Redhat Linux 9.0 system. The files to be used can be found on the enclosed CD, [CD, \rtlinux\]. The files are:

rtlinux-3.2-pre2.tar.bz2 The source codes for RTLinux 3.2 prerelease 2.

linux-2.4.20.tar.bz2 The source codes for the Linux kernel version 2.4.20 suitable for RTLinux installation.

Important note: The Linux kernel source codes distributed with RedHat Linux (e.g. 2.4.20-8) is not usable for installing RTLinux. A “clean” version of the kernel sources must be used.

The steps of the installation are as follows:

1. Make sure you have RedHat set up appropriately. When installing RedHat you need to perform a Custom install and select Development, Kernel Development, Utilities, and Select Individual Packages.

2. Copy the files from the CD to `/root/` or goto

   `ftp.rtlinux.com/pub/rtlinux`
   `ftp.kernel.org/pub/linux/kernel/`

   to get newer versions. There is no guarantee that this installation guide will apply to other versions than the ones used here!

3. Unpack the files using `bunzip2` and `tar` into `/usr/src/rtlinux/`:

```
$ cd /root
$ bunzip2 linux-2.4.20.tar.bz2
$ bunzip2 rtlinux-3.2-pre2.tar.bz2
$ rm -rf /usr/src/rtlinux
$ mkdir /usr/src/rtlinux
$ cd /usr/src/rtlinux
$ tar -xvf /root/linux-2.4.20.tar
$ tar -xvf /root/rtlinux-3.2-pre2.tar
```

4. Patch the Linux kernel source codes with the RTLinux source codes:

```
$ cd linux-2.4.20
$ patch -p1 < /usr/src/rtlinux/rtlinux-3.2-pre2/patches/\\
  kernel_patch-2.4.20-rtl3.2-pre2
```

5. Configure the Linux kernel:

```
$ make xconfig
```

   Make sure, that APM Support is disabled and that no generic driver for the serial port is installed (if so, RTLinux' excellent serial port driver will not work).

6. Compile and install the Linux kernel and the modules:

```
$ make dep
$ make bzImage
$ make modules
$ make modules_install
$ cp arch/i386/boot/bzImage /boot/rtzImage
```

7. Configure the bootloader (here GRUB is used). Edit the file `/boot/grub/grub.conf` so it looks something like:

```
default=0 timeout=10
splashimage=(hd0,1)/grub/splash.xpm.gz
title RTLinux (2.4.20-rtl3.2-pre2)
       root (hd0,1)
       kernel /rtzImage ro
       root=/dev/hda6
title Red Hat Linux (2.4.20-8)
       root (hd0,1)
       kernel /vmlinuz-2.4.20-8 ro
       root=LABEL=/ initrd /initrd-2.4.20-8.img
```

   Here the root partition `/` is placed on `/dev/hda6`. This is dependent on the disk partitioning performed during install, and therefore it might be different in other installations. Reboot the machine, and make sure you start up using the newly compiled kernel.

8. Configure and compile RTLinux

```
$ cd /usr/src/rtlinux/rtlinux-3.2-pre2
$ ln -sf /usr/src/rtlinux/linux-2.4.20 linux
$ make xconfig
$ make dep
$ make
$ make devices
$ make install
```

9. Post installation.To be able to run any real time programs, you need to load the RTLinux modules:

   ```
   $ rtlinux start
   ```

   To start RTLinux automatically upon boot (if you select the rtlinux kernel), edit /etc/rc.d/rc.local and put the following at the end of the file:

   ```
   if [ ‘cat /proc/sys/kernel/osrelease‘ = "2.4.20-rtl3.2-pre2" ];then
     /usr/bin/rtlinux start
   fi
   ```

   Important note: Be sure to copy the text exactly. Using the wrong accents or removing spaces can cause the code not to work!

10. Additional information. It is strongly advised to sign up for the RTLinux mailing-list on `www.fsmlabs.com`. There is also an archive of old postings which can help solving many problems.

APPENDIX F

Calculations

In this appendix calculations of different substantial equations are shown. In section F.1 an expression for the kinetic energy T_{Total} of a wheel is found. In section F.2 the expression behind the short hand notation used in the Lagrangian formalism is derived.

F.1 Energy of a Wheel

In this section more substantial calculations are shown for the deviation of the total kinetic energy. The energy of the wheel is given by the equation:

$$T_W = \frac{1}{2}\int_0^{2\pi} (\dot{x}_P^2 + \dot{y}_P^2 + \dot{z}_P^2) p\, d\psi$$

$$\Downarrow$$

$$T_W = \frac{1}{2}\left(\int_0^{2\pi} \dot{x}_P^2 p\, d\psi + \int_0^{2\pi} \dot{y}_P^2 p\, d\psi + \int_0^{2\pi} \dot{z}_P^2 p\, d\psi\right) \tag{F.1}$$

In the following each of the three terms $\dot{x}_P^2$, $\dot{y}_P^2$ $\dot{z}_P^2$ is derived and integrated.

Deriving $\dot{x}_P^2$

$$\begin{aligned}
\dot{x}_P^2 \;=\;& \dot{x}^2 - 2\dot{x}l\dot{\theta}\sin(\alpha+\theta) - \\
& 2\dot{x}r\dot{\psi}\sin(\psi)\cos(\gamma+\theta) - \\
& 2\dot{x}r(\dot{\gamma}+\dot{\theta})\cos(\psi)sin(\gamma+\theta) + \\
& l^2\dot{\theta}^2\sin^2(\alpha+\theta) + \\
& 2l\dot{\theta}r\dot{\psi}\sin(\alpha+\theta)\sin(\psi)\cos(\gamma+\theta) + \\
& 2lr(\dot{\theta}\dot{\gamma}+\dot{\theta}^2)\sin(\alpha+\theta)\cos(\psi)\sin(\gamma+\theta) + \\
& r^2\dot{\psi}^2\sin^2(\psi)\cos^2(\gamma+\theta) + \\
& 2r^2(\dot{\psi}\dot{\gamma}+\dot{\theta}\dot{\psi})\cos(\psi)\sin(\psi)\cos(\gamma+\theta)\sin(\gamma+\theta) + \\
& r^2(\dot{\gamma}^2+2\dot{\theta}\dot{\gamma}+\dot{\theta}^2)\cos^2(\psi)\sin^2(\gamma+\theta)
\end{aligned}$$

$$\int_0^{2\pi} \dot{x}_P^2 p_w d\psi = 2\dot{x}^2\pi p_w - 4\dot{x}l\dot{\theta}\pi p_w \sin(\alpha+\theta) + 2l^2\dot{\theta}^2\pi p_w \sin^2(\alpha+\theta) + r^2\dot{\psi}^2\pi p_w \cos^2(\gamma+\theta) + r^2(\dot{\gamma}^2 + 2\dot{\theta}\dot{\gamma} + \dot{\theta}^2)\pi p_w \sin^2(\gamma+\theta) \quad \text{(F.2)}$$

Deriving $\dot{y}_P^2$

$$\begin{aligned}\dot{y}_P^2 = {} & \dot{y}^2 + 2\dot{y}l\dot{\theta}\cos(\alpha+\theta) - \\ & 2\dot{y}r\dot{\psi}\sin(\psi)\sin(\gamma+\theta) - \\ & 2\dot{y}r(\dot{\gamma}+\dot{\theta})\cos(\psi)cos(\gamma+\theta) + \\ & l^2\dot{\theta}^2\cos^2(\alpha+\theta) + \\ & 2l\dot{\theta}r\dot{\psi}\cos(\alpha+\theta)\sin(\psi)\sin(\gamma+\theta) + \\ & 2lr(\dot{\theta}\dot{\gamma}+\dot{\theta}^2)\cos(\alpha+\theta)\sin(\psi)\sin(\gamma+\theta) + \\ & r^2\dot{\psi}^2\sin^2(\psi)\sin^2(\gamma+\theta) + \\ & 2r^2\dot{\psi}(\dot{\gamma}+\dot{\theta})\cos(\psi)\sin(\psi)\cos(\gamma+\theta)\sin(\gamma+\theta) + \\ & r^2(\dot{\gamma}^2+2\dot{\theta}\dot{\gamma}+\dot{\theta}^2)\cos^2(\psi)\cos^2(\gamma+\theta)\end{aligned}$$

$$\int_0^{2\pi} \dot{y}_P^2 p_w d\psi = 2\dot{y}^2\pi p_w + 4\dot{y}l\dot{\theta}\pi p_w \cos(\alpha+\theta) + 2l^2\dot{\theta}^2\pi p_w \cos^2(\alpha+\theta) + r^2\dot{\psi}^2\pi p_w \sin^2(\gamma+\theta) + r^2(\dot{\gamma}^2+2\dot{\theta}\dot{\gamma}+\dot{\theta}^2)\pi p_w \cos^2(\gamma+\theta) \quad \text{(F.3)}$$

