Diploma Thesis IST-

Bluetooth-networked trajectory control of an autonomous LEGO-Mindstorms vehicle

A low-cost laboratory experiment

cand. kyb. Martin Weiss

Supervisor: Dr. Ángel Valera Fernandez (DISA)

Prof. Dr.-Ing. Herbert Wehlan (ISR)

University of Stuttgart

Institute for System Dynamics and Control Engineering (ISR)

Prof. Dr.–Ing. Dr. h.c. mult. E.D. Gilles

Universidad Politécnica de Valencia

Department of Systems Engineering and Control (DISA)

Prof. Dr.–Ing. XXX

31. May 2006

Preamble

This thesis was written at the Department of Systems Engineering and Control (DISA) at the Technical University of Valencia (Universidad Politécnica de Valencia).

Hereby I want to thank Ángel Valera Fernandez for making it possible to write this thesis under his supervision and the help and hospitality I received from him during my time in Valencia.

Furthermore I want to thank Alberto Encimas for his dedicated help corresponding technical problems with the Bluetooth module.

To my future wife, who agreed to marry me during my time in Valencia.

Solemn Affirmation

Hereby I, Martin Weiss, declare on oath that I made this thesis self-dependent and only under use of the mentioned appliances. Stuttgart, 15th of Mai 2006

Contents

Chapter 1

Introduction and problem formulation

Nowadays increasing the efficiency of technical processes is center of investigation in many fields of engineering as is this way the use of raw materials and costs can be dropped. Here control engineering play an important role to achieve these objectives by stabilizing, speeding up the desired processes and making them more precise.

Also the Instituto de Automática e Informática Industrial (AI2) (Institute of Automatization and industrial computer sciences), where the thesis was made and written, is mainly working in the field of control engineering, mainly with industrial robotics, autonomous vehicles and image processing. It is part of the Department of Systems Engineering and Control (DISA) at the Technical University of Valencia (Universidad Politécnica de Valencia).

As it is also in charge of several teaching actions, there is a constant need of laboratory experiments that can be used for practicals. The following thesis will describe the development and theory of an experiment for trajectory control of a small LEGO vehicle. It will also show that even a cheap solution, as the LEGO®-Mindstorms™invention system, can be used to teach more complex control engineering tasks, where especially the system uncertainties require robust and fault-tolerant control solutions.

1.1 LEGO-Mindstorms in control engineering education

By performing a small search in the internet, it soon becomes obvious that LEGO-Mindstorms is widely used in education, mostly in schools and undergraduate courses. However they are mostly applied to give the students a basic idea of what sensor-feedback means, and how it can be used to perform simple control tasks, as line-tracking and autonomous steering by collision detection.[25],[26] The course of studies Cybernetic Engineering of the University of Stuttgart, for example, organizes every year a competition between groups of first-year students and pupils from local schools. The tasks for the LEGO®-Robots change every year, so the groups have to come up with new ideas and construction solutions every time. But there can also be found several attempts to use LEGO-Mindstorms for more theory based and serious control tasks.

Figure 1.1: LEGO Pendulum controller; Source [32]

In an experiment is described, where the position of a pendulum attached on a non-steerable LEGO-vehicle (fig. 1.1) is stabilized and controlled by a linear controller. The motor dynamics are linearized by adding a second short-circuited motor as brake in addition to a self-made high-precision angular sensor for measurement of the pendulum amplitude.

Furthermore in [11] a tracked vehicle similar to the one used in this thesis and a simple mathematical simulation model were developed to provide a Simulink framework for controller-design. This was also meant be used in practicals to develop trajectory- or point-to-point controller for the robot model.

In engineering education, especially in control engineering, practicals are of high importance to understand the complex coherences between system-structure, controller types and technical realization problems as filtering, discretization and delays. There are a lot of companies selling complete laboratory experiments of all thinkable types and many universities build them themselves. However this experiments mostly cost several thousand

Euros, what means that only few can be bought. As a consequence the student groups grow very big.
Here LEGO-Mindstorms forms a cheap alternative with approximately 200 Euros for one invention set. It is also very flexible with more than 700 different parts and offers a great variety of possible experiments. It includes a microprocessor, called RCX, with 3 analog inputs and outputs, which can be easily programmed by the delivered graphical software ROBOLAB or one of the many cross-compilers which allow to write programs in common languages like C/C++, BASIC and JAVA. Furthermore it provides motors (actuators) and different types of sensors: light sensors, contact sensors, rotation sensors and temperature sensors.
The drawback is that almost no technical data is available for mathematic modelling and has to be obtained by measurements. Furthermore there are extensive inexactness in the mechanical parts which lead to notable non-linear friction phenomenons, which have to be handled. Also the sensors, especially the rotations sensors, are very imprecise. This will be discussed more closely in chapters 2 and 4, which treat the technical details of the sensors and the development of a PID velocity controller for a LEGO®vehicle.
The objective of this thesis is to develop such a laboratory experiment.

1.2 Objectives of the thesis

The most common application for LEGO-Mindstorms are small vehicles to perform different tasks. A possible way to use such a vehicle for educational purposes in control engineering is to combine it with position detection which allows better control actions like position control, path tracking or trajectory control.
To provide such a experiment, especially for practicals with several groups at the same time, a small framework for communication and controller design has to be developed, building on the following previous works done at the AI2.

Previous works and starting position

Alberto Encinas, also student at the UPV, constructed a Bluetooth-to-infrared adaptor, based on the Initium Promi-ESD Bluetooth chip.[31] Using the S̈erial Port Profileït can easily be accessed via an emulated serial port. The serial data then is translated into a infrared signal for the RCX using a infrared transmitter and receiver. With this device the RCX becomes independent from the LEGO USB tower and can operate with bigger range and without the need for visual contact. It will be discussed more detailed in Chapter 2.
Furthermore Carlos Garcia implemented for his final project a simple C-based communication with a Visual Basic frontend to send commands and data to the RCX.[36] He realized a simple PC side feedfoward control by sending commands for moving and turning to the vehicle, but without knowledge about the result of this action. On the vehicle the command was realized by moving until encoders on the axes reach a certain value, proportional to the way to drive or the angle to rotate. This only encoder-based control is also very imprecise because of the integral error of encoders. Especially in respect of rotation there are large uncertainties caused by friction and slip.
A different approach and task was realized in the 'LEGO Maze Pathfinder' project citeLego-Maze, where a vision-based path-finding algorithm for a LEGO-Vehicle was developed. This was also realized by encoder-based point-to-point movements, supplemented by position-feedback and path recognition with a webcam and image-processing. Image-processing and communication (here with the USB tower) were implemented in JAVA, using the open-source JAVA-firmware leJOS. JAVA turns out to be a good solution, especially for the communication, but problematic for practicals because of the diverse fundamentals and knowledge of this programming language among students.

previous works and starting position

As a better solution the complete experimental setup has to be developed for use out of MATLAB/Simulink as this is the standard tool for control development. So the student groups only have to work with MATLAB/Simulink and don't need any further knowledge

of other programming languages and the communications setup.
Therefor the following requirements build the objectives of the thesis, complemented by a example vehicle and controller setup.

- **Bluetooth communications with the RCX**

  A possible way has to be investigated, to send and receive data from the RCX via the Bluetooth interface out of MATLAB, with a sample time of approximately 100ms. If possible, the used protocol should be robust to transmission errors and assure the reception of transmitted data.

  In addition to this, it is aimed at connecting with several robots at a time for further, possibly interacting experiments.[Chapter 2,3]

- **Vehicle with velocity controller**

  For demonstration purposes a example vehicle should be created. As there are several possibilities, it was decided to construct a tracked vehicle, since this means additional requirements to the controller because of the slip of the chains.

  A velocity-controller is to be implemented on the vehicle to control the two tracks independently.[Chapter 2,4]

- **Image processing**

  An obligatory demand for trajectory control is the measurement or estimation of the vehicles position.

  A standard webcam shall be used for this purpose, detecting the coordinates from a top-view position.

  Therefor the vehicle has to be equipped with an adequate marker to guarantee robust and precise tracking within the bounds of accuracy needed for such an experiment.

  Moreover example algorithms for image-filtering, segmentation and object-detection have to be implemented for the controller demonstration.

  In addition a possibility for calibrating the camera is required to obtain the real-world coordinates of the vehicle from the image data.

As a possible supplement, also the detection of (possibly marked) obstacles should be available to offer the possibility of a path-searching trajectory control.[Chapter 3]

- **Applications**

  For demonstration and a possible example practical, two or three different trajectory controllers shall be realized.

  Therefore firstly all parts of the communication system and the robot's velocity controller have to be completed to close the control loop.[Chapter 5,7,8]

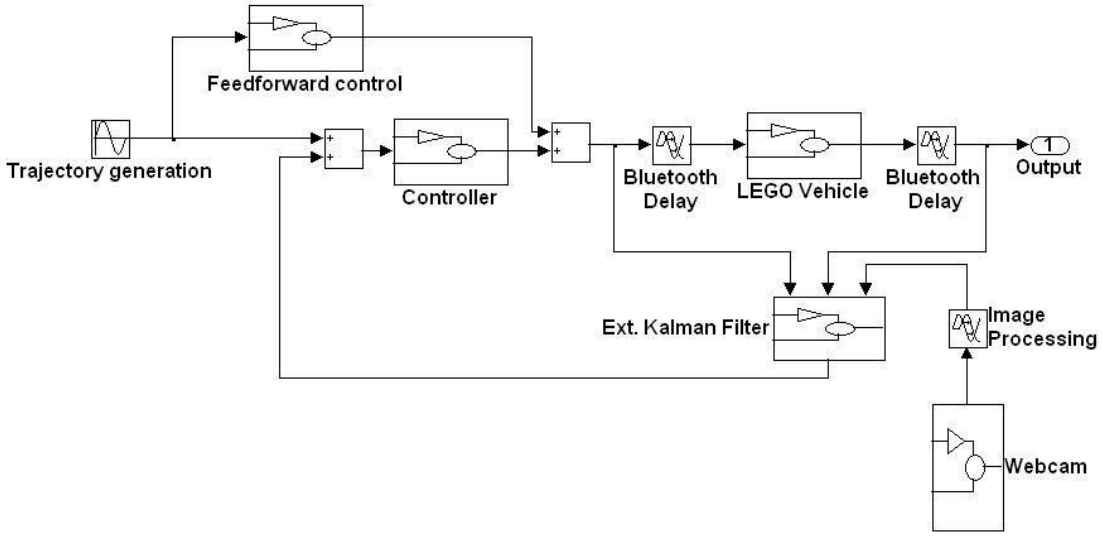


Figure 1.2: Schematic buildup of the final control system

Furthermore for providing example curves, algorithms to generate time-dependent trajectories have to be researched and implemented. Here a possible extension could be a path-search algorithm to generate a trajectory by avoiding obstacles.[Chapter 6]

Chapter 2

Basics and problem-setup

Before discussing details of experiment and controller design, the used technologies and tools the work is based on will be introduced.
One task of the project was to use only LEGO parts for the Robot, despite the Bluetooth-infrared-device and the markings. Later we will see that this demand couldn't be held up and an additional part had to be inserted to allow sufficiently short sample times.
However only LEGO sensors and actuators were used XXX whose technical details will be discussed in the following.
Furthermore the used hardware and software will be presented, as well as the LEGO-vehicle used for the experiments.

2.1 The LEGO-Mindstorms-System

LEGO®-Mindstorms™were first introduced in 1998 with the Robotics Invention Set 1.0 (RIS), which was developed for children of 12 and older. Additional to the common LEGO pieces this set included a programmable brick called RCX (Robotics Command Explorer), 4 different sensors and motors.
Soon after launch enthusiasts began to extend the hard- and software and Mindstorms turned out to be not only interesting for children, but also for professionals (as hobby) and

education.
Today various self-made sensors can be found in internet (e.g. at [13],[8] and [30]) which allow far more applications.
To programm the RCX the RIS is delivered with a graphical software that should be intuitive especially for children. However many users, as experienced programmers for instance, felt the drag-and-drop environment too limiting.
As LEGO didn't publish technical information about the RCX and its firmware, a group of enthusiasts began to work out all details on their own by reverse engineering. They created the following alternative programming languages and firmwares for the RCX,

- NQC (Not-quite-C) was developed by Dave Baum in 1999. It has a C-like syntax and works on the standard LEGO firmware, but still is very limited.

- LegOS by Markus L. Noga is the first alternative firmware and realtime system. RCX programs are compiled by a real C-compiler (now BrickOS).

- Ralph Hempel wrote a Forth interpreter called pbForth

- leJOS, a firmware and software environment that allows programming the RCX in JAVA was first published in April 2002 by a group of open-source programmers

In addition LEGO began to enhance the hardware and to release further sets as the RIS 1.5 (1999) and 2.0 (2000), Robotics Discovery System (RDS, 1999) and Droid Developer Kit (DDK, 1999). RDS and DDK use simpler versions of programmable bricks with less or internal sensors and motors.[12],[30]
The next step will be the NXT which is planned for summer of 2006 with extended communications as USB and Bluetooth, 4 Sensors and enhanced sensors and motors.
In the following the most important parts of these Mindstorms sets will be presented.

2.1.1 RCX - The Robotics Command Explorer

The RCX, as core of the Mindstorms system, was initially developed as educational tool in collaboration with Massachusetts Institute of Technology (MIT) [30]. The first version allowed six inputs and outputs which were reduced to three in the commercial version for reasons of energy saving.