Deriving $\dot{z}_P^2$

$$\dot{z}_P^2 = r^2\dot{\psi}^2\cos^2(\psi) \quad \text{(F.4)}$$

$$\int_0^{2\pi} \dot{z}_P^2 p_w d\psi = r^2\dot{\psi}^2\pi p_w \quad \text{(F.5)}$$

Result

Inserting equation F.2,F.3 and F.5 in equation F.1 yields:

$$
\begin{aligned}
T_W &= \frac{1}{2}\dot{x}^2 2p\pi + \frac{1}{2}\dot{y}^2 2p\pi + 2p\pi l\dot{\theta}(\dot{y}cos(\alpha+\theta) - \dot{x}sin(\alpha+\theta)) + \\
&\quad \frac{1}{2}l^2\dot{\theta}^2 2p\pi + \frac{1}{2}r^2\dot{\theta}^2\dot{\psi}^2 2p\pi + \frac{1}{4}r^2(\dot{\gamma}^2 2p\pi + 2\dot{\theta}\dot{\gamma} + \dot{\theta}^2)
\end{aligned} \tag{F.6}
$$

The following two relations are known:

$$
m = 2\pi p \tag{F.7}
$$

$$
I_W = mr^2 \tag{F.8}
$$

Substituting equation F.7 and F.8 in equation F.6 yields:

$$
\begin{aligned}
T_W &= \frac{1}{2}\dot{x}^2 m + \frac{1}{2}\dot{y}^2 m + ml\dot{\theta}\big(\dot{y}cos(\alpha+\theta) - \dot{x}sin(\alpha+\theta)\big) \\
&\quad + \frac{1}{2}l^2\dot{\theta}^2 m + \frac{1}{2}I_W\dot{\theta}^2\dot{\psi}^2 + \frac{1}{4}I_W\dot{\gamma}^2 + \frac{1}{2}I_W\dot{\theta}\dot{\gamma} + \frac{1}{4}I_W\dot{\theta}^2
\end{aligned} \tag{F.9}
$$

Which is the result used in section 6.2.

F.2 Calculation of $[T]_\xi$, $[T]_\psi$ and $[T]_\gamma$

The calculation of $[T]_\xi$ is longer that $[T]_\psi$ and $[T]_\gamma$ and is therefore devided into the calculation of the two terms $\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\xi}}\right)$ and $\frac{\partial T}{\partial \xi}$. To reduce the length of the expressions in the calculations of $[T]_\xi$, $[T]_\psi$ and $[T]_\gamma$ the matrix M is written as:

$$
M = \begin{bmatrix} M_1 & 0 & M_2 \\ 0 & M_1 & M_3 \\ 0 & 0 & M_4 \end{bmatrix} \tag{F.10}
$$

In section 6.3 the total kinetic energy is found as:

$$
T_{Total} = T_F + \sum_{i=1}^{4} T_W i \tag{F.11}
$$

$$
\begin{aligned}
&= \frac{1}{2}\dot{x}^2(M+4m) + \frac{1}{2}\dot{y}^2(M+4m) + \frac{1}{2}\dot{\theta}^2(4l^2m + 2I_W + d^2M + I_F) \\
&\quad + \frac{1}{4}I_W\sum_{i=1}^{4}\dot{\gamma}_i^2 + \frac{1}{4}I_W\sum_{i=1}^{4}\dot{\psi}_i^2 + \frac{1}{2}I_W\dot{\theta}\sum_{i=1}^{4}\dot{\gamma}_i \\
&\quad - \dot{x}\dot{\theta}\left(2dM\sin(\theta+\rho) + ml\sum_{i=1}^{4}\sin(\alpha_i+\theta)\right) \\
&\quad + \dot{y}\dot{\theta}\left(2dM\cos(\theta+\rho) + ml\sum_{i=1}^{4}\cos(\alpha_i+\theta)\right)
\end{aligned} \tag{F.12}
$$

With T defined as above the first term in $[T]_{\xi}$ can be calulated as:

$$\begin{aligned}
\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\xi}}\right) &= \frac{d}{dt}\begin{bmatrix} \frac{\partial T}{\partial \dot{x}} \\ \frac{\partial T}{\partial \dot{y}} \\ \frac{\partial T}{\partial \dot{\theta}} \end{bmatrix} \\
&= \frac{d}{dt}\begin{bmatrix} \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} R^T(\theta) M R(\theta)\dot{\xi}^T + \frac{1}{2} R^T(\theta) M R(\theta)\dot{\xi}^T \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ \frac{1}{2}\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} R^T(\theta) M R(\theta)\dot{\xi}^T + \frac{1}{2} R^T(\theta) M R(\theta)\dot{\xi}^T \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \\ \frac{1}{2}\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R^T(\theta) M R(\theta)\dot{\xi}^T + \frac{1}{2} R^T(\theta) M R(\theta)\dot{\xi}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ +\frac{1}{2} I_{WV}\dot{\gamma} \end{bmatrix} \qquad \text{(F.13)} \\
&= \frac{d}{dt}\begin{bmatrix} \dot{x}M_1 + \frac{1}{2}\dot{\theta}(M_2\cos(\theta) - M_3\sin(\theta)) \\ \dot{y}M_1 + \frac{1}{2}\dot{\theta}(M_2\sin(\theta) + M_3\cos(\theta)) \\ \frac{1}{2}\big(2\dot{\theta}M_4 + \dot{x}(M_2\cos(\theta) - M_3\sin(\theta)) + \dot{y}(M_2\sin(\theta) \\ +M_3\cos(\theta)) + I_{WV}\dot{\gamma}\big) \end{bmatrix} \qquad \text{(F.14)} \\
&= \frac{d}{dt}\left(P\dot{\xi} + \frac{1}{2}K I_{WV}\gamma\right) \qquad \text{(F.15)}
\end{aligned}$$

Where:

$$P = \begin{bmatrix} M_1 & 0 & \frac{1}{2}(M_2\cos(\theta) - M_3\sin(\theta)) \\ 0 & M_1 & \frac{1}{2}(M_2\sin(\theta) + M_3\cos(\theta)) \\ \frac{1}{2}(M_2\cos(\theta) - M_3\sin(\theta)) & \frac{1}{2}(M_2\sin(\theta) + M_3\cos(\theta)) & M_4 \end{bmatrix}$$

$$K = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The second term of $[T]_{xi}$ is:

$$\begin{aligned}
\frac{\partial T}{\partial \xi} &= \begin{bmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \\ \frac{\partial T}{\partial \theta} \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \\ -2\dot{x}\dot{\theta}^2 dM\cos(\theta+\rho) - 4\dot{x}\dot{\theta}^2 ml\cos(\alpha+\theta) - \\ 2\dot{y}\dot{\theta}^2 dM\sin(\alpha+\theta) - 4\dot{y}\dot{\theta}^2 ml\sin(\alpha+\theta) \end{bmatrix} \\
&= K\dot{\xi}^T R^T(\theta) \begin{bmatrix} 0 & 0 & -\theta M_3 \\ 0 & 0 & \theta M_2 \\ 0 & 0 & O \end{bmatrix} R(\theta)\dot{\xi} \qquad \text{(F.16)}
\end{aligned}$$

$[T]_\xi$

From equation F.15 and F.16 $[T]_\xi$ is given as:

$$\begin{aligned}
[T]_\xi &= \frac{d}{dt}\left(P\dot{\xi} + \frac{1}{2}KI_{WV}\gamma\right) - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \dot{\xi}^T R^T(\theta) \begin{bmatrix} 0 & 0 & -\theta M_3 \\ 0 & 0 & \theta M_2 \\ 0 & 0 & O \end{bmatrix} R(\theta)\dot{\xi} \\
&= \dot{P}\dot{\xi} + P\ddot{\xi} + \frac{1}{2}KI_{WV}\dot{\gamma} - K_1\dot{\xi}^T R^T(\theta) N R(\theta)\dot{\xi} \qquad \text{(F.17)}
\end{aligned}$$

Where N is the matrix:

$$N = \begin{bmatrix} 0 & 0 & -\frac{1}{2}\theta M_3 \\ 0 & 0 & \frac{1}{2}\theta M_2 \\ -\frac{1}{2}\theta M_3 & \frac{1}{2}\theta M_2 & O \end{bmatrix}$$

$[T]_\psi$

$$\begin{aligned}
[T]_\psi &= \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\psi}}\right) - \frac{\partial T}{\partial \psi} = \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\psi}}\right) = \frac{d}{dt}\left(\frac{\partial(\frac{1}{4}\dot{\psi}^T I_{WM}\dot{\psi})}{\partial \psi}\right) \\
&= \frac{d}{dt}\left(\frac{1}{4}\frac{\partial(\dot{\psi}^T)}{\partial \dot{\psi}} I_{WM}\dot{\psi} + \frac{1}{4}\dot{\psi}^T I_{WM}\frac{\partial(\dot{\psi})}{\partial \dot{\psi}}\right) \\
&= \frac{d}{dt}\left(\frac{1}{2}\frac{\partial(\dot{\psi}^T)}{\partial \dot{\psi}} I_{WM}\dot{\psi}\right) \\
&= \frac{1}{2}\frac{d}{dt}\left(I_{WM}\dot{\psi}\right) = \frac{1}{2}I_{WM}\ddot{\psi} \qquad \text{(F.18)}
\end{aligned}$$

Where $K_2^T = [1\ 1\ 1\ 1]$.

$[T]_\gamma$

$$
\begin{aligned}
[T]_\gamma &= \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\gamma}}\right) - \frac{\partial T}{\partial \gamma} = \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\gamma}}\right) \\
&= \frac{d}{dt}\left(\frac{\partial(\frac{1}{4}\dot{\gamma}^T I_{WM}\dot{\gamma})}{\partial \dot{\gamma}} + \frac{\partial(\frac{1}{2}\dot{\theta} I_W \dot{\gamma})}{\partial \dot{\gamma}}\right) \\
&= \frac{d}{dt}\left(\frac{1}{4}\frac{\partial(\dot{\gamma}^T)}{\partial \dot{\gamma}} I_{WM}\dot{\gamma} + \frac{1}{4}\dot{\gamma}^T I_{WM}\frac{\partial(\dot{\gamma})}{\partial \dot{\gamma}}\right) + \frac{d}{dt}\left(\frac{1}{2}\dot{\theta} I_W \frac{\partial(\gamma)}{\partial \dot{\gamma}}\right) \\
&= \frac{d}{dt}\left(\frac{1}{2}\frac{\partial(\dot{\gamma}^T)}{\partial \dot{\gamma}} I_{WM}\dot{\gamma}\right) + \frac{d}{dt}\left(\frac{1}{2}\dot{\theta} I_W K_2\right) \\
&= \frac{1}{2} I_{WM}\ddot{\gamma} + \frac{1}{2}\ddot{\theta} I_W K_2
\end{aligned}
\tag{F.19}
$$

F.3 Determining the Rank of $C_1(\gamma)$

Since $\Sigma(\gamma)$ is the null space of $C_1(\gamma)$, the rank of $C_1(\gamma)$ is considered to see if the calculations can be reduced. The intention is to show that the effective rank of $C_1(\gamma)$ is 2, making the following statement true:

$$
\begin{aligned}
C_1(\gamma) &= \begin{bmatrix} \sin\gamma_1 & -\cos\gamma_1 & -l_1\cos(\alpha_1-\gamma_1) \\ \sin\gamma_2 & -\cos\gamma_2 & -l_2\cos(\alpha_2-\gamma_2) \\ \sin\gamma_3 & -\cos\gamma_3 & -l_3\cos(\alpha_3-\gamma_3) \\ \sin\gamma_4 & -\cos\gamma_4 & -l_4\cos(\alpha_4-\gamma_4) \end{bmatrix} \\
&\sim \begin{bmatrix} \sin\gamma_i & -\cos\gamma_i & -l_i\cos(\alpha_i-\gamma_i) \\ \sin\gamma_j & -\cos\gamma_j & -l_j\cos(\alpha_j-\gamma_j) \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = C_1^*(\gamma)
\end{aligned}
\tag{F.20}
$$

If this is possible the null space $\Sigma(\gamma)$ is more easily calculated as $\Sigma(\gamma) = \mathcal{N}\left(C_1^*(\gamma)\right)$ instead of $\Sigma(\gamma) = \mathcal{N}\left(C_1(\gamma)\right)$.

Using algebra it can be shown that the rank of $C_1(\gamma)$ is 2 when $\gamma_i = \gamma_j = 0$. First row reduction is applied to $C_1(\gamma)$ keeping in mind that $\gamma_1 = \gamma_4$, $\gamma_2 = \gamma_3$ and $l_1 = l_2 = l_3 = l_4 = l$:

$$
\begin{aligned}
C_1(\gamma) &= \begin{bmatrix} \sin\gamma_1 & -\cos\gamma_1 & -l_1\cos(\alpha_1-\gamma_1) \\ \sin\gamma_2 & -\cos\gamma_2 & -l_2\cos(\alpha_2-\gamma_2) \\ \sin\gamma_3 & -\cos\gamma_3 & -l_3\cos(\alpha_3-\gamma_3) \\ \sin\gamma_4 & -\cos\gamma_4 & -l_4\cos(\alpha_4-\gamma_4) \end{bmatrix} \\
&\sim \begin{bmatrix} \sin\gamma_1 & -\cos\gamma_1 & -l\cos(\alpha_1-\gamma_1) \\ \sin\gamma_2 & -\cos\gamma_2 & -l\cos(\alpha_2-\gamma_2) \\ \sin\gamma_2 & -\cos\gamma_2 & -l\cos(\alpha_3-\gamma_2) \\ \sin\gamma_1 & -\cos\gamma_1 & -l\cos(\alpha_4-\gamma_1) \end{bmatrix} \\
&\sim \begin{bmatrix} \sin\gamma_1 & -\cos\gamma_1 & -l\cos(\alpha_1-\gamma_1) \\ \sin\gamma_2 & -\cos\gamma_2 & -l\cos(\alpha_2-\gamma_2) \\ 0 & 0 & -l\left(\cos(\alpha_3-\gamma_2)-\cos(\alpha_2-\gamma_2)\right) \\ 0 & 0 & -l\left(\cos(\alpha_4-\gamma_1)-\cos(\alpha_1-\gamma_1)\right) \end{bmatrix}
\end{aligned}
\tag{F.21}
$$