Microcontroller

The central part of the RCX is a Hitachi H8/300 high-speed processor which is ideally suited for realtime control applications. The on-chip supporting modules implement peripheral functions needed in system configurations, as ROM, RAM, eight 16-bit registers, three types of timers (a 16-bit free-running timer, 8-bit timers, and a watchdog timer), a serial communication interface (SCI), an A/D converter, and I/O ports.[7]

Memory

The CPU is equipped with a 16kB ROM containing low-level software that guarantees basic functionality as data-upload via infrared after a reset. Additionally the RCX is provided with 32KByte of external RAM to store LEGO or alternative firmware as well as user programs and data.

Software architecture

All different levels of software on the RCX lead to the architecture showed in 2.1, with LegOS as example for alternativ firmware .

The different firmware types mainly use the basic system on the ROM to control the hardware. For generating programs for the RCX on a PC, an individual cross-compiler has to be used for each firmware. It generates byte code which then can be transferred to the RCX.

IR

For communication with the PC or other devices, an infrared device is connected to the

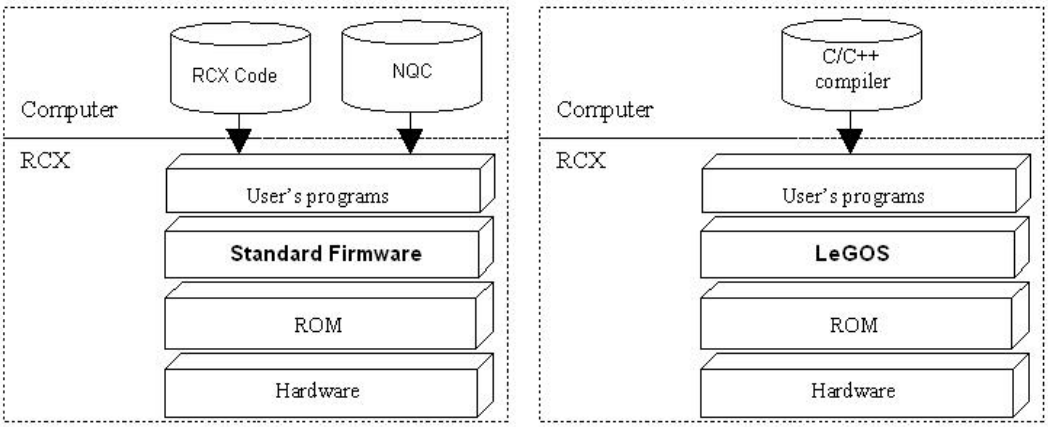


Figure 2.1: Software architecture for programming the RCX. A basic firmware in the ROM provides for hardware control. Using this interface a more complex firmware can be uploaded and run user programs.

serial port of the CPU. Using the IrDA (Infrared Data Association) standard with a carrier frequency of 38 kHz a serial connection with 2400bps (4800bps for RCX 2.0) can be established with any IrDA device.

Inputs and outputs

For control applications the RCX provides three analog inputs and three motor outputs. The outputs support 3 modes, 'on', 'off' and 'floating'. In floating-mode the motor isn't powered but free-running whereas in 'off'-mode the motor is stopped by using energy.
The inputs are connected with the 10-bit A/D converters of the microprocessor and operate in the range of 0V to 7V.

Other features

Additional to this basic functions the RCX provides some more features as buttons for interactions and handling, and an internal beeper for sound applications.
A very useful feature is provided by the liquid crystal display (LCD), which can be used for displaying important information and program debugging.

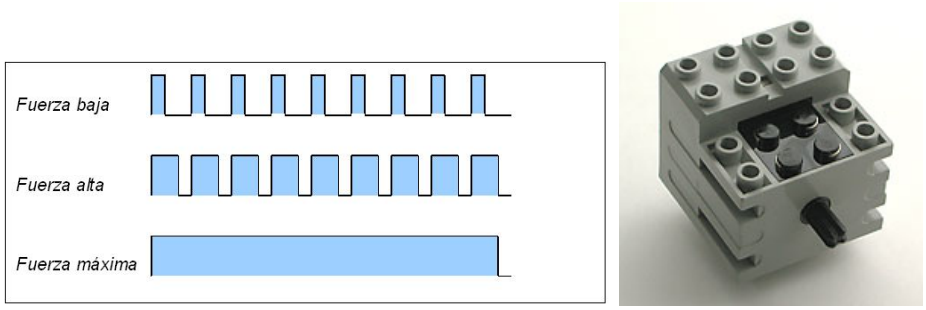


Figure 2.2: left: Output voltage for different motor-speeds using PWM; right: LEGO motor 43362

The RCX normally is powered by 6 AA batteries with a total voltage of 9V. The RCX 1.0 version additionally provides a charging bush which also can be used as external power supply.

2.1.2 Motors

LEGO provides about 6 very diverse DC motor types, mostly part of LEGO Technics but also compatible with Mindstorms. They are described in detail in [8]. The RIS only includes one of them, the 43362.
This motor is very efficient for its low weight of 28g. Its maximum speed is about 340 upm and the stalled torque is about 5.5 Ncm, both measured at 9V.
It can move in both directions, depending on the output polarization of the RCX which can be switched. As the motor output is digital and can only have the values 0V or $\pm$9V the motor speed is controlled by pulse width modulation (PWM). As shown in 2.2 this method uses a periodic digital signal to generate different power levels, which is done by varying the ration between the durations of 0 and 1. At low power levels only brief pulses with 9V are generated whereas at full power the pulse never ends. By using PWM the LegOS firmware allows 256 different speed levels.
As LEGO parts are developed as toys, the precision mostly is not very high. Hence also the motor shows variation of velocity up to 11% within different exemplars. Also the velocity of a single motor is fluctuating and oscillating, supposably caused by the internal motor

Figure 2.3: Gear, rotor and stator of the LEGO DC motor 43362.

gear, showed in 2.1.2 with other motor components.[26]

2.1.3 Sensors

There are four different types of standard LEGO sensors; touch sensor, light sensor, rotational sensor and temperature sensor. The following section describes the properties of these sensors.

For the vehicle design and control experiment only the rotational sensor was used. The basic idea behind the sensors is that the voltage on the input is converted to an internal RAW value in the range of 0(0V) to 1024 (5V). This value is processed further what results in 3 values for each sensor, whereas every sensor type is connected with one of theses types by default.

The three available values are:

raw value

Voltage level between 0(0V) and 1024 (5V)

boolean value

The raw value is converted into 0 (raw $<$ 460) and 1 (raw $>$ 562). In the range between 460 and 562 the previous value is kept to avoid clattering effects.

sensor type	code	sensor class	processed
empty	0	passive	read
contact	1	passive	Boolean
temperature	2	active	Celsius
light	3	active	percent
rotation	4	active	steps

Table 2.1: Available LEGO standard sensors and classification.

processed

Every sensor can have a more complexly processed value with a specific unit, e.g. percentage of the maximum value (light sensor), temperature or an encoder value(rotational sensor)

Furthermore every sensor can be passive or active. Active means that the sensor has to be powered to work properly. For example the rotational sensor internally uses a light barrier that needs power supply. Passive sensors normally are resistances and switches, that don't need to be powered.
Table 2.1 gives a brief overview about the 4 standard sensors in respect of sensor class and type of processed value.

Touch/contact sensor

A LEGO touch sensor is a simple sensor with a sliding part at the front to determine when a contact is made. It returns 1 in basic state and 0 when pressed.

Light sensor

LEGO light sensor is a complex and an inefficient sensor type. The sensor has a small red LED in the front, which flashes during operation. The light sensitive unit next to it reads the ambient light value. It can be used as active sensor measuring mainly the reflected light of the LED, e.g. for line-tracking if it's fixated near to the ground. So it can differentiate between dark and bright zones. It also can be used a passive sensor for measuring the ambient light. In this

case the LED is not powered.[27]

Rotational sensor

The rotational sensor works as an angular encoder and allows to detect the rotation angle of an axis in steps of 22.5° (16 pulses per revolution). The pulses are summed up to an integral value which can be initialized to an arbitrary value at any time. The sensor can differentiate between clockwise and counter-clockwise rotation. The technical details of this sensor will be treated more closely in 3.1.1.

Temperature sensor

The temperature is the less known sensor as it's not very useful in most applications. It can read the ambient temperature in a range of - 20 to 70 °Celsius. The value also is available in °Fahrenheit.

2.1.4 USB-Tower

For infrared communication with the RCX LEGO delivers two different devices, the older serial IR-tower and the newer USB version, which is used in this case. It's working on a frequency of 76 kHz and supports a transmission rate up to 4800 baud, the standard of the recent RCX 2.0. The integrated filter also has a bandpass at 38 kHz to provide compatibility to the RCX 1.0.[14]

2.2 The LEGO-Vehicle

After introducing the most significant LEGO pieces in the following section the construction of the vehicle used in the experiments will be described more detailed.
For the design initially a few demands were made considering drive and geometry. It was decided to use a tracked vehicle with a design as small as possible. It should be powered by two LEGO motors, each for one track. Furthermore, corresponding geometry, the cen-

ter of gravity should be close to the geometric center for decreasing the modelling errors. Additionally a holding for the Bluetooth adaptor and a color marker have to be attached.

2.2.1 Construction

Therefore the chassis was built by perforated girders on left and right side each. A double perforated girder in the center supplemented by the outer girders are used to fixate the four half axes. This basic construction is stabilized by LEGO plates attached above and underneath on all areas not blocked by the gear. This basic construction is shown in figures 2.4 and 2.5. There also additional perforated girders at the front axes are visible. They are used to shorten the length between the two bearings for play reduction.

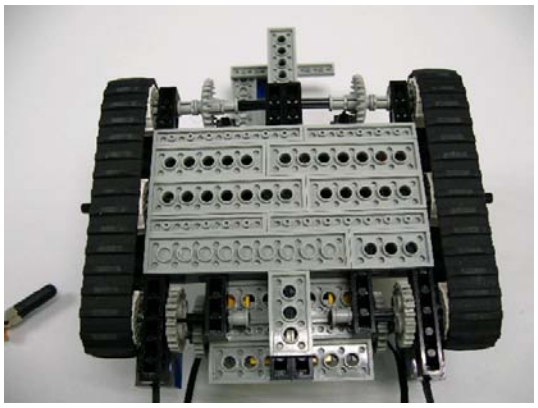


Figure 2.4: Bottom view: The chassis is stabilized by several LEGO plates. Also the construction of front and rear axis can be seen.

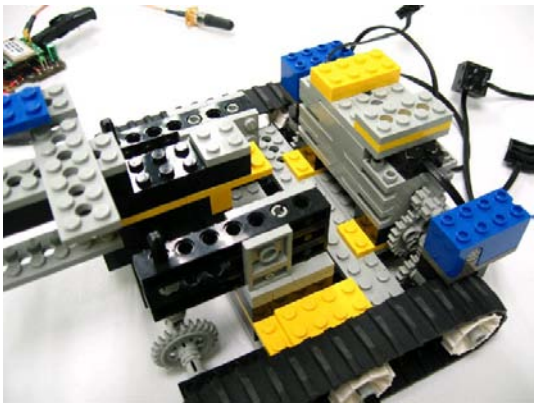


Figure 2.5: Top view on chassis without RCX showing motor attachment and holding for RCX and BT adaptor.

As already mentioned, for each wheel an independent half axis was used. Therefore the wholes in the girders were used as bearings. The position of the axes was fixated by LEGO sleeves.

The motors were attached directly on top of the front axes as can be seen in figures 2.5 and 2.6. The gears (Fig. 2.7) are consisting of the motor and drive axes, connected by one

additional axis for decreasing the ratio. Thus the complete gear ratio is 1:9, composed by the two sequential ratios 8:24 (motor:middle axis) and 8:24 (middle axis:drive axis).
The drive consists of two LEGO rubber tracks, lead by originally 4 wheels, as the standard Mindstorms kit only contains 4 such pieces. During the experiments it became obvious that this construction is complicated in respect of varying slip. As the tracks aren't very

Figure 2.6: Front view of motor and gears.

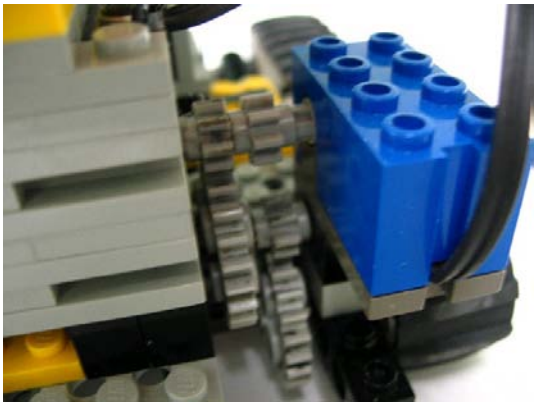


Figure 2.7: Zoom on gear and rotational sensor. The sensor is directly attached on the motor axis.

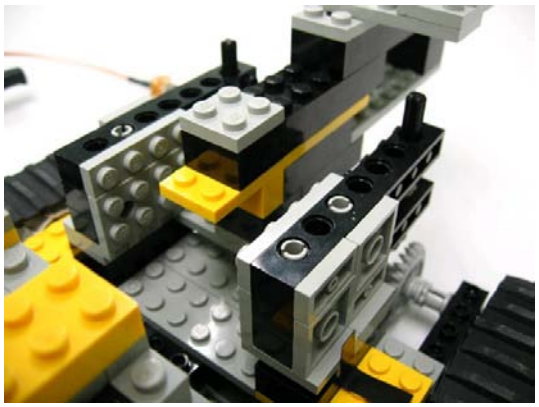


Figure 2.8: Zoom on RCX holding. Due to the motors the RCX has to be positioned very high.