To ensure that the rank is 2, the following must be satisfied:

$$\cos(\alpha_3-\gamma_2)-\cos(\alpha_2-\gamma_2)=0 \tag{F.22}$$

$$\cos(\alpha_4-\gamma_1)-\cos(\alpha_1-\gamma_1)=0 \tag{F.23}$$

The angles α_i can be expressed by the angle α_1:

$$\alpha_1=\alpha_1 \tag{F.24}$$

$$\alpha_2=\pi-\alpha_1 \tag{F.25}$$

$$\alpha_3=\pi+\alpha_1 \tag{F.26}$$

$$\alpha_4=2\pi-\alpha_1 \tag{F.27}$$

Considering equation F.22:

$$
\begin{aligned}
&\cos(\alpha_3-\gamma_2)-\cos(\alpha_2-\gamma_2)=0 \\
&\Downarrow \\
&\cos(\pi+\alpha_1-\gamma_2)-\cos(\pi-\alpha_1-\gamma_2)=0 \\
&\Downarrow \\
&2\sin(\pi-\gamma_2)\sin(\alpha_1)=0 \\
&\Downarrow \\
&\sin(\pi-\gamma_2)=0 \\
&\Downarrow \\
&\gamma_2=0\pm p\pi, \quad p=1,2,3\ldots
\end{aligned}
\tag{F.28}
$$

Extending the calculations to equation F.23 a similar result is obtained for γ_1:

$$\gamma_1=0\pm p\pi, \quad p=1,2,3\ldots \tag{F.29}$$

Since the maximum steering angles are $-\frac{\pi}{9}$ and $\frac{\pi}{9}$ the only steering angle of interest here is $\gamma_1 = \gamma_2 = 0$. Thus the rank of $C_1(\gamma)$ is only 2 if $\gamma_1 = \gamma_2 = 0$:

$$\text{rank}\,(C_1(\gamma)) = 0 \Leftrightarrow \gamma_1 \;=\; \gamma_2 \;=\; 0 \tag{F.30}$$

In order to investigate the rank when the angles are not zero, the rank of the matrix is estimated. A reliable way to estimate the rank is to count the number of nonzero singular values. Small nonzero singular values are assumed to be zero for practical purposes, and the effective rank of a matrix is the number obtained by counting the remaining nonzero singular values [Lay, 1997]. A simulation of the singular values as a function of the steering angles are shown in figure F.1 and F.2. Two cases are evaluated: The case where $\gamma_1 = \gamma_2$ and the case where $\gamma_2 = -\gamma_1$.

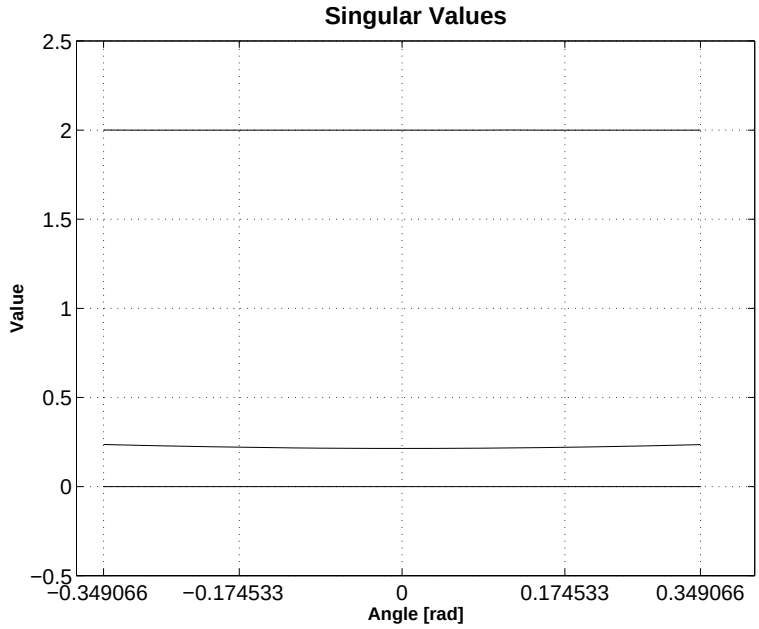

Figure F.1: *The singualar values of $C_1(\gamma)$ found using SVD for $\gamma_1 = \gamma_2$ varied from $-\frac{\pi}{9}$ to $\frac{\pi}{9}$.*

Figure F.1 reveals that the rank is 2 if the steering angles are equal ($\gamma_1 = \gamma_2$), or in other words if the robot is moving in a straight line, since the smallest singular value is zero for all angles.

Figure F.2 confirms that the rank is 2 when the angles are both zero (as seen above), since the smallest singular value is zero in that point. But for angles not equal to zero, the smallest singular value increases to 0.1185. Although this singular value is relatively high, it is still assumed that the rank of $C_1(\gamma)$ is 2 when $\gamma_2 = -\gamma_1$.

F.4 Determining η and γ_1

The variables η and γ_1 are calulated using other known variables as shown in the following. From the kinematic model, it is known that:

$$\dot{\xi} = R^T(\theta)\Sigma_L(\gamma)\eta(t) \tag{F.31}$$

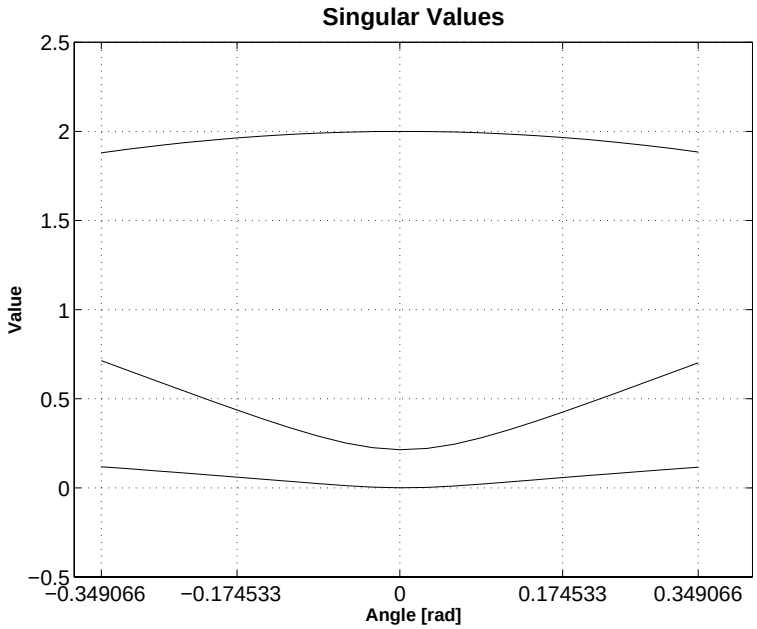

Figure F.2: *The singular values of $C_1(\gamma)$ for $\gamma_2 = -\gamma_1$. The singluar values are shown for the interval $-\frac{\pi}{9}$ to $\frac{\pi}{9}$.*

Since η and γ_1 are to be calculated for each sampling instant, the time dependency for η can be omitted, resulting in:

$$\dot{\xi} = R^T(\theta)\Sigma_L(\gamma)\eta \tag{F.32}$$

Letting $\gamma_2 = 0$, and calculating for $\dot{x}$, $\dot{y}$ and $\dot{\theta}$ yields:

$$\dot{x} = (\cos\theta\,(\ell\cos(\alpha_1-\gamma_1) - \ell\cos(\gamma_1)\cos(-\alpha_2)) + \sin\theta\,(-\ell\sin(\gamma_1)\cos(-\alpha_2)))\,\eta \tag{F.33}$$

$$\dot{y} = (-\sin\theta\,(\ell\cos(\alpha_1-\gamma_1) - \ell\cos(\gamma_1)\cos(-\alpha_2)) + \cos\theta\,(-\ell\sin(\gamma_1)\cos(-\alpha_2)))\,\eta \tag{F.34}$$

$$\dot{\theta} = \sin(\gamma_1)\eta \tag{F.35}$$

Rearranging equation F.35 results in an expression for η:

$$\eta = \frac{\dot{\theta}}{\sin(\gamma_1)} \tag{F.36}$$

Inserting equation F.36 in equations F.33 and F.34 then reduces to:

$$\dot{x} = (\cos\theta\,(\ell\cos(\alpha_1-\gamma_1) - \ell\cos(\gamma_1)\cos(-\alpha_2)))\,\eta + \sin\theta K \tag{F.37}$$

$$\dot{y} = (-\sin\theta\,(\ell\cos(\alpha_1-\gamma_1) - \ell\cos(\gamma_1)\cos(-\alpha_2)))\,\eta + \cos\theta K \tag{F.38}$$