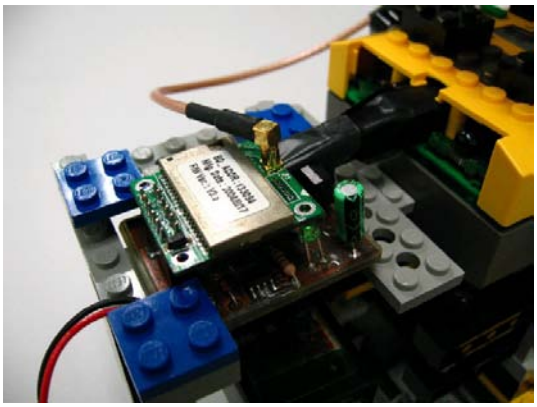
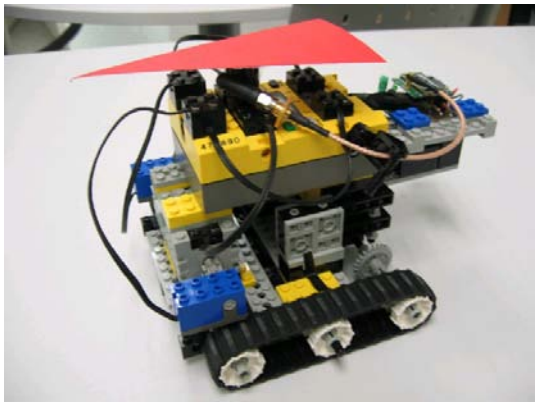


Figure 2.9: Holding for the BT adaptor. Also the tube connecting the adaptor's IR emitter with the RCX IR receiver is visible.

tense they arch in the middle of the vehicle and so there mostly is no contact between tracks and ground. Therefore two additional wheels were added in the vehicles center to equally distribute the weight on the track area. Using the 6-wheel configuration a much more uniform slip behaviour was observed.

To level weight distribution and counteract the motors weight a holding for the RCX was constructed in the rear section of the vehicle (Figures 2.8 and 2.5). There also the holding for the Bluetooth adaptor, shown in figure 2.9, is attached, which fixates BT card and battery. The resulting vehicle can be seen in figure 2.10 and possesses the following parameters:



Technical data:

max speed:	≈4.5cm/s
length:	19.5 cm
width:	15.5 cm
height:	14 cm
weight:	≈640 g

Figure 2.10: Complete LEGO vehicle with red marker for position detection.

2.2.2 Sensor positioning

Another important part of the vehicle's design is the rotational sensors' position. As will be discussed in chapter 3.1, because of the low angular resolution and long duration between two pulses (for low speeds 50ms and more), the rotational speed in this application is in a critical region and very important for the quality of the velocity measurement. For low speeds, long delays to the next update of the velocity value occur. So a higher ratio between axis and sensor will result in higher rotational speed and consequently a better quality of the velocity measurements.

Besides limited space and number of pieces, the most significant constraint for increasing the ratio is given by the friction added to the gear with every additional cogwheel or ratio. Furthermore it has to be considered that mechanical gears are inert, so every ratio will slow down the system response time. In this respect it is also important that the sensor is placed as far as possible from the system input at a place were the desired output can be measured (best positions here: rear axis or directly on front axis). There the physically relevant value, the rotation that is directly transformed into track movement, can be measured including all delays of the gear.
This demand here is put into perspective by the bad quality of the velocity measurement for low speeds. Every decrease of rotational speed, caused by additional ratios between motor and sensor, severely decreases the signal quality. So a compromise for the sensor position has to be found.
For the previously described vehicle several possible sensor positions were tested and compared in the following configurations:

1. Sensor connected to motor axis by additional cogwheel
2. Sensor directly on motor axis (Fig. 2.11 left)
3. Sensor on (non-motorized) rear axis, connected by a 90° redirection and an additional ratio (Fig. 2.11 middle)
4. Sensor on front axis with additional cogwheel to increase rotational speed (Fig. 2.11 right)

Finally, it was decided to use option 2 because it contains least friction at a acceptable ratio of 0.53 mm/step. Options 1 and 3 introduce a high level of friction that results in velocity oscillations and a wide deadzone (the range of input values to the motor where no movement occurs because of friction). As option 4 also increases friction without augmenting the ratio it also was discarded.
As shown in figure 2.12 the delay induced by the gear can be neglected in comparison to the delay induced by the sensor. It shows the velocity value measured on the front axis

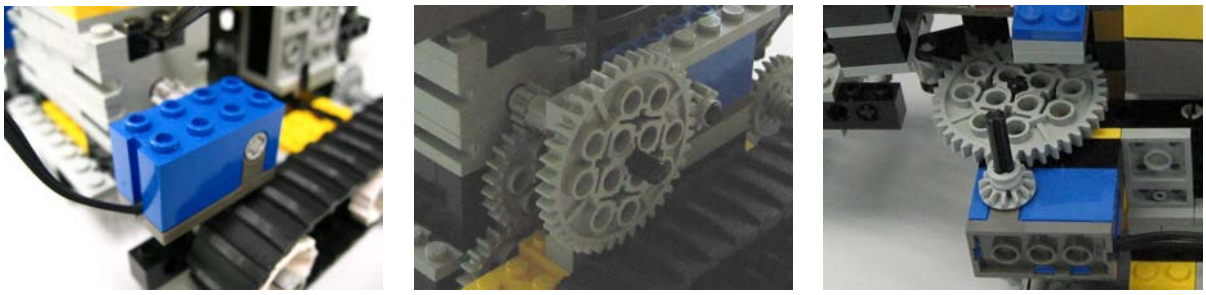


Figure 2.11: Encoder on positions 2) 3) and 4).

and directly at the motor axis. As the ratio differs by a factor of three (ratio 8:24), most of the additional delay of the velocity of the front axis is generated by the slower sensor dynamics.

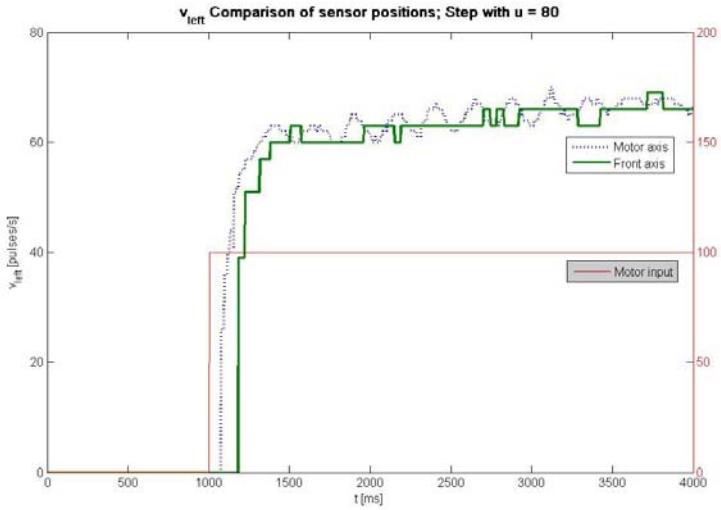


Figure 2.12: Comparison of encoder signal at different positions. The sensor on the front axis is about 100ms slower. A major part of this difference is caused by the lower ratio and resulting longer pulse duration.

2.3 Cameras

For image processing and position detection of the vehicle two different cameras were tested, the Logitech Qickcam Pro 3000 and the LEGO camera, which is available as a add-on for LEGO®-Mindstorms™.

Logitech Webcam

The Logitech QuickCam® Pro 3000 is equipped with a built-in microphone and a CCD chip with a real resolution of up to 640x480 pixels and a 24-bit color depth (16.7 million colors). Possible frame rates are 15 or 30 Hz. The lens aperture is F/2.0 and the drivers guarantee an optimal lighting. It has to be connected to a standard USB port.

LEGO Webcam

The LEGO Studios camera/Mindstorms Camera internally corresponds to a Logitech Quickcam Web. The CMOS chip also allows a maximum resolution of 640x480 and 30 fps (framse per second), but it has to be very well lit to perform at any satisfactory level. Here also a microphone is integrated and the camera is delivered with a extra long 5m USB cable.[2]

Comparison and choice

After beginning to work with the Logitec camera, it turned out to be difficult to adjust it in a position exactly parallel to the ground because of its unstable holder and the spherical form.

Here the LEGO camera has advantages because of its box form and the possibilities to

build a stable holder of LEGO pieces. Attached on a board which can be levelled, a very exact positioning can be achieved.
The drawback of the LEGO camera is the bad image quality. The picture noise is much higher, compared to the Logitech camera. As a consequence there is a drastic quality reduction of the image processing result because of variations in the segmentation algorithm. Furthermore above an altitude of 1.50m the ground is no longer focusable, which results in a loss of contrast what additionally deteriorates the outcome.
To get better image processing data it was decided to work with the Logitec camera because of its better quality . A fixation was constructed as can be seen in 3.10. Furthermore a satisfying result for the levelling could be achieved by the method described in 3.11.

2.4 Communications

2.4.1 Bluetooth specifications

The Bluetooth wireless technology is a short-range communications technology intended to replace the cables connecting portable or fixed devices. Main features of Bluetooth technology are robustness, low power drain and low cost.
The Bluetooth specification defines a uniform structure for a wide range of devices to connect and communicate with each other, up to eight devices simultaneously, mostly within a 10m radius (class 2). Such an ad-hoc network is called piconet and one device can belong to several piconets simultaneously.
The operating range depends on the device class:

- Class 3 radios : range of up to 1 meter
- Class 2 radios : range of 10 meters, most commonly found in mobile devices
- Class 1 radios : range of 100 meters, used primarily in industrial use cases

The Bluetooth 1.0 standard allows 1 megabit per second (Mbps) which is increased in the recent version 2.0 to 3 Mbps. Therefore it uses a very weak, frequency hopping and full-

duplex signal between 0.5 - 1 mW within the unlicensed industrial, scientific and medical (ISM) band at 2.4 to 2.485 GHz.
To avoid interference a technique called spread-spectrum adaptive frequency hopping (AFH) is used which makes it rare for more than one device to be transmitting on the same frequency at the same time. AFH works within the spectrum to take advantage of the available frequency. This is done by detecting other devices in the spectrum and avoiding the frequencies they are using. This adaptive hopping allows for more efficient transmission within the spectrum, providing users with greater performance even if using other technologies along with Bluetooth technology. 1,600 times every second the signal hops randomly among 79 frequencies at 1 MHz intervals to give a high degree of interference immunity.
Bluetooth offers several security modes as e.g. 'trusted devices', which can be defined and can exchange data without asking permission. Every other data-transfer has zu be confirmed by the user. It's also possible to make the device 'non-discoverable' and avoid connecting with other Bluetooth devices entirely. Security methods include authorization and identification procedures.

2.4.2 The Bluetooth-Infrared-Device

In order to overcome IR limitations of the RCX, a Bluetooth wireless solution has been developed at the AI2[31].
Bluetooth has be chosen because of its easy handling, already integrated protocols and the 'serial port' profile which makes it compatible with the IrDA signal. It consists of a small electronic circuit on a 4.5 x 4.5 cm card that has to be posed in front of the RCXs' IR port. The overall scheme is shown in figure 2.13. The core of the system is commercial Bluetooth chip (UART standard with TTL-CMOS compatible logical levels), the Initium PromiSD, with a range of 30m and a maximum data rate of 115200 baud. This device was chosen because of its low cost and small dimensions.
The output of the Bluetooth chip is connected to a switch inverter (MAX3233 by Dallas-

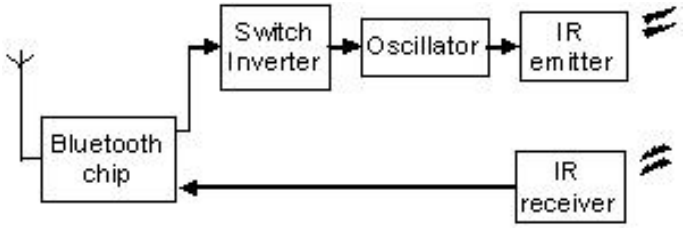


Figure 2.13: Scheme of the Bluetooth-IR-adapter.

Maxim) and an oscillator (TLC555CDR by Texas Instruments). If the chip receives (from the PC) a zero bit, the oscillator generates a pulse signal of 38 KHz, for a logical one the oscillator is switched off. The oscillator output is connected to an IR emitter.

The reception stage is simpler as the TSOP-34838 (by Vishay) IR receiver already delivers a processed logical signal and is connected directly to the Bluetooth chip. This receiver was selected because of its robustness with the noise and ambient light interferences.

For power supply at TTL tension level (3.3volts) the linear voltage regulator MAX884 (by Dallas-Maxim) was integrated.

The serial communication protocol used by LEGO, works at 2400 bauds, odd parity and one stop bit. The bit encoding is 470μs where a IR '0' bit is encoded as a 470μs pulse train of 38 KHz, and a '1' bit as 470μs of nothing (0V).

In this way, after an adaptation of IR serial signal to logical levels for the Bluetooth chip, the communication between RCXs and PC or among RCXs can be performed.