With

$$K = \ell\cos(-\alpha_2)\dot{\theta} \tag{F.39}$$

Using that $\cos(\alpha_1-\gamma_1) = \cos\alpha_1\cos\gamma_1 + sin\alpha_1\sin\gamma_1$ equation F.38 is equal to:

$$\dot{y} = -\left(\ell\sin\theta(\cos\alpha_1\cos\gamma_1 + \sin\alpha_1\sin\gamma_1 - \cos\gamma_1\cos(-\alpha_2))\right)\eta + \cos\theta K \tag{F.40}$$

Once again inserting equation F.36 yields:

$$\dot{y} = -\ell\sin\theta\cot\gamma_1(\cos\alpha_1 - \cos\alpha_2)\dot{\theta} - \ell\sin\theta\sin\alpha_1\dot{\theta} + \cos\theta K \tag{F.41}$$

with

$$\cot(\gamma_1) = \frac{\cos(\gamma_1)}{\sin(\gamma_1)} \tag{F.42}$$

Rearranging the expression solving for γ_1 results in:

$$\gamma_1 = \arctan\left\{\frac{-\ell \sin\theta(\cos\alpha_1 - \cos\alpha_2)\dot{\theta}}{\dot{y} + \ell \sin\theta \sin\alpha_1 \dot{\theta} - \cos\theta K}\right\} \tag{F.43}$$

Normally η would have been determine now using equation F.36. However this is not a valid solution to determining η since the equation implies that without a change in orientation$\dot{\theta}$, η must be zero. This is not true. In fact the expression F.36 is only valid for determining γ_1 as a function of $\dot{\theta}$. Instead an expression for η is found by rearranging expression F.33:

$$\eta = -\frac{\dot{x}}{\ell(-\cos\theta\cos(\alpha_1 - \gamma_1) + \cos\theta\cos\gamma_1\cos\alpha_2 + \sin\theta\sin\gamma_1\cos\alpha_2)} \tag{F.44}$$

Thereby expressions for η and γ_1 has been determined.

APPENDIX G

Matrices

G.1 General

The change in generalizes coordinates:

$$\dot{q} = \begin{bmatrix} \dot{\xi} \\ \dot{\gamma} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\gamma}_1 \\ \dot{\gamma}_2 \\ \dot{\gamma}_3 \\ \dot{\gamma}_4 \\ \dot{\psi}_1 \\ \dot{\psi}_2 \\ \dot{\psi}_3 \\ \dot{\psi}_4 \end{bmatrix} = \begin{bmatrix} R(\theta)\Sigma(\gamma) & 0 \\ 0 & I_4 \\ J_2^{-1} J_1(\gamma)\Sigma(\gamma) & 0 \end{bmatrix} \begin{bmatrix} \eta \\ \dot{\gamma} \end{bmatrix} \tag{G.1}$$

The rotation of a point i frame I to E is performed with the rotation matrix

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{G.2}$$

The rotation from frame $\mathbb{E}$ to $\mathbb{I}$ is performed with the transpose of equation G.2.

$$\dot{R}^T(\theta) = \dot{\theta} \begin{bmatrix} -\sin\theta & -\cos\theta & 0 \\ \cos\theta & -\sin\theta & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{G.3}$$

G.2 Kinematic Model

One of two matrices expressing the no slip-constraint for the four wheels. 4×3:

$$J_1(\gamma) = \begin{bmatrix} \cos(\gamma_1) & \sin(\gamma 1) & l_1 \cos(\gamma_1 - \alpha_1) \\ \cos(\gamma_2) & \sin(\gamma 2) & l_2 \cos(\gamma_2 - \alpha_2) \\ \cos(\gamma_3) & \sin(\gamma 3) & l_3 \cos(\gamma_3 - \alpha_3) \\ \cos(\gamma_4) & \sin(\gamma 4) & l_4 \cos(\gamma_4 - \alpha_4) \end{bmatrix} \tag{G.4}$$

The second matrix expressing the no slip-constraint for the four wheels. 4×4:

$$J_2(\gamma) = \begin{bmatrix} R_w & 0 & 0 & 0 \\ 0 & R_w & 0 & 0 \\ 0 & 0 & R_w & 0 \\ 0 & 0 & 0 & R_w \end{bmatrix} \tag{G.5}$$

R_w is radius of a wheel. The inverse of equation G.5 is:

$$J_2^{-1}(\gamma) = \begin{bmatrix} \frac{1}{R_w} & 0 & 0 & 0 \\ 0 & \frac{1}{R_w} & 0 & 0 \\ 0 & 0 & \frac{1}{R_w} & 0 \\ 0 & 0 & 0 & \frac{1}{R_w} \end{bmatrix} \tag{G.6}$$

$C_1(\gamma)$ expresses the no lateral movement-constraint for all four wheels. 4×3:

$$C_1(\gamma) = \begin{bmatrix} \sin(\gamma_1) & -\cos(\gamma 1) & -l_1\cos(\alpha_1 - \gamma_1) \\ \sin(\gamma_2) & -\cos(\gamma 2) & -l_2\cos(\alpha_2 - \gamma_2) \\ \sin(\gamma_3) & -\cos(\gamma 3) & -l_3\cos(\alpha_3 - \gamma_3) \\ \sin(\gamma_4) & -\cos(\gamma 4) & -l_4\cos(\alpha_4 - \gamma_4) \end{bmatrix} \tag{G.7}$$

The contraint matrix Λ:

$$\Lambda = \begin{bmatrix} J_1(\gamma)R(\theta) & 0 & -J_2 \\ C_1(\gamma)R(\theta) & 0 & 0 \end{bmatrix} \tag{G.8}$$

such that:

$$\Lambda\dot{q} = \Lambda \begin{bmatrix} \dot{\xi} \\ \dot{\gamma} \\ \dot{\psi} \end{bmatrix} = 0 \tag{G.9}$$

$\Sigma(\gamma)$ is the null-space of $C_1(\gamma)$:

$$\Sigma(\gamma) = \begin{bmatrix} l_j cos(\gamma_i)cos(\alpha_j - \gamma_j) - l_i cos(\gamma_j)cos(-\alpha_i + \gamma_i) \\ l_i sin(\gamma_j)cos(-\alpha_i + \gamma_i) - l_j sin(\gamma_i)cos(\alpha_j - \gamma_j) \\ cos(\gamma_i)sin(\gamma_j) - sin(\gamma_i)cos(\gamma_j) \end{bmatrix} \tag{G.10}$$

The derivative of equation G.10:

$$\dot{\Sigma}(\gamma) = \begin{bmatrix} \dot{\gamma}_1 l sin(\alpha_1 - \gamma_1)cos(\gamma_2) - \dot{\gamma}_2 l cos(\alpha_1 - \gamma_1)sin(\gamma_2) + \\ \dot{\gamma}_1 l sin(\gamma_1)cos(-\alpha_2 + \gamma_2) + \dot{\gamma}_2 l cos(\gamma_1)sin(-\alpha_2 + \gamma_2) \\ \\ \dot{\gamma}_2 l cos(\gamma_2)cos(\alpha_1 - \gamma_1) + \dot{\gamma}_1 l sin(\gamma_2)sin(\alpha_1 - \gamma_1) + \\ \dot{\gamma}_2 l sin(\gamma_2 - \alpha_2)sin(\gamma_1) - \dot{\gamma}_1 l cos(\gamma_2 - \alpha_2)cos(\gamma_1) \\ \\ \dot{\gamma}_1 cos(\gamma_1 - \gamma_2) - \dot{\gamma}_2 cos(\gamma_1 - \gamma_2) \end{bmatrix} \tag{G.11}$$

For simulation purposes two versions of $\Sigma(\gamma)$ exists:

$$\Sigma_L(\gamma) = \begin{bmatrix} l\cos(\gamma_2)\cos(\alpha_1 - \gamma_1) - l\cos(\gamma_1)\cos(\gamma_2 - \alpha_2) \\ l\sin(\gamma_2)\cos(\alpha_1 - \gamma_2) - l\sin(\gamma_1)\cos(\gamma_2 - \alpha_2) \\ \sin(\gamma_1 - \gamma_2) \end{bmatrix} \tag{G.12}$$

$$\Sigma_R(\gamma) = \begin{bmatrix} l\cos(\gamma_4)\cos(\alpha_3-\gamma_3) - l\cos(\gamma_3)\cos(\gamma_4-\alpha_4) \\ l\sin(\gamma_4)\cos(\alpha_3-\gamma_3) - l\sin(\gamma_3)\cos(\gamma_4-\alpha_4) \\ \sin(\gamma_3-\gamma_4) \end{bmatrix} \tag{G.13}$$

The kinematic model matrix:

$$S(q) = \begin{bmatrix} R^T(\theta)\Sigma(\gamma) & 0 \\ 0 & I \\ J_2^{-1}J_1(\gamma)\Sigma(\gamma) & 0 \end{bmatrix} \tag{G.14}$$

G.3 Dynamical Model

Matrices used to calculate the kinetic energy of the robot:

$$M = \begin{bmatrix} M_1 & 0 & M_2 \\ 0 & M_1 & M_3 \\ 0 & 0 & M_4 \end{bmatrix} \tag{G.15}$$

$$P = \begin{bmatrix} M_1 & 0 & \frac{1}{2}(M_2\cos(\theta) - M_3\sin(\theta)) \\ 0 & M_1 & \frac{1}{2}(M_2\sin(\theta) + M_3\cos(\theta)) \\ \frac{1}{2}(M_2\cos(\theta) - M_3\sin(\theta)) & \frac{1}{2}(M_2\sin(\theta) + M_3\cos(\theta)) & M_4 \end{bmatrix} \tag{G.16}$$

$$\dot{P} = \dot{\theta}\begin{bmatrix} 0 & 0 & \frac{1}{2}(-M_2\sin(\theta) - M_3\cos(\theta)) \\ 0 & 0 & \frac{1}{2}(M_2\cos(\theta) - M_3\sin(\theta)) \\ \frac{1}{2}(-M_2\sin(\theta) - M_3\cos(\theta)) & \frac{1}{2}(M_2\cos(\theta) - M_3\sin(\theta)) & 0 \end{bmatrix} \tag{G.17}$$

$$K_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{G.18}$$

$$K_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{G.19}$$

$$N = \begin{bmatrix} 0 & 0 & -\frac{1}{2}\theta M_3 \\ 0 & 0 & \frac{1}{2}\theta M_2 \\ -\frac{1}{2}\theta M_3 & \frac{1}{2}\theta M_2 & O \end{bmatrix} \tag{G.20}$$

Matrix equal to the transpose of the constraint matrix. Is used in the Lagrange equation:

$$A(q) = \begin{bmatrix} R^T(\theta)J_1^T(\gamma) & R^T(\theta)C_1^T(\gamma) \\ 0 & 0 \\ -J_2^T & 0 \end{bmatrix} \tag{G.21}$$

Vector of forces:

$$\tau = \begin{bmatrix} \tau_\psi \\ \tau_\gamma \end{bmatrix} = \begin{bmatrix} \tau_{\psi 1} \\ \tau_{\psi 2} \\ \tau_{\psi 3} \\ \tau_{\psi 4} \\ \tau_{\gamma 1} \\ \tau_{\gamma 2} \\ \tau_{\gamma 3} \\ \tau_{\gamma 4} \end{bmatrix} \tag{G.22}$$

Vector which transformes the input τ into forces performing work on the generalized coordinates:

$$Q(q) = \begin{bmatrix} 0_{3\times 8} \\ I_{8\times 8} \end{bmatrix} \tag{G.23}$$

Inertia matrix. Used when writing the kinetic energy of the robot in a compact form:

$$D(q) = \begin{bmatrix} & & & 0 & 0 & 0 & 0 & & & & \\ & [R^T(\theta)MR(\theta)] & & 0 & 0 & 0 & 0 & & 0_{3\times 4} & & \\ & & & \frac{1}{2}I_W & \frac{1}{2}I_W & \frac{1}{2}I_W & \frac{1}{2}I_W & & & & \\ 0 & 0 & \frac{1}{2}I_W & \frac{1}{2}I_W & 0 & 0 & 0 & & & & \\ 0 & 0 & \frac{1}{2}I_W & 0 & \frac{1}{2}I_W & 0 & 0 & & 0_{4\times 4} & & \\ 0 & 0 & \frac{1}{2}I_W & 0 & 0 & \frac{1}{2}I_W & 0 & & & & \\ 0 & 0 & \frac{1}{2}I_W & 0 & 0 & 0 & \frac{1}{2}I_W & \frac{1}{2}I_W & 0 & 0 & 0 \\ & & & & & & & 0 & \frac{1}{2}I_W & 0 & 0 \\ & 0_{4\times 3} & & & 0_{4\times 4} & & & 0 & 0 & \frac{1}{2}I_W & 0 \\ & & & & & & & 0 & 0 & 0 & \frac{1}{2}I_W \end{bmatrix} \tag{G.24}$$

Matrices and vectors for the resulting dynamical model:

$$F(\gamma) = \begin{bmatrix} \Sigma^T(\gamma)E^T(\gamma) & 0 \\ 0 & I_4 \end{bmatrix} \tag{G.25}$$

$$f(\gamma, u) = \begin{bmatrix} f_1(\gamma, u) \\ f_2(\gamma, u) \end{bmatrix} \tag{G.26}$$

$$\begin{aligned} f_1(\gamma, u) = {} & \Sigma^T(\gamma)\Big[\big[R(\theta)\dot{P}R^T(\theta) + R(\theta)P\dot{R}^T(\theta) + \\ & -R(\theta)K_1\eta\Sigma^T(\gamma)N + \frac{1}{2}E^T(\gamma)I_{WM}E(\gamma)\big]\Sigma(\gamma) + \\ & \big[R(\theta)PR^T(\theta) + \Sigma^T(\gamma)R(\theta)\frac{1}{2}E^T(\gamma)I_{WM}E(\gamma)\big]\dot{\Sigma}(\gamma)\Big]\eta + \frac{1}{2}K_1 I_{WV}\zeta \end{aligned} \tag{G.27}$$

$$f_2(\gamma, u) = \frac{1}{2}\ddot{\theta} I_W K_2 \tag{G.28}$$

$$H(\gamma) = \begin{bmatrix} H_1(\gamma) & 0 \\ 0 & \frac{1}{2} I_{WM} \end{bmatrix} \tag{G.29}$$

$$H_1(\gamma) = \Sigma(\gamma)[R(\theta) P R^T(\theta) + \frac{1}{2} E^T(\gamma) I_{WM} E(\gamma)]\Sigma(\gamma) \tag{G.30}$$

G.4 Control by Feedback Linearization

A_s and B_s are matrices which describes the linear system after feedback linearization:

$$A_s = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \quad \text{and} \quad B_s = \begin{bmatrix} 0_{3\times3} \\ I_{3\times3} \end{bmatrix} \tag{G.31}$$

Vector in the control law for the feedback linearization:

$$\alpha(z) = -\beta^{-1}(z)\dot{R}^T(\theta)\Sigma(\gamma)\eta \tag{G.32}$$

Matrix in the control law for the feedback linearization:

$$\beta(z) = R^T(\theta)\beta_1(z) \tag{G.33}$$

$$\beta_1(z) = \begin{bmatrix} l_1 cos(\gamma_2) cos(\alpha_1 - \gamma_1) & \big(l_1 cos(\gamma_2) sin(\alpha_1 - \gamma_1) & \big(- l_1 sin(\gamma_2) cos(\alpha_1 - \gamma_1) \\ -l_2 cos(\gamma_1) cos(-\alpha_2 + \gamma_2) & l_2 sin(\gamma_1) cos(-\alpha_2 + \gamma_2)\big)\eta & l_2 cos(\gamma_1) sin(-\alpha_2 + \gamma_2)\big)\eta \\ \\ l_1 sin(\gamma_2) cos(\alpha_1 - \gamma_1) & \big(l_1 sin(\gamma_2) sin(\alpha_1 - \gamma_1) & \big(l_1 cos(\gamma_2) cos(\alpha_1 - \gamma_1) \\ -l_2 sin(\gamma_1) cos(-\alpha_2 + \gamma_2) & -l_2 cos(\gamma_1) cos(-\alpha_2 + \gamma_2)\big)\eta & l_2 sin(\gamma_1) sin(-\alpha_2 + \gamma_2)\big)\eta \\ \\ sin(\gamma_1 - \gamma_2) & cos(\gamma_1 - \gamma_2)\eta & -cos(\gamma_1 - \gamma_2)\eta \end{bmatrix} \tag{G.34}$$