In [13] it is mentioned that in some cases it is even possible to run the RCX 1.0 with a USB tower at 4800 baud, which can be defined in the firmware. But tryouts for decreasing the transmission time with the used RCXs weren't successful, so this statement couldn't be verified.

On PC side the Conceptronics CBT100U and CBT200U2 USB Bluetooth USB adapter were used in combination with the software Blue-Soleil. Both adapters are class 3 devices and support all common Bluetooth profiles. CBT100U has a range of 100m and maximum data rate of 1 Mbps (Bluetooth 1.0), while the range of the CBT200U2 is 200m at a maximum rate of 3 Mbps (Bluetooth 2.0).

2.4.3 LNP - The LEGO-Network-Protocol

Particularly with regard to bit errors in the communication, the IR-connection between the RCX and the Bluetooth-module is the most critical part. Hence many application use the 38kHz carrier frequency, interferences with other IR-signals are possible, as well as reflections or ambient light can influence the signal quality [30]. Therefore with the introduction of the legOS (now BrickOS) the LEGO Network Protocol was developed [30]. It guarantees a received data-packet to be correct, otherwise the packet is discarded. The correctness is verified by a running checksum over all bytes of the message (with initial value 0xff). The drawback of the LNP is that, indeed errors can be detected by the receiver if the packet arrives, but no information for the sender is available if the packet was received.

The LNP was also developed to communicate with various RCX units at the same time. Therefor the LNP can be used in two different ways, each with an own protocol implementation:

Broadcast Transmission

Messages sent by this protocol layer can be received by any unit in the system. The packet length can variate, but with a rising number of data bytes, the percentage of undetectable errors rises. Because of that the LNP packet length of the JAVA implementation in leJOS is limited to two bytes.

A LNP-packet shows the following structure

H (0xF0)	LEN	D	CHK

H : Header (0xF0)

LEN : Number of data-bytes (1..255)

D : Data

CHK : Checksum

Figure 2.14: Structure of a LNP broadcast transmission packet.

Addressing Transmission

For sending messages to a specified target the LNP addressing layer is used. Therefor the packet-structure is expanded by four bytes, two for the address of the destination and two more for the source address.

With this modifications a client will reject a message with a destination other than his address and receives in addition the address of the sending client (source).

The packet structure changes as follows:

H (0xF1)	LEN	DEST	SRC	D	CHK

H : Header (0xF0)

LEN : Number of data-bytes (1..255)

DEST : Address of destination (2 bytes)

SRC : Address of source (2 bytes)

D : Data

CHK : Checksum

Figure 2.15: Structure of a LNP addressing transmission packet.

On the firmware side receiving data is handled via interrupts, while the transmission is running as a background process.

An incoming packet will trigger an interrupt routine, wherefor an interrupt handler has to be defined first as followed

```
#include lnp.h
⋮
void my_integrity_handler(const unsigned char data, unsigned char len)
{ ... }
```

or

```
void my_addr_handler(const unsigned char data, unsigned char len, unsigned char
src)
```

```
{ ... }
```

where data is a byte-array filled with the content of the packet, len the number of bytes received and src, in case of addressing transmission, the address of the sending RCX.
For using this interrupt handler it first has to be declared in the main method

```
lnp_integrity_set_handler (my_integrity_handler);
```

or

```
lnp_addressing_set_handler (my_port , my_integrity_handler);
```

Once a packet was completely received, the firmware calls the handler function to process the incoming data. As this function is an interrupt routine, it should be as short as possible to loose no other interrupts. Also methods dealing with memory operations and thread priorities (as for example the timing-methods sleep and msleep) don't work within this method and will lead to unpredictable behaviour of the RCX program. The best way to initialize processing of longer subroutines is to create a static command buffer and only insert the command byte and the parameters in the buffer. Processing the command should be done by a regular thread.[15]

Sending a packet can be achieved by using the functions

```
result = lnp_integrity_write(data, length);
```

or

```
result = lnp_addressing_write(data, length, DEST_ADDR, MY_PORT);
```

which generate the package structure including the checksum and add the data to a transmission buffer, which holds the next bytes to be transmitted.

The legOS-firmware is processing incoming data byte-wise with a state machine. While no packet is processed it is waiting for a specific header byte to initiate further processing. For each type of packet, specified by a unique header, an independent path of states if followed until processing is finished, either by a error-caused reset or completion of the package.

Some very powerful complements to the LNP-implementation in the legOS-firmware

were developed for Linux (LibLNP) and Windows (winLNP) operating systems. They allow to access a serial- or USB-tower to send and receive LNP-messages. Furthermore there exists a Linux LNP Daemon that is running as a service in background to constantly receive packages and prevent the tower from being shut down. It can be easily contacted by other programs to communicate with the RCX.[30]
This structure is also included in the pcrcxcomm JAVA package, a PC-side implementation of communications that comes with the leJOS firmware but can easily be used with BrickOS. Here the developers even went one step further and developed a proxy that routes the LNP messages into the local network. The RCX has to send the requested IP address or host name to connect with when initializing the connection and thereby can easily establish a socket connection to a TCP server running on any PC in the LAN.

By using the Bluetooth-IR interface, the addressing layer is irrelevant because every vehicle is equipped with it's own Bluetooth card and is therefore addressable by a unique emulated serial port. Sending a message to a specific serial port corresponds to addressing transmission, for a broadcast the same message has to be send to all serial ports.
However first tests of the message speed showed that transmitting data with the LNP is very slow because of the huge amount of additional data. For a data packet of 2 bytes, which is the intended packet-size, 5 bytes must be sent over the IR-interface. In the firmware transmitting one byte is supposed to take 30ms (5 ms transmission + 20/15 ms waiting before/after the transmission), which means a total time of at least 150ms for 2 bytes, which is not acceptable for a continuous control of the LEGO-vehicle.
Therefor changes in the firmware have been made, as well as a new transmission protocol has been introduced, which is described in the following section.

2.4.4 LCP - The LEGO-Control-Protocol

Using the LNP for the intended control application has two mayor drawbacks: low speed and no information about if the transmission was received. Therefore a new protocol had to be integrated in the firmware.

To overcome the drawback of no reception information it was tried to use acknowledges that are sent after a packet arrived. But very soon it was clear that the thereby generated time delay was not acceptable for control applications, and especially the problem of waiting timeouts appeared to be critical.
On the other hand reliability test of the communication showed that installing the Bluetooth card directly in front of the IR port of the RCX reduces the package loss to a negligible level. It only gets problematic when the power level of the Bluetooth-adapters battery drops significantly. Furthermore some randomly occurring extremely long delays during the transmission were observed, which lead to the assumption that packet-losses/timeouts on the controller-side have to be handled anyway.
So the matter of possibly lost packages was no longer pursued and the new protocol was developed for shorter transmission times.

The new protocol was designed in respect to a minimum package length with acceptable error detection capabilities. To further guarantee compatibility to other protocol types, the header byte system was adopted.
The intended use was to send the signal of the rotational sensors from RCX to PC and the reference speed for left and right side from PC to RCX. So a dataword length of two bytes seemed to be appropriate. The changes in the rotational sensors are small enough to filter out the overflows of the byte value-range, which is done in the receiving MATLAB-function. An overflow can be detected when the absolute change of the value $x_t - x_{t-1}$ is greater than a pre-defined threshold (here: 150). The sign of the change depends on if the overflow passed the upper or lower bound of the value range.
Corresponding the reference speed a byte is more than sufficient. The speed is measured by firmware in steps per second (integer-valued), which usually has values between 20 and 80 in this application. Only interpretation as signed byte must be assured, so positiv and negativ speed can be sent directly. In case of larger values would be required, the value can be scaled on both sides of the channel.
To shorten the packet length as much as possible, error detection and compatibility to the

protocol system were reduced to one byte. The first 4 bits represent the header (0xaX), the last four bits the checksum.
As result, the packet structure of the LCP (LEGO-Control-Protocol) looks as follows

H (0xA0) $\vee$ 0x0CHK	D

H : Header (0xA0)
CHK : Checksum (4 bit)
D : Data

Figure 2.16: Structure of a LCP packet.

The remaining question of how to get a 4-bit checksum of a 2-byte dataword will be described in the next section.

Error detection

Whenever digital data has to be send through a communication channel or stored on a memory device, the risk of random bit changes occur, in case of communications by sporadic interferences, in case of memories by physical defects (hard errors) or disturbances (soft errors) caused for example by α-radiation.[40]
For detecting such errors exists a variety of detection mechanisms, each with different detection abilities in respect to single/double-bit errors, systematic errors. In general a message or stored amount of data has to be split into datawords of length d. Every dataword is embedded in a codeword of length $c > d$, such that a checkword of length $s = c - d$ can be generated by a certain algorithm.

CW =	DW	PW

The most common techniques are:

Repetition schemes

A message is sent a fixed number of times. This is not very efficient and problematic

for systematic errors, where the error occurs always on the same position.

Bit Parity

Only one bit is used as checkword (s=1); It is generated by a bit-wise XOR of the dataword and complements the total count of ones to a even number (even parity). The drawback is that double-, quadruple-, sextuple-,...errors can not be detected. This can be achieved by inversion, which results in a complement to an odd number of ones (odd-parity). Parity-checks are not very strong and are only used for small datawords of mostly 8 bits, such as in serial connections for instance.

Running Checksum

One of the easiest, but also very basic error-detection mechanisms is to use a checksum of arbitrary length s as realized in LNP. Initialized with a fixed value, the dataword is added in parts of s bits. Overflows are ignored, so the checksum always stays within the range of s bits. The algorithm is very rudimentary because only one or two bits of the checkword are influenced by a bit-position in the dataword, so that some types of errors stay undetected even for very long checkwords. An example in the following section will demonstrate that.

CRC - cyclic redundancy checks

CRC's are one of the most popular error-detection techniques. Although they are computational extensive they can easily be realized in hardware, which is the reason they are mostly used directly on devices like Ethernet, CAN, USB and Bluetooth. CRC's can be used for any dataword length and mostly use a s, which is a multiple of 8, as e.g. the CRC8, CRC16, CRC24, CRC32 and CRC64. An exception is the CRC4 and CRC5 which is used for very short datawords.

Different from a checksum, the CRC algorithm uses division instead of addition, which makes the algorithm more complex. The theory CRC's will be treated more closely in the following section.

Hash functions

Better known from search- and encryption algorithms, hash functions can also be used for error detection.
A hash function H(x) is an algorithm that generates a fixed-size string h (hash value) from any input x. It's major attributes are:

- The input can be of any length
- The output has a fixed length
- H(x) is relatively easy to compute
- H(x) is one-way
- H(x) is collision-free

Instead of using the hash value for searching in a list, it also can be used as checkword with very high complexity. H(x) being (almost) collision-free guarantees that almost no pair of different messages can have the same hash value.[18]
The most known hash functions are algorithms of the families MD (md = message digest, as MD2, MD4, MD5, ...), SHA (SHA0, SHA1, SHA256, ...) and TIGER (TIGER, TIGER2, TIGER128, ...). MD and SHA are often used to generate so-called fingerprints of documents and files for faster search and comparison.
Mostly hash functions are computationally more expensive than CRC's in case of long messages.

Although CRC and hash functions are no sum but complex functions, in the context of error correction methods mostly the term checksum is used. This is a remaining habit from the times, where only parity-checks were used, which are general spoken a bit-wise sum of the dataword.[41]
The checksum used in LNP is a real checksum. The bytes of the message are summed and the resulting byte added to the message as checkword. The problem of this algorithm is its simplicity. If, for example, two random corruptions occur, there is a 1 in 256 chance that they will not be detected.

Message	: 6 23 4
Message with checksum	: 6 23 4 33
Message after transmission	: 8 20 5 33

Even a longer checkword wouldn't help to detect this error. Using a 16-bit register instead of a 8-bit register (e.g. sum the bytes mod 65536 instead of mod 256) would fail because in this case the formula used is not sufficiently 'random'. Each byte influences roughly only one byte of the summing register. Therefore an algorithm that influences wide parts of the register is needed and can be found e.g. in the CRC algorithm.
So two main properties characterize an error detection algorithm:

- Length of the code word
- Complexity of generating algorithm

The desired amount of data included in one LCP packet is 2 bytes. With a possible length of 4 bits for the check word it was obvious to use the CRC4 algorithm that is widely used and already proven to work properly and robust in technical applications (e.g. Cisco router [5]). The functionality and theory of this technique is described in the following section.

Theory of Cyclic redundancy checks

As summation appears to be too simple for a robust error-detection algorithm another arithmetic operation is more complex: division.
The basic idea of CRC is to use rest of a polynomial division as checksum. The bit ordered message is regarded as a dyadic polynomial T(x)(a polynomial XXX whose coefficients only can have the values 0 or 1) of degree d-1 (data length) and is divided modulo 2 by a certain generator polynomial G(x) that has to be used by both sides, sender and receiver. The remaining rest E(x) represents the CRC value and is attached to the message.
The receiver can verify the data integrity by performing the same operation to the message including the CRC value (T(x) + E(x)). If no rest remains, either no error occurred, or

an error that contains G(x) as a factor has happened. It is obligatory that sender and receiver use the same polynomial and processing technique. A worked out example of the calculation of such a CRC can be found in A.

Polynomials

The polynomials normally are written as a bit sequence where the most significant bit represents x^{n-1} and the least significant bit, which is always 1, represents x^0. For example, the polynomial used in CRC4 can be written as:

$$1x^4 + 1x^1 + 1x^0 = 1x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0 = 10011 \tag{2.1}$$

It also exists a second representation with reverse order of the polynomial coefficients, but it is less intuitive and less used.
In both representations the most significant bit has to be one (to guarantee the polynomial order) and is omitted for the division.

Not every polynomial is equally useful for error detection, and so there exist a few building rules to be regarded when choosing a polynomial:

1. For G(x) more than one term, all single bit-error will be detected
2. By guaranteeing that G(x) does not divide $x^k + 1$ for any k up to the frame length, also all cases with two isolated single-bit errors are detected
3. By making x + 1 a factor of G(x) all errors consisting of an odd number of inverted bits can be caught
4. r check bits will detect all burst errors (several wrong bits in a row, e.g. by an interruption) of length $\leq r$

Certain polynomials have become international standard as for example the polynomial 0x04C11DB7, known as CRC-32 (IEEE 802) used in Ethernet, FDDI, ZIP and PNG.
The CRC4 polynomial (eq. 2.1) used for LCP also is a standard that is used in technical applications [5].

With the coefficients $x^5 + x + 1$ it is able to detect single and double bit errors, errors with an odd number of wrong bits and burst-errors up to a length of 3.

Technical realization

CRC's mostly are realized by hardware as they only need shift register and XOR logic. The software implementation mostly is more complex but still within justifiable limits of calculation time, which particularly is important for the implementation in the RCX. It can be speeded up by lookup tables ([41]), which only is realized in the PC-side because of lack of memory on the RCX.
The following pseudo code shall demonstrate the shift register algorithm, a complete C code sample can be found in [23].

```
shift register sr := 0000 (initial value)
while ( data-bits left )
{
        if ( most significant bit of sr ≠ next bit of data)
              sr := ( sr shift left ) XOR CRC-Polynomial
        else
              sr := ( sr shift left )
}
CRC-value := content of sr
```

Method variations

In some cases (such as CRC32) an initial value different from 0 is chosen for the shift register, mostly filled with ones. Equivalently, the first n bits of the message may be inverted before feeding them into the algorithm. This is done because an unmodified CRC does not distinguish between two messages which differ only in the number of leading zeros.

A second very common variation is to bit-wise invert the CRC value before attaching it to the data frame. A message only consisting of zeros will always have zero as CRC value. If a total breakdown occurs in the transmission channel, such that only zeros are read, the CRC will not detect it. This is known as the 'zero-problem' and can be solved by simply turning the zeros of the CRC value into ones by inverting them.

Both of this modifications aren't useful in LCP and though not used for the reason of computing time. The message data is clearly delimited by the header byte, so no sequences with varying leading zeros can appear. An inversion also isn't necessary because in case of a total breakdown the serial protocol guarantees that no data is read.

2.5 Software

After introducing the used hardware and technical details, the following section shall give a brief overview about the software and software technologies used for the experiments.

BrickOS

BrickOS is an alternative operating system for the LEGO Mindstorms RCX which was originally developed with the name legOS by Markus L. Noga and first published in July 2000. It also provides a C/C++ development environment for RCX programs using gcc and g++ (the GNU C and C++ cross compilation tool chain) and the necessary tools to upload programs to the RCX. Compared to the standard firmware it provides some clear advantages as:

- Theoretically unlimited number of variables (only limited by the memory size)
- New data types like matrices and arrays
- Floating point calculations, but these are very time consuming
- Enhanced motor functionality (255 instead of 8 levels)
- Full compatibility to C, what allows reuse of code for other platforms

- Many additional library functions for directly accessing the hardware
- A (quasi) real-time kernel that allows multiple simultaneous processes and timing operations
- Control of interrupts and synchronization via semaphores
- Implementation of LNP communication protocol

Considering all the functionality legOS makes available, it gives a level of control that can't be matched by any other platform. Especially for control purpose the real-time kernel and multi-threading were needed. Furthermore it is published under open-source license and thus all code is available. By its modular structure it can be easily altered, adjusted and extended for specific problems as used several times for adding LCP and velocity calculation. [3],[30],[6]

Bricx Command Center

To programm the RCX the most comfortable solution is given by Bricx Command Center (BricxCC), which is an open-source Windows based developing environment for the RCX (all versions). It's graphical user interface is very intuitive and very similar to known programming environment as Eclipse or Visual Studio, albeit with less functionality and specific on RCX programming. Therefor it supports all common programming languages as Not Quite C (NQC), MindScript™, LASM™, C, C++, Pascal, Forth, and Java using the brickOS, pbForth, and lejOS alternate firmwares.
BricxCC is an enhanced version of Mark Overmar's RCX Command Center, which was developed only for NQC programming. It is equipped with many additional tools for firmware and program upload, LNP configuration, graphical remote controls and communication tools which are helpful for programm development and debugging.[4]

JAVA™and Eclipse

For establishing Bluetooth communication, image processing and ethernet data transfer it

was decided to use JAVA in the Version 1.4.2 to sustain compatibility to MATLAB.
This object-oriented language by Sun Microsystems, first available in Version 1.0 in 1996, is getting more and more common and accepted. Reasons are its platform independence, free availability of several development environments and application programming interfaces (APIs) as well as good use in web applications.
The decision for JAVA was taken due to its platform independence and available API's for network application and communication with the RCX.
Furthermore Eclipse (http://www.eclipse.org/) was used for program development in JAVA. It provides a very comfortable user interface and a big amount of additional, time saving help functions. In addition it supports direct use of the code version system CVS which was used for source code management and synchronization between the several PC's.

MATLAB/Simulink

MathWorks' MATLAB is one of the most successful standard tools for numerical engineering calculations, especially in respect of dynamic modelling and simulations with the Simulink toolbox. It is widely spread in industry and universities and also available at the AI2 laboratory in version 7.0.1. In contrary to computer algebra systems it is mainly used for numerical solutions, even though there exists also the possibility for symbolic calculations.
Numerous extensions and problem specific code packages are available in form of toolboxes covering bioinformatics, financial mathematics, image processing, system modelling, identification and simulation, etc.
Simulink is an additional toolbox for system modelling. It allows hierarchical modelling of discrete and continuous systems by using graphical blocks. The data flow is realized by lines connecting the blocks, what makes the modelling very intuitive and clear. Further, more complex blocks are sold by MathWorks and other manufacturers.

For this thesis the following toolboxes were used predominantly.

- **Image Processing Toolbox**

  Used for development of image processing routines like position and obstacle detection and camera calibration

- **Simulink**

  Used for system modelling, optimization of Extended Kalman Filter, controller simulation and realtime control

- **Spline Toolbox**

  Used for trajectory generation

- **Control Toolbox**

  Used for system modelling and parameter identification

The utilized software concludes the technical preliminaries needed for developing the laboratory experiment. So we can now turn on acquisition of sensor information and the resulting system and communication structure.

Chapter 3

Measurements and communications setup

In order to be able to control the vehicle it is necessary to obtain data about the system state. Only so we can apply the required feedback to the system and close the control loop. As the system state often is not directly measurable we have to obtain outputs containing enough data for reconstructing the state, by using an observer for example.
In the developed experiment it was decided to use the following system outputs: encoder values of the vehicle's rotational sensors and vehicle position obtained by the webcam. Position estimation only based on encoder values is almost impossible in systems with friction, slip and, as it is the case for the LEGO vehicle, input noise. So using position data of the camera is is required to enhance the position estimation. For measurement of the track-speed and better predictions of the movement the encoder values are helpful.

3.1 Encoder signal

Encoders are a widely used method for measurement of velocity and relativ change of position in mobile robotics. Other than the continuous working mostly electro-magnetic speed indicators, encoders are discrete counters that measure angular rotation in fixed steps

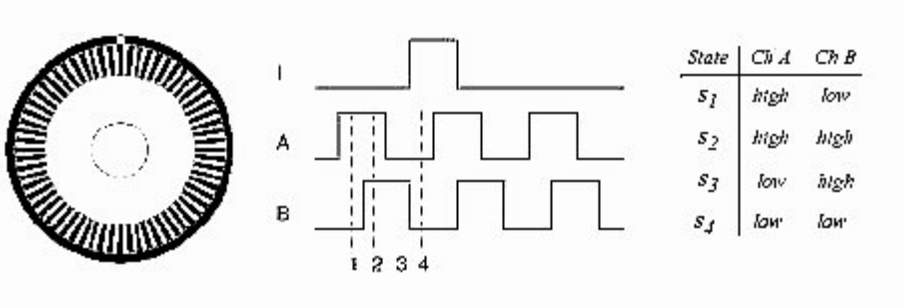


Figure 3.1: Quadrature optical encoder. The observed phase relationship between channel A and B pulse trains are used to determine the direction of the rotation. A single slot in the outer track generates a reference (index) pulse per revolution.

and count them. This can be achieved by magnetic or optical methods. Magnetic encoders consist of a magnetic cogwheel on the moving axis in combination with a hall sensor. The gaps between the teeth of the cogwheel produce a sinusoidal fluctuating magnetic field that it measured by the hall sensor.[38] The more common method in mobile robotics are optical encoders that work either by a reflection or light-interruption principle. Therefore a marking with a stripe pattern is attached either radial, on a disc fixed on the axis, or along a region on the axis (only reflection method). With a light barrier or LED-reflection sensor combination the bright and dark zones can be distinguished and produce a sinusoidal modulation of the light intensity which is measured by a photo sensor. One bright-dark-period is counted as one step. Additionally, by using two sensors shifted by 90° (of the pulse period) it is possible determine the rotational direction as showed in 3.1. By multiplicating the angular difference by e.g. the wheel radius it can be directly used to measure distances, assuming a constant wheel radius and no slip. For measuring the rotational speed the covered angle has to be differentiated. Therefore in most cases one of the following methods is used.

1. Counting the steps in a fixed time-interval. The number of steps counted, divided by the interval period give a linear approximation for the rotational speed, which again can be transformed into absolut speed using a geometric factor. This method is used

in applications with small pulse intervals compared to the sampling time.

2. In applications with less speed, where the pulse width is much longer that the sampling time, the velocity is updated after each pulse. The velocity than can be obtained by the inverse of the pulse duration.

In case of sampling time and step interval getting closer, a high discretization error will result. For method one the number of pulses per interval N will alternate between the two surrounding integer values which gets more severe the less the number of pulses is.
For the second method sampling time and step interval getting closer means high velocities. Here the measured interval T_P between two pulses can vary between $T_P - T_S$ and $T_P + T_S$ with T_S being the sample time. As the velocity v is equal to $1/T_P$ this also can cause significant errors.
The second method additionally is problematic in case of very low velocities (very long pulse durations) because an update of the velocity value is only possible when a new pulse is measured. Especially in the case of the velocity dropping to zero, no new pulse will occur and a timeout has to be implemented that sets the velocity to zero when being reached.

3.1.1 System extensions

In this application LEGO rotational sensor are used as encoders in two different ways. Firstly they are used to measure the track speeds by the RCX firmware what is necessary for the inner control loop (control of the vehicles velocity). Therefore the previously described method of measuring the time between two pulses is used.
Furthermore the encoder values are sent via Bluetooth to the outer controller running on a network PC. Here the encoder value is used by an observer to estimate the system state, what will be discussed in chapter 5. Therefore a sufficiently short sampling time has two be achieved such that the sum of all transmission delays (checksum calculation, infrared transmission, Bluetooth transmission and network transmission) stays smaller than the controllers sampling time to guarantee new measurements.
In both cases the standard legOS firmware has been modified.

Velocity measurement

The legOS firmware already contains a module for velocity calculation which is not activated in the standard configuration. After activating it and recompiling the reason for that became obvious as the velocity value is severely fluctuating due to systematic errors (Fig. 3.2).

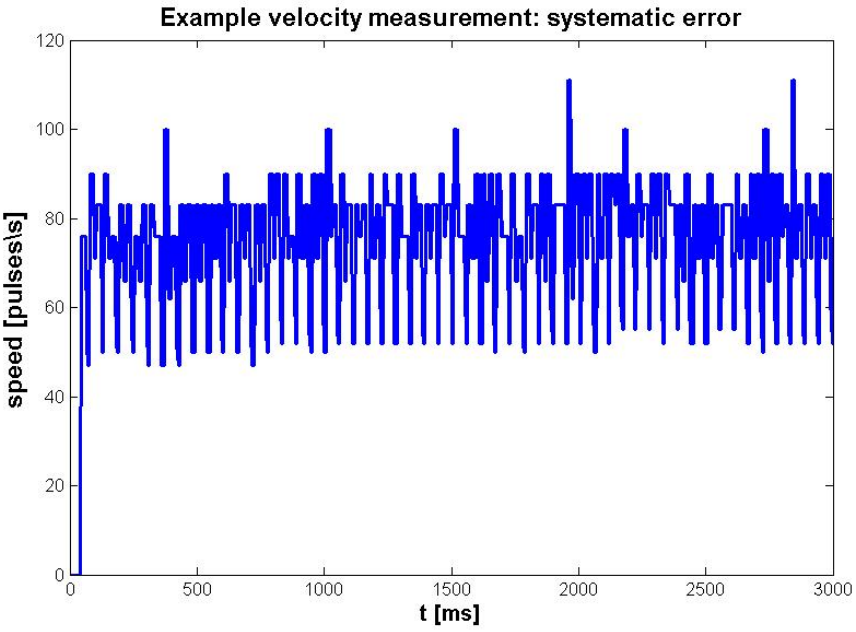


Figure 3.2: Velocity signal by original firmware implementation. The systematic error caused by the different interval lengths leads to fluctuations in the velocity value up to 50%.