Matrix of gain constants, used in the linear control law:

$$K_c = \begin{bmatrix} 0.05 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0.05 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0.05 & 0 & 0 & 0.001 \end{bmatrix} \tag{G.35}$$

A threshold vector which determines the state of the hybrid controller:

$$\Delta = \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_\theta \end{bmatrix} \tag{G.36}$$

G.5 Passivity-Based Control

Vector used to model the dynamics of the model:

$$m(q, u) = S^T(q) D(q) \dot{S}(q) u + S^T(q) n(q, S(q)u) \tag{G.37}$$

Matrix related to the inertia matrix $D(q)$:

$$D^*(q) = S^T(q)D(q)S(q) \tag{G.38}$$

Storage function used in the control design:

$$V = \frac{1}{2} e_{Kin}^T K_{pKin} e_{Kin} \tag{G.39}$$

Gain matrices used in the control law for the dynamics:

$$K_p = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0.01 \end{bmatrix} \tag{G.40}$$

$$K_d = 0.003 \cdot I_{5\times 5} \tag{G.41}$$

Gain matrices used in the control law for the kinematics:

$$K_{pKin} = \begin{bmatrix} 0 & 0 & 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.4 \\ 0.4 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0.4 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0.4 & 0 & 0 & 0.001 \end{bmatrix} \tag{G.42}$$

$$K_{dKin} = \begin{bmatrix} 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 \\ 0.01 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0.01 & 0 & 0 & 0.001 \end{bmatrix} \tag{G.43}$$

APPENDIX H

Programming the PIC

This appendix explains how the PIC16F877 is programmed. First an introduction to MPLAB is given, followed by information about the PIC C-compiler CC5X. The last section of this appendix describes the bootloader, which was used to download the usercode in the microprocessor.

H.1 MPLAB

The development tool MPLAB v. 5.7 was used to program the PIC. This is a free tool from Microchip and can be found in [CD, \PIC util\MPLAB\]. In MPLAB it is possible to test the code before implementing it in the microprocessor. The generated code can be simulated continuously or line by line, making debugging easier. Furthermore MPLAB is able to show the time used to execute the code, and the status of the different special function registers, memory banks etc.

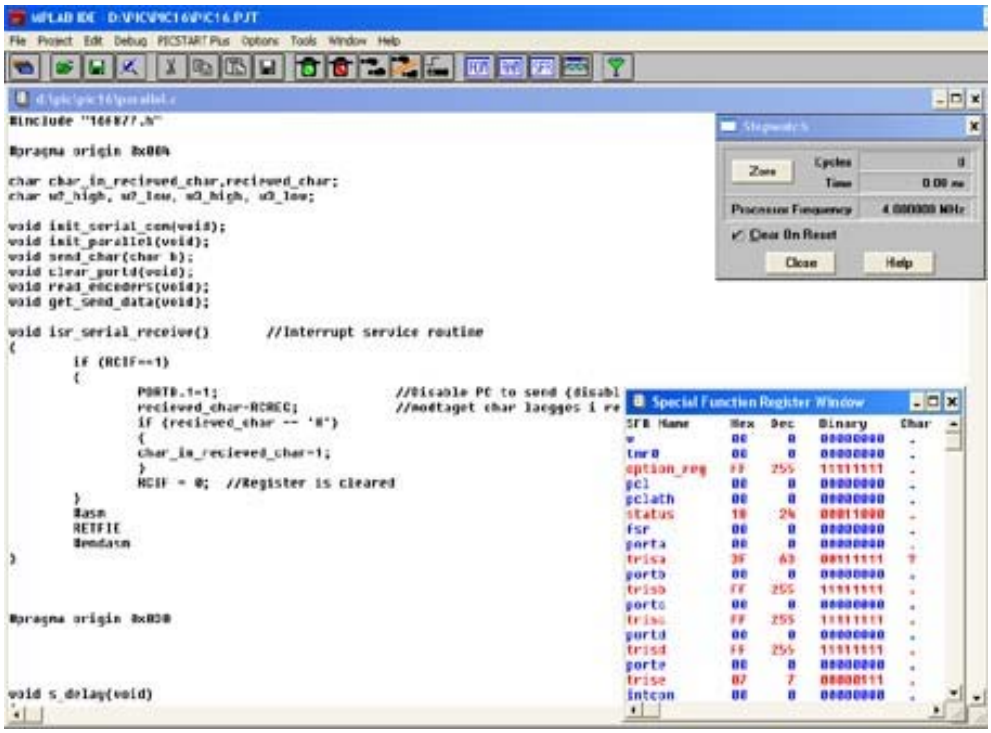

Figure H.1: *The development window of MPLAB. In the right side the stopwatch and special function register windows.*

H.2 CC5X

In this project MPLAB was used in cooperation with the C-compiler CC5X. CC5X only supports MPLAB v. 5.x and is a free PIC C-compiler. The compiler as well as a manual can be found on [CD, \PIC util\CC5X\]. Using this compiler has been an effective and fast method of programming the microprocessor.

In the following a description is given of how to use CC5X with MPLAB.

H.2.1 Installation and general use

1. First install CC5X on your computer as described initially.

2. Then copy the files *.MTC and *.INI from the CC5X folder (directory) to the MPLAB folder (this folder contains other .ini and .mtc files).

3. Next time MPLAB is started, select the Project\ Install Language Tool menu item. Select CC5X from the Language Suite. Then the Tool Name (C-Compiler). Then the right Executable (c:\cc5x\cc5x.exe or similar). Also mark the Command-line box. Then click OK. CC5X will now be one of the selectable tools in MPLAB.

The following is a brief description on how to use CC5X in a new project under MPLAB.

1. Start MPLAB and create a new project (Project\New Project). Chose a project name (*.prj) and a directory where to locate this file and the other project files (C,H,HEX,ASM). Type <Enter> or the OK button.

2. Edit Project is the next window. MPLAB suggests a Target Filename based on the project name. This is automatically changed during step 4. Include Path do not have to be specified (and should be left open initially). Library Path and Linker Script Path are not used anyway. Use Development Mode to select the processor and simulator\debugger (ignore any MPLAB warning at the current stage). Change Language Tool Suite to CC5X (this is one of the menu items if the installation steps was completed).

3. Double-click on the (target) name in the Project File box. A window named Node Properties pops up. The typical selections are already marked. Options are disabled or enabled by clicking on the box (second column) after the option name. Note that the right Include Path (c:\cc5x) is required to make CC5X find the header files. Click the OK button. NOTE: It is not enough to select the right processor in MPLAB alone. CC5X also need to know the processor type, either through a command line option (-p16C74) or by a pragma statement in the beginning of the program (#pragma chip PIC16C74).

4. Click on the Add Node button. Type the name of your main C file or chose an existing C file (sample1.c). It is recommended to try one of the supplied example files initially. If the (sample) C file does not reside in the selected project directory, copy it to this directory first. Note that files included in the main C file (and nested include) can not be listed in the Project Files box. Click the OK button.

5. Open the main C file. Compile the file using Project\Make Project (F10). Project\Build Node (Alt-F10) requires that the main C file is in the current active window. Double-click on the error messages (if any) and correct the C code. Repeat the compilation until there are no error messages. Use Open file to inspect the generated files. The *.occ file contains compiler output information. IMPORTANT: If you selected the Error File command line option, then MPLAB will suppress the output from the compiler and display the content of the *.err file only. Change this option to the desired setting. It may be necessary to change some of the command line options (Processor, Hex Format) if MPLAB pops up a warning window.