For PID control the signal is not usable in this way and had to be filtered in some way. Usage of common FIR and IIR filters turned out to require too much computing time on the RCX and to be too slow. Thus analysis of the systematic error was made to find a computational less expensive way to enhance the signal quality.
In [8] the errors origin is partly described. The problem originates from the function principle LEGO used for the rotational sensor. Normally encoders use one bright-dark period to generate one pulse. The LEGO encoder only possesses four rotor blades and

uses one period for generating 4 pulses, one when reaching the bright or dark level and one each at intermediate levels. These voltage levels and the corresponding encoder values are shown in figure 3.4. The problem seemes to originate from different pulse widths, as for example the dark and bright zone cover different angular areas (fig. 3.3).
By analyzing the pulse widths of a constant speed signal the following variations were found for the four pulses.

1	2	3	4
14.78ms	8.7ms	10.3ms	8.75ms
34.7%	20.4%	24.3%	20.5%

Table 3.1: Interval ratios.

Here the first pulse corresponds to the dark zone. The pulse duration of dark and bright zone almost exactly build the same ratio as the angular regions (5:4). Especially the intermediate pulses are problematic as they appear to be much shorter.

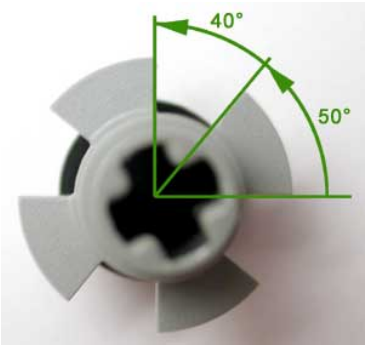


Figure 3.3: Zoom on the encoders' rotor: A clear difference between the angular region of rotor blade and free space exists. Source [8]

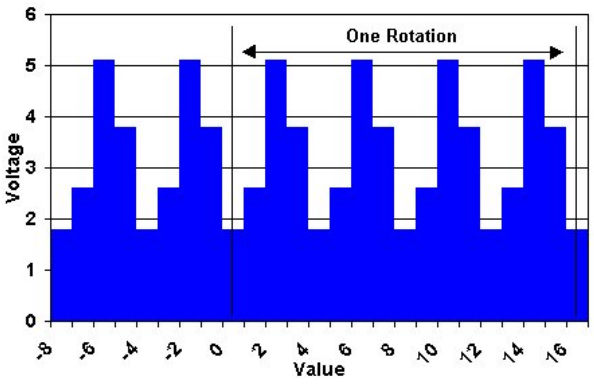


Figure 3.4: Voltage levels of the encoder raw value. Four different levels are returned per rotor blade pass. Source [8]

Using the knowledge of these systematic errors it was tried to improve the signal quality by applying a correction factor $\frac{meanpulsewidth}{pulsewidth_i}$ to each interval. Although the filtered signal oscillated with less magnitude the quality still was not acceptable, due to high variations

in the interval durations.

Thus a new approach using a kind of moving average filter was tested. Therefore the velocity value is calculated by

$$v = \frac{4}{t_k - t_{k-4}} \tag{3.1}$$

with t_k being the pulses' timestamps. Using this approach the desired signal quality was reached. Drawback of this filter is the slow response time that amounts to the time needed for four pulses. But still it represents the fastest compromise for obtaining a good signal quality.

The calculation of the velocity was included in the firmware to guarantee smallest possible discretization errors. It was added in the file 'brickos/kernel/dsensor.c' which implements the sensor evaluation. The evaluation is triggered by interrupt handlers when a new pulse arrives. Thus the discretization accuracy corresponds to the accuracy of the system clock (1 ms).

Wireless signal transmission

Further extension to the firmware had to be done for speeding up the infrared/Bluetooth transmission of the encoder values.

For ensuring that for every evaluation of the controller a new encoder value is present, a minimum sample rate of 10 samples/s was assessed.

To estimate the sample time and as a first version of the later communication system a small JAVA application was written which uses the serial port interface provided by the pcrcxcomm package included in lejOS. Here the class josx.rcxcomm.Tower represents a simple interface to the hardware functionality which is available to initialize a serial connection via USB tower or one of the emulated serial ports. It also allows to read and write bytes of data and some basic configuration and error handling functions.

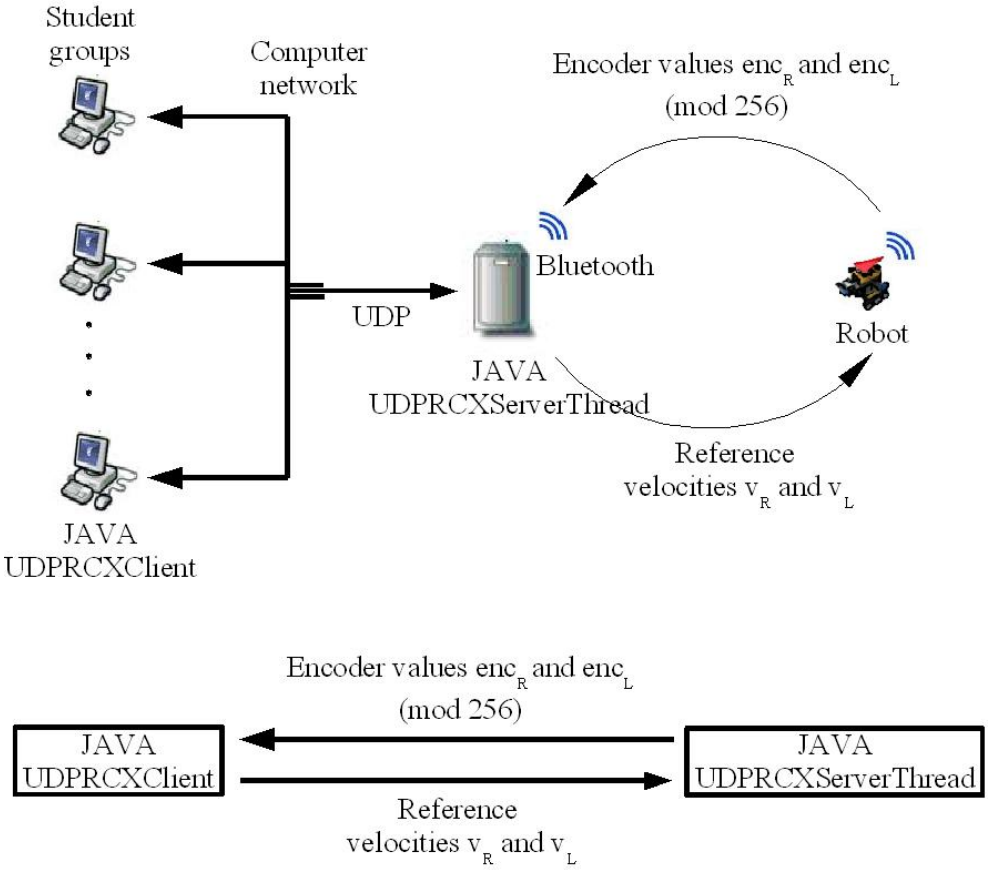


Figure 3.5: Bluetooth communication scheme. The two encoder values are sent via Bluetooth to a PC running the java server class which relays them to the laboratory PCs.

Firstly a synchronous communications scheme was used in combination with the USB tower. In this application two threads are running independently on the PC, one each for reading incoming data and writing. On the RCX an application is running which is sending the encoder values via the infrared port after receiving a new packet from the PC. The PC's sender thread sends a packet when either a packet from the RCX was received or a timeout after 250ms occurred.
Thus it's assured that only one device at a time is sending data.
First tests with this system showed fatal results as almost no packet was received within the timeout interval. So as a next step it was tries to accelerate the transmission with the new protocol LCP described in chapter 2.4.4 due to its smaller packet length.

This measure already reduced the roundtrip time to approximately 250ms.
After searching the firmware implementation for further possibilities to accelerate the transmission process it became clear that unnecessarily high waiting intervals between the transmission of two bytes were responsible for most of the delays. In the standard configuration, whenever a byte is received by the IR interface the transmitter has to wait 4 times the time needed to send one byte before transmitting and 2 times after each byte transmitted. By reducing this constants to 1 and 0.5 the roundtrip time could be reduced to 120-170ms. As further decrease of this values lead to unstable transmission behaviour no further decrease was possible. The remaining transmission time consists of checksum coding and decoding, internal processes regarding the protocol handling and the infrared communication. Furthermore other system processes on RCX and PC affect the transmission time and lead to wide variations.
The obtained sample time still wasn't close to the objective of less than 100ms. Tests with the Bluetooth communication even showed that here the roundtrip time still were bigger than 150ms.

USB synch.	USB asynch.	BT synch.	BT asynch.
120-170 ms	40-70 ms	150-190 ms	50-90 ms

Table 3.2: Final roundtrip times.

So a new communication concept was tested using asynchronous communication. Again beginning with the USB tower an application was implemented where on each side a process is running that is constantly sending the corresponding data with small waiting intervals between. This is done to ensure enough computation time to other processes, which also reduces the statistical spread of the sample times. Data is received whenever there is a packet available. In the later control application data is only transmitted from PC to RCX once per controller sample after calculation of the new input. So the traffic is further reduced.
As the infrared connection is half-duplex (only one side can send at a time) this communi-

cation scheme only is possible because USB-tower and RCX internally handle it by collision detection and ensuring that the channel is free before sending, which is called hardware flow control. So anyhow the channel can be used only oneway but its capacity is used much better than with the synchronous mode.
However the Bluetooth device has no possibility to avoid collisions (the hardware realization described in 2.4.2 only translates incoming and outgoing signal from Bluetooth to infrared separate from each other, without regarding the other component's state).
According to this, tests for using the asynchronous communication mode with the Bluetooth device failed because of interfering signals from adaptor and RCX.
When using infrared transceivers, the problem of infrared echoes or cross transmissions always occurs. This can be handled in two ways. The receiver can be deactivated while the transmitter is sending, so no echoes will be read. If this is not possible due to missing hardware compatibility (for instance on the RCX), the same number of bytes that was sent has to be discarded (read echo). It has to be mentioned that this method only is possible if there surely is read an echo because of a cross transmission, for example. The RCX firmware uses this principle.
Because of the missing hardware flow control on the Bluetooth device the asynchronous communication leads to a mixture of both signals that can't be separated, otherwise it is the only possibility to reach the desired sample time.
The only way to allow this communication method in full-duplex is to separate one transmitter-receiver combination and so introduce a second infrared channel. Considering the two microprocessors responsible for the serial IR connection on RCX and Bluetooth adaptor, full-duplex is supported such that both channels can be used simultaneously.
To separate the two channels a nontransparent and completely enclosed tube was constructed around the adaptor's ir-emitter and the RCX's ir-receiver as shown in figure 3.6.
Furthermore the legOS firmware had to be altered as it is not supposed to allow reading and writing at the same time. Parts of the firmware had to be removed or changed inducing collision handling and prompting the transmitter to wait until incoming data is received.
Another problem occurred when setting the RCX to high intensity IR mode. Here few

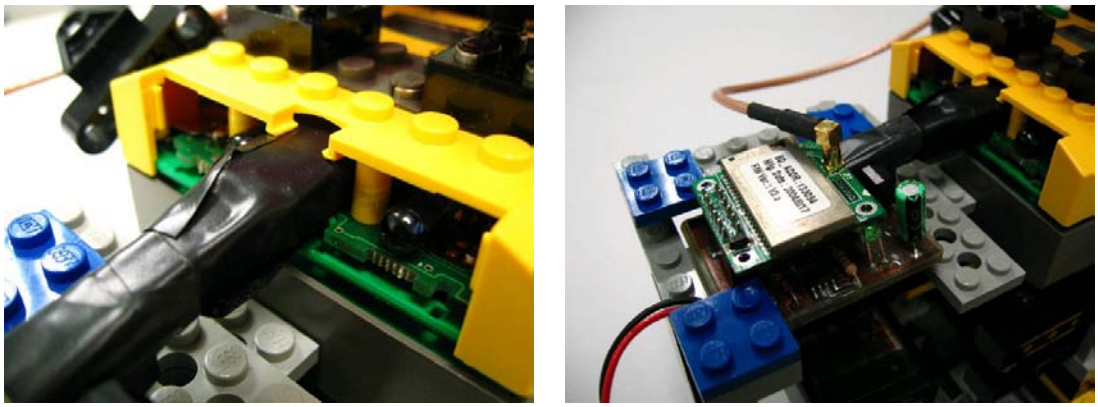


Figure 3.6: Tube constructed to separate the IR channels.

reflections of the RCX emitters are registered by the RCX receiver despite usage of the tube. Consequently when using the tube, the RCX has to be run at low infrared intensity level.

Using the described buildup the sample time could be decreased to the aspired length. Table 3.2 shows the resulting sample times using the tested communication setups and concepts. In figure 3.7 the corresponding sample time distributions can be found.

3.1.2 Communication scheme

Figure 3.5 shows the communication concept used for the encoder data. On the PC equipped with a Bluetooth device the JAVA class UDPRCXServer is running and continously listening for data coming via UDP (Ethernet) or Bluetooth. So reference values for track speeds coming from the controller over network are relayed to the vehicle whereas the encoder values a relayed to the JAVA class UDPRCXClient(s) running in MATLAB. This class provides the actual values to the controller running in MATLAB.

3.2 Position detection by webcam

The second system output is the vehicle's position which is needed for controller and observer. As will be shown in 5 position and state estimation only based on encoder

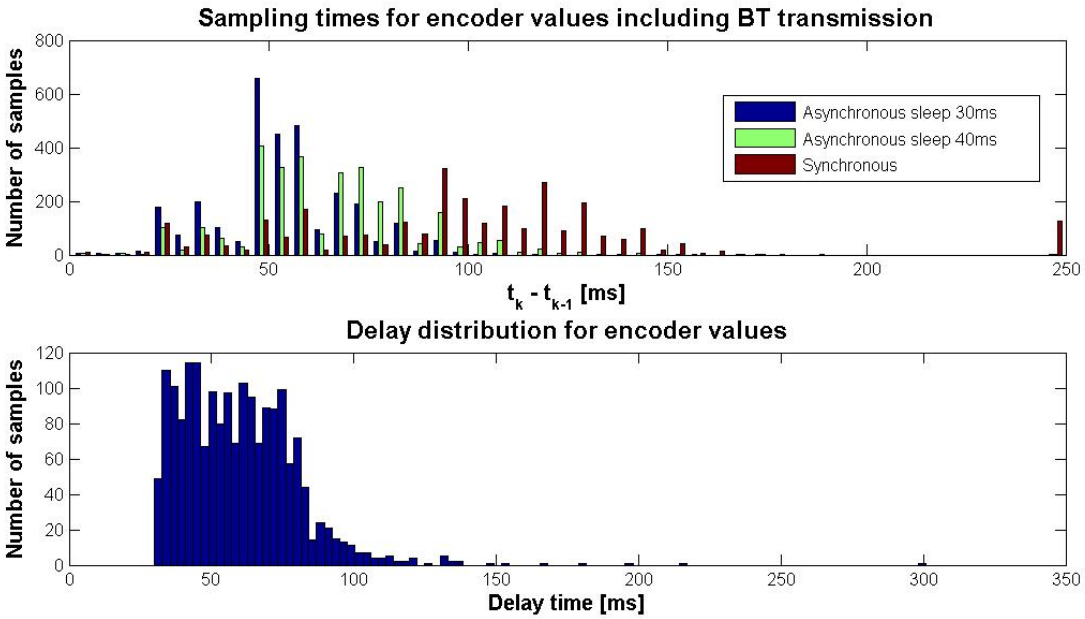


Figure 3.7: Distribution of sample times and delays for the encoder signal. The distribution of sample times (top) shows the intervals between two incoming sensor values while the distribution of delays (bottom) represents the actual delays at the moment of the controller action. Both were measured with a PID running simultaneously. The sample times were recorded for a synchronous and two asynchronous setups with a short pause after sending a sensor value.

signals rapidly generates significant errors, depending on the encoder quality and errors like slip and friction. So in most cases an additional measurement as for example the angle to a reference beacon or position data are used to enhance the estimate.
Therefore in the underlying experiment the position data is used and obtained by a webcam and subsequent image processing which will be discussed closer in the following section.

3.2.1 Image processing

Before looking at the concrete algorithms used to detect the vehicles position first a few basics terms in image processing will be introduced.

Image processing basics

Digital image processing is the alternation of images in bitmapped graphics format in order to enhance or transform them. Furthermore techniques to identify shades, colors ans relationships are used to attempt to understand the image and extract important information.
An image normally is represented by the image matrix S

$$S = (s(x,y)) = \begin{bmatrix} s(0,0) & \dots & s(0,n-1) \\ \vdots & \ddots & \vdots \\ s(m-1,0) & \dots & s(m-1,n-1) \end{bmatrix} \tag{3.2}$$

s(x,y) is the value of one quadratic image point and may be a single value or a vector, in case of composed images of several color layers. The numerical value can be of any type, but most common are unsigned integers of different precision (color depth).
Image processing normally is done in two steps. Firstly the raw image is processed to emphasize or reveal the desired image properties. Afterwards further passes for extracting the relevant information and reducing the amount of information are performed.

Operators

Mathematical functions working on a pre-defined region of the image matrix are called operators. They are used to alter the picture in some way and normally don't reduce the

information amount. Operators can be distinguished by the number of points used by the function into the following categories:

Local pixelwise operators

Local operators apply a function on each pixel value. Typical local operators are histogram modifications, such as contrast enhancement, binarization or histogram equalization.

Global operators

In contrary to local operators, global operators use information of every image pixel to generate the new value for each point. This kind of operation also is called transform and normally is found in form of:

$$s'(x,y) = \frac{1}{MN} \sum_{y=0}^{M-1} \sum_{x=0}^{N-1} f(s(x,y)) \tag{3.3}$$

Typical transforms are the 2-dimensional Fourier transform, the cosine transform and the wavelet transform for converting the image in its frequency domain. Another important transform is the Hough transform that can be used to find lines or other basic geometric objects.

Regional operators

Regional operators only use a defined subset of pixels surrounding the position the function is evaluated for. This subset is called neighbourhood and normally is a square or disc of fix dimensions. Regional operators often have the following form:

$$s'(\hat{x},\hat{y}) = \frac{1}{ab} \sum_{y=\hat{y}-\frac{a}{2}}^{\hat{y}+\frac{a}{2}} \sum_{x=\hat{x}-\frac{b}{2}}^{\hat{x}+\frac{b}{2}} f(s(x,y) \tag{3.4}$$

A special issue concerning this kind of operators are the boundary regions which have to be treated different. When using a function in form of equation 3.4 x and y can be outside the image boundaries. Here several techniques exist, as setting outer values fixed or zero, or performing a modulo on the index.

This kind of operators are often called filters. Well-known filters are high-,low- and band passes which work comparable to their 1-dimensional analogons in signal processing. Another important filter is the median filter that is used for noise reduction without losing sharpness. Further for detection of gradients differentiating filters are used, which can be orientation specific.

A non-linear class of filters working on binary images are morphological operations. Here the following four different types can be distinguished:

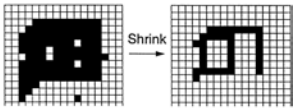
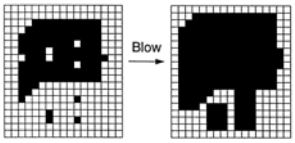
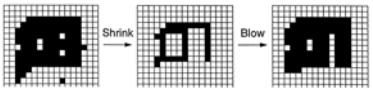
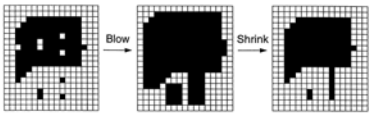
Erosion		Logical AND of s(x,y) and its neighbours. Also called 'shrink'.
Dilatation		Logical OR of s(x,y) and its neighbours. Also called 'blow'.
Open		Erosion + dilatation; eliminates dots, closes white lines and smooths peninsulas
Close		Dilatation + erosion; eliminates holes, closes black lines and smooths bays

Table 3.3: Morphological operations.

Post processing passes

After preprocessing the image, further passes can be done to analyze the image and extract the relevant information, mostly in form of statistical analysis or recognizing objects within the image.

Edge extraction

Detection of edges is an important method for object recognition and tracking as edges separate objects from each other and the background.
Here edge means a steep gradient in gray scale, color , texture or other significant properties. Thus edges normally can't be extracted directly from a picture but need preprocessing in form of differentiation filtering. A typically sequence for edge extraction is

1. Filtering (differentiation)
2. Edge detection (thresholding)
3. Edge localization / following
4. Edge grouping

By following edges surrounding an object it is possible to determine its boundaries. Additionally description of the object boundaries can be done independent from its orientation using direction coding, which is very helpful for tracking objects.
Edges also directly can be the objects of interest, as for example streets, electric wires in integrated circuits or as indication for pattern orientation.

Segmentation

Dividing an image into different regions as for example 'object' or 'background' is called segmentation. This is done to find the contours of objects so they can be analyzed for their properties (as size, shape, etc.) and cut from the background.
This can be done as follows:

Edge based

by searching closed edge sequences that surround an object

Area based

by binarizing the image with a fixed or adaptive threshold. Areas of connected 1's build an object and neighbourhood relations define which pixels are connected and

if two very close areas of 1's belong to the same object or not. Typically are the 8-neighbourhood $\begin{pmatrix} X & X & X \\ X & o & X \\ X & X & X \end{pmatrix}$ and the 4-neighbourhood $\begin{pmatrix} & X & \\ X & o & X \\ & X & \end{pmatrix}$

Motion based

is used for processing image sequences. By building a difference picture, objects can be distinguished from the fix background by their movement.

Attribute extraction

Based on the preprocessed or segmented picture, the image can now be analyzed for contour attributes (object related) or content attributes (object or image related).

Contour attributes Contour attributes can be properties belonging to the object-contour (e.g. size, perimeter or bounding box), properties of the objects position (e.g. center of gravity, orientation) or properties describing the objects form such as compactness, central moments or the edge direction sequence as result of direction coding.

Content attributes Other than contour attributes, content attributes describe the content of an image or object in form of stochastic data (e.g. histogram, histogram moments, correlations or spectra) or texture related values as edge directions and frequencies or grain size distribution.

As previously mentioned these attributes can belong to single pixels, objects or a complete image. By describing the attribute by a value an attribute map can be defined, written as m(x,y) for pixels (also called attribute image), m(o) for objects or m/m(k) for images (m(k) in case of image sequences).

Classification

The previously discussed attribute extraction returns an amount {m} of values for the

corresponding items (pixels, objects or images) for that an frequency distribution p(m) can be calculated.

By joining multiple attributes $m_1, ..., m_n$ to a vector attribute $\mathbf{m}$, $\mathbf{m}$ defines a single point in an n-dimensional attribute space, whereas $\{\mathbf{m}\}$ provides an n-dimensional frequency distribution (fig. 3.8).

Building subspaces (e.g. ellipsoids) of this attribute space can be used to distinguish different classes of objects, as shown in figure 3.9. Whenever two classes share all attributes (here classes 3 and 4) additional attributes have to be found to separate them.

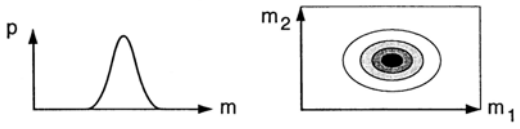


Figure 3.8: Example for a 1- and 2-dimensional attribute distribution. The attribute value m_i is evaluated for a large number of objects which delivers the shown distribution (left). Performed for a second attribute can be visualized in topographic form (right). (Source [39])

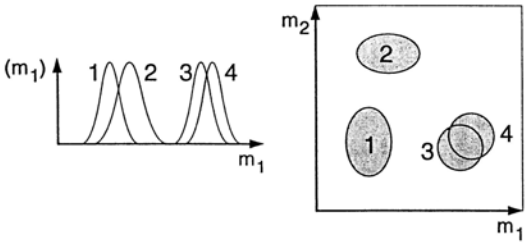


Figure 3.9: Classification example. Using multidimensional attribute distributions can lead to easily separable classes which overlay when only one attribute is regarded separate. (Source [39])

Marker detection

After introducing the basic terms of digital image processing we now can turn on the methods used for detecting the LEGO vehicle's marker.

Sun Mircosystems offers a large variety of APIs (Application programming interfaces) for JAVA. Among other things there also exists an 'advanced imaging library' for image processing. But as this library doesn't contain any classes for segmentation, object detection and color filtering, own algorithms had to be developed.

Since the marker area builds a very clear contrast to the surrounding areas due to its color,

contour based algorithms were used in this case. For differentiating pixels belonging to the marker from background pixels, the following two filters were developed to generate an attribute value from the RGB image.

Color distance filter

When dealing with unicolored objects, such as the used marker, all points belonging to this object will have a similar color, varying slightly because of different shadings. So the euclidian distance (in normalized RGB space) between the color of a point and a reference color can propose a good property for distinguishing object points from the background. This can be done as follows:

$$s'(x,y) = \sqrt{(r(x,y) - r_R(x,y))^2 + (g(x,y) - g_R(x,y))^2 + (b(x,y) - b_R(x,y))^2} \quad (3.5)$$

with r,g and b being the normalized color values calculated by:

$$c(x,y) = \frac{C(x,y)}{\frac{1}{3}(R(x,y) + G(x,y) + B(x,y))} \quad (3.6)$$

Normalization enhances the robustness towards changes in ambient light as it works with the ratio between component brightness and mean brightness. This value differs less than the absolute values.

When binarizing the image by applying a threshold, all points which posses a color within a sphere around the reference color will be determined as object points. The radius of the sphere is defined by the threshold.

Dominant color filter

When using a marker colored in one of the three basic colors red, green or blue, pixels belonging to the marker will have a much higher value in this component (here red) than in the remaining ones. So a weighted difference in form of

$$s'(x,y) = R(x,y) - a_1 G(x,y) - a_2 B(x,y) \quad (3.7)$$

is building a good property for identification of the marker.