H.3 Bootloader

This section gives a description of the bootloader and the procedures for downloading code to the microprocessor. The bootloader is a piece of code that makes it possible to download new usercode to the microcontroller from a terminal program through a serial connection. From now on, when usercode is mentioned, this is the code written by the project group. Microchip, the producer of the PIC16F877, has developed a bootloader. This bootloader was used in the project and the source code can be found in [CD, \sourcecode\PIC16F877\Bootloader\]. An application note from Microschip concerning the bootloader can be found in [CD, \datasheets\00732a.pdf].

Initially the bootloader, BOOT877.hex, is written into the microprocessor in the more common way, using a EEPROM writer. The procedure using the EEPROM writer is only done once and hereafter it is possible to download usercode from the terminal program. The terminal program used on the PC is Tera Term Pro. Tera Term Pro can be found in [CD, \PIC util\Tera Term Pro\]. Figure H.2 shows the wiring and the hardware parts needed in implementing a bootloader. The procedure for using the bootloader is as follows:

- On the PC, set up the serial port baud rate and flow control (hardware handshaking).
- Connect the serial port of the PIC16F877 device to the serial port of the PC.
- Press the red switch (SW1 in figure H.2) to pull pin RB0 low.
- Power up the board to execute the boot code by pressing the black button.
- From the PC, send the hex-file to the serial port.
- A period ”.” will be received from the serial port for each line of the hex-file that is sent.
- An ”S” or ”F” will be received to indicate success or failure.
- The user must handle a failure by resetting the board and starting over.
- Release the red switch to set pin RB0 high.
- Power-down the board and power it up again to start executing the usercode.

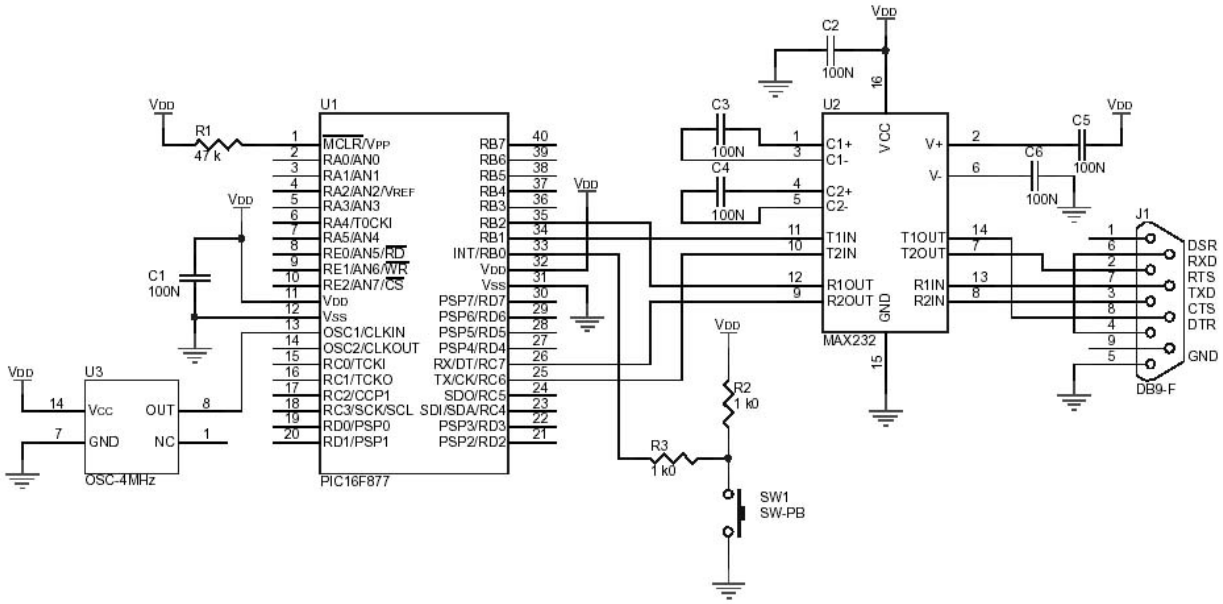

Figure H.2: *The parts and wiring needed when using the bootloader*

Bibliography

[Champion, 1996] Georges Bastin; Brigitte D'Andréa-Novel; Guy Champion. *Structual Properties and Classification of Kinematic and Dynamic Models of Wheeled Robots*. *IEEE Transactions on Robotics and Automation*, February 1996.

[Dimon, 2002] Palle Andersen; Tom Søndergård Pedersen; Jan Dimon. *Robust Feedback Linearization-based Control Design for a Wheeled Mobile Robot*. *Proc. of the 6th International Symposium on Advanced Vehicle Control*, 2002.

[Divakaran, 2002] Dinil Divakaran. *RTLINUX HOWTO*. http://www.tldp.org/HOWTO/RTLinux-HOWTO/, 2002.

[Emami-Naeini, 1994] Gene F. Franklin; J. David Powell; Abbas Emami-Naeini. *Feedback Control Of Dynamic Systems, 3rd. Ed.* Addison-Wesley, 1994.

[Khalil, 2002] Hassan K. Khalil. *Nonlinear Systems*. Prentice-Hall, 2002.

[Lay, 1997] David C. Lay. *Linear Algebra And Its Applications, 2nd Ed.* Addison-Wesley, 1997.

[Lima, 1999] Luis Custódio; Pedro Aparício; Rodrigo Ventura; Pedro Lima. *A Functional Architecture for a Team of Fully Autonomous Cooperative Robots*. *RoboCup-99: Robot Soccer World Cup III*, 1999.

[Micaelli, 1996] Benoit Thuilot; Brigitte D'Andréa-Novel; Alain Micaelli. *Modeling and Feedback Control of Mobile Robots Equipped with Several Steering Wheels*. *IEEE Transactions on Robotics and Automation*, June 1996.

[Nijmeijer, 1998] Antonio Loría; Henk Nijmeijer. *Passivity Based Control*. http://www-lag.ensieg.inpg.fr/ aloria/aloria-publis/Papers/pbc.pdf, 1998.

[Oriolo, 1995] Alessandro De Luca; Guiseppe Oriolo. *Kinematics and Dynamics of Multi-Body Systems, Chap. 7: Modeling and Control of Nonholonomic Mechanical Systems*. Springer-Verlag, Wien, 1995.

[Sagatun, 1991] Thor I. Fossen; Svein I. Sagatun. *Adaptive Control of Nonlinear Underwater Robotic Systems*. *Journal of Robotic Systems, JRS-8(3):393-412*, 1991.

[Schiøler *et al.*, 2001] H. Schiøler, L.T. Son, O.B. Madsen, and J.D. Nielsen. Communicating cooperative robots with bluetooth. Technical report, Department of Control Engineering, July 2001.

[Sørensen, 2001] Mathias Jesper Sørensen. *Modeling and Control of Wheeled Farming Robot*. Master's thesis, Aalborg University, 2001.

[Tamiya, 2003] USA Tamiya. *Homepage of Tamiya TXT-1 Xtreme Truck*. http://www.tamiyausa.com/product/rc/electric/110scaleoffroadtrucks/58280.html, February 2003.

[Tarozzi, 2002] Paolo Tarozzi. *A.P.I. Advanced Model and Control*. Master's thesis, Aalborg University, 2002.

[Taylor, 2002] C. J. Taylor. *A Framework and Architecture for Multirobot Coordination*. *International Journal of Robotics Research*, July 2002.

[van der Schaft, 1999] Hans Schumacher; Arjan van der Schaft. *An Introduction to Hybrid Dynamical Systems*. Springer, 1999.

[Wang, 1988] C. Ming Wang. *Location Estimation And Uncertainty Analysis For Mobile Robots*. *IEEE International Conference on Robotics and Automation*, pages 1231–1235, 1988.