Both values have to be scaled to the interval 0...255 for simplifying the binarization. The color distance filter for instance returns for each pixel a floating point value between 0 and $\sqrt{18}$ which can be scaled to 0...255 linear or logarithmic.
The color distance filter works very robust when adjusted right and can be easily adopted to a new environment or ambient light conditions by redefining the reference color.
The dominant color filter also works robust and shows low sensitivity to changing illumination. Furthermore it is extremely fast and approximately 2-3 times faster than the color distance filter (here ∼30ms for a 240x320 pixel image, including object detection and property calculation).
As the ambient conditions were constant over the whole period of the thesis it was decided to use the dominant color filter due to its better computational efficiency and its good results in early experiments.
After generating an attribute image with the filter, the image is transformed into a binary image by thresholding. In case of the color distance filter all pixels smaller than the threshold are defined as true, for the other filter all pixels with a bigger value belong to the object.
For reducing the effects of image noise that may create single false positive (in background) or false negative pixels (within the object) a sequence of 'closings' was used.

Object labelling and calculation of properties

After preprocessing and binarizing the image, the next step is to find the vehicle's marker and calculate its contour attributes. As the application also should be able to detect multiple objects all connected areas are analyzed and therefore have to be labelled. Furthermore image noise and false positive pixels (mostly from the floor) can produce small areas of 1's (so-called blobs = binary large objects, meaning connected areas of 1's) such that a detection algorithm has to identify and label all blobs. Afterwards further filtering can be done, for instance regarding the object size.

Connected component labelling

Connected component labelling works on binary images and different measures of connectivity neighbourhoods can be used (see 3.2.1). Here 8-connectivity is used. The connected components labelling operator scans the image by moving along a row until it comes to a point p for which s(x,y)=1. There it examines the four neighbours of p which have already been encountered in the scan (i.e. the neighbours (i) to the left of p, (ii) above it, and (iii and iv) the two upper diagonal terms). Based on this information, the labelling of p is done as follows:

1. If all four neighbours are 0, assign a new label to p, else
2. if only one neighbour has V=1, assign its label to p, else
3. if one or more of the neighbours have V=1, assign one of the labels to p and make a note of the equivalences.

After completing the scan, the equivalent label pairs are sorted into equivalence classes and a unique label is assigned to each class. Finally a second scan is performed through the image, where each label is replaced by the label assigned to its equivalence class. This label corresponds to the objects index.[9]

Contour attributes

Based on the labelled binary image as a next step for each object the following contour attributes are calculated and saved in a data structure.

Center of gravity

The center of gravity can be obtained by dividing the image moments of first order by the number of object pixels.

$$x_c = \frac{1}{N} M_x = \frac{1}{N} \sum_{i=1}^{N} x_i, \quad y_c = \frac{1}{N} M_y = \frac{1}{N} \sum_{i=1}^{N} y_i \tag{3.8}$$

Bounding box

The bounding box represents the smallest rectangle (with orientation along image axes)

including the complete object and is described by the upper left and lower right corner.

$$bb = ((\min(x_i), \min(y_i)), (\max(x_i), \max(y_i))) \tag{3.9}$$

Orientation by image moments

By calculating the image moments of second order

$$M_{xx} = \frac{1}{N}\sum_{i=1}^{N} x_i^2, \quad M_{yy} = \frac{1}{N}\sum_{i=1}^{N} y_i^2, \quad M_{xy} = \frac{1}{N}\sum_{i=1}^{N} x_i y_i \tag{3.10}$$

and using the following help variables

$$a = \frac{M_{xx}}{N} - x_c^2, \quad b = \frac{M_{xy}}{N} - x_c y_c \quad c = \frac{M_{yy}}{N} - y_c^2 \tag{3.11}$$

we obtain

$$\theta = \frac{arctan(b, (a-c))}{2}, L_{1/2} = \sqrt{6(a + c \pm \sqrt{b^2 + (a-c)^2})}, \tag{3.12}$$

where θ is the orientation (0° - 180°) and $L_{/2}$ are the dimensions of an equivalent rectangle with the same moments as the object.

As the orientation only is within the range of 0 and π, the absolute orientation of the vehicle can't be determined. To provide this information, the form of the marker was chosen to be an isosceles rectangle with significantly larger height than base length. Using this special form, the pixel with the maximum distance to the center of gravity is determined within the pixels belonging to the marker, additionally to the previously discussed attributes. This point is also saved as attribute since the vector between the marker center and this point directly delivers the vehicles orientation.

3.2.2 Camera calibration

The algorithms described in the previous section only delivers the gravity center of the vehicle in image coordinates, which is a raw value that has to be processed further. Regarding the parallel alignment of the two planes, image plane (CCD-chip) and floor, we can characterize the transformation as a orthogonal projection. Thus the resulting affine

transformation is composed of scaling and translation, and can be described by a 3x3 matrix (considering only x and y coordinates).[21] This transformation matrix can be found by calibration of the camera. But before discussing the calibration algorithms, the camera buildup shall be treated first, as fixation and adjustment of the camera build an important part to reduce systematic errors in position detection.

Installation and alignment

For simplification and higher precision it was decided to fixate the camera in a position perpendicular to the ground. For first tests and development it was situated at an altitude of 2 m, for the final experiments directly under the ceiling at 2.60 m to provide a larger operation range for the vehicle. The area observed in this position is 194.0 x 145.5 cm.

Figure 3.10: Fastening of the webcam. It was attached on a wooden board and hung up directly below the ceiling.

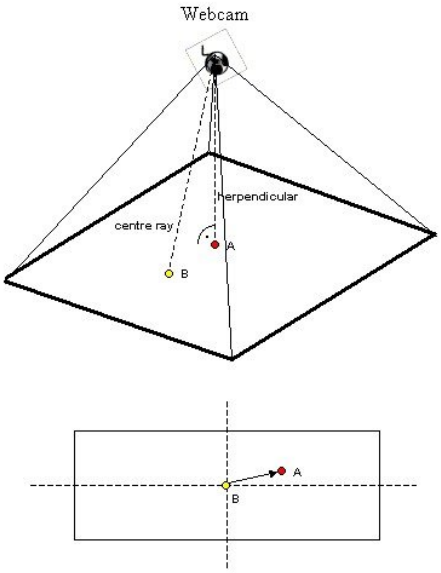


Figure 3.11: Camera adjustment: The camera position has to be altered until image center and perpendicular point match.

The complete setup and the fixation are shown in 3.10 and 3.11. The spherical shape of the Logitec webcam complicates a precise adjustment. Even in the constructed holding

attached on a levelled board it has two degrees of freedom that can generate perspective errors. To guarantee the best possible adjustment, a perpendicular was fixated at the position of the lens. The point where the perpendicular touches the ground is the vertical projection of the lens center. If the camera is adjusted right, it has to be exactly in the center of the image, assuming a symmetric alignment of the CCD chip to the lens center. The perpendicular's end point on the ground was marked with a cross so the camera could be turned until the marked point was exactly in the middle of the image.

Calibration theory

After guaranteeing smallest possible perspective errors by correct adjustment it now is possible to calculate the transformation matrix.
Therefore normally some defined markings with known coordinates (x,y) are placed on the ground. By obtaining the corresponding coordinates in the image (pixel index) the matrix can be calculated. As one point gives 3 equations (one for each coordinate), but the matrix delivers 9 unknowns at least 3 points are needed. Considering the special structure of an affine transformation matrix ([33])

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} w & z & 1 \end{bmatrix} T = \begin{bmatrix} w & z & 1 \end{bmatrix} \begin{bmatrix} t_1 & t2 & 0 \\ -t2 & t1 & 0 \\ t3 & t4 & 1 \end{bmatrix} \tag{3.13}$$

, or in transposed form

$$\mathbf{x} = \mathbf{T}\mathbf{z} \qquad \mathbf{T} = \begin{bmatrix} t1 & t2 & t3 \\ -t2 & t1 & t4 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

at least 4 equations would be sufficient for the four unknowns.
To obtain a higher robustness towards effects of measurement and systematic errors, the common method is to use a highly over-determined set of equations and search a set of parameters that minimize the error in a certain way. A very well-known method which was used here, is the least-square-approximation. It returns the set of parameters that have

the smallest error in quadratic sense, meaning

$$\sum_{i=1}^{N}(x - x_i)^2 = min \tag{3.15}$$

The least-square-approximation of $\mathbf{T}$ can be obtained by

$$\begin{bmatrix} x_{r,1} & \dots & x_{r,n} \\ y_{r,1} & \dots & y_{r,n} \\ 1 & \dots & 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} x_{p,1} & \dots & x_{p,1} \\ y_{p,1} & \dots & y_{p,1} \\ 1 & \dots & 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{y} = \mathbf{Tx} \tag{3.16}$$

$$\mathbf{T} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \tag{3.17}$$

where $(x_{r,i}, y_{r,i})$ are the real-world coordinates and $(x_{p,i}, y_{p,i})$ the coordinates within the picture.

The third component of the coordinate vector is set to 1 for simplicity reasons. This value may be chosen freely but is suggested not to be zero for avoiding numerical problems during the matrix inversion.

Transformation matrices $\mathbf{T}$ obtained in calibration tests showed exactly the structure described in eq. 3.13. This shows that methodic errors by neglecting intrinsic camera parameters and numerical errors are sufficiently small.

Calculation of camera altitude

Another important parameter is the camera altitude which is needed for correcting the perspective errors caused by the vertical marker position.

Therefore the angle of beam spread has to be determined with the buildup shown in figure 3.14 where a_1 and a_2 are either vertical or horizontal vision range. For two different camera altitudes the field of vision covered by the camera was measured (tab. 3.4) to determine the outer boundary beams.

Using the following relation defined by the theorem on intersecting lines for the scheme in figure 3.14

$$\frac{0.5a_1}{h_1 + x} = \frac{0.5a_2}{h_2 + x} \tag{3.18}$$

	hor. range	vert. range
h=11.5cm	10.2cm	7.7cm
h=6.2cm	6.575cm	4.95cm

Table 3.4: Measurements of vision field at different camera altitudes h.

the distance x between camera container and virtual intersection of beams can be determined to

$$x = \frac{\frac{2h_1}{a_1} - \frac{2h_2}{a_2}}{\frac{2}{a_2} - \frac{2}{a_1}} = \frac{a_2 h_1 - a_1 h_2}{a1 - a2} \tag{3.19}$$

and for the angle of beam spread follows:

$$\beta = 2 \arctan(\frac{0.5 a_1}{h_1 + x}) \tag{3.20}$$

For the Logitech webcam a mean value of $\beta = 37.7°$ and $x = 3.4cm$ was calculated.

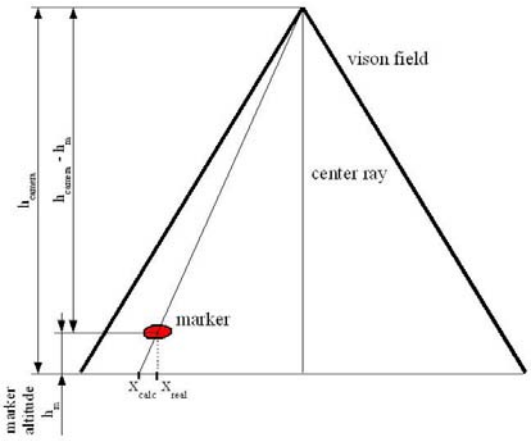


Figure 3.12: Correction of position measurement due to perspective error. The transformation resulting from camera calibration is valid only for ground level. Due to the marker altitude a perspective correction has to be performed.

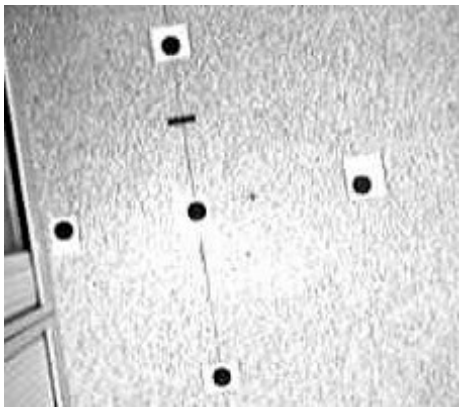


Figure 3.13: Example for a calibration pattern used in this work.

Position correction

As the camera calibration is done for points on the floor but the marker is posed on a different level (here 14cm above the ground), using the marker position directly produces an error in position estimate due to the perspective view. This error is zero if the vehicle is located directly in the center of the image, otherwise it is proportional to the distance from the center.
A correction can be done using the theorem on intersecting lines as shown in figure 3.12. By using the dependency

$$\frac{h_{camera}}{h_{camera} - h_m} = \frac{x_{calc}}{x_{real}} = \frac{x_{real} + e}{x_{real}} = \frac{1}{k} \tag{3.21}$$

for each dimension the projected image position can be obtained by

$$\begin{bmatrix} \hat{x}_p \\ \hat{x}_p \end{bmatrix} = \begin{bmatrix} (x_p - 160)k \\ (y_p - 120)k \end{bmatrix} + \begin{bmatrix} 160 \\ 120 \end{bmatrix} \tag{3.22}$$

This correction could be in included directly into the transformation matrix **T** by also using the z coordinate and thus expanding it to a 3-dimensional transform. But this would require more complex calibration patterns and is less accurate due to numerical problems during matrix inversion. So for applications where the camera is situated parallel to the floor the presented method builds a more stable and easier solution.

Calibration GUI

For simplification of the camera calibration process the MATLAB GUI shown in figure 3.15 was developed. It can use any 'black circles on bright background'-pattern (e.g. fig. 3.13) to calibrate the camera. It determines the centers of gravity of the black objects and using them as reference points. Therefore the real-world coordinates have to be provided in form of a structure shown in 3.16 which includes number, order size and coordinates of the pattern objects.

For calibrating the camera the following steps have to be taken:

1) Load pattern-data struct ('Load Pattern Data' button)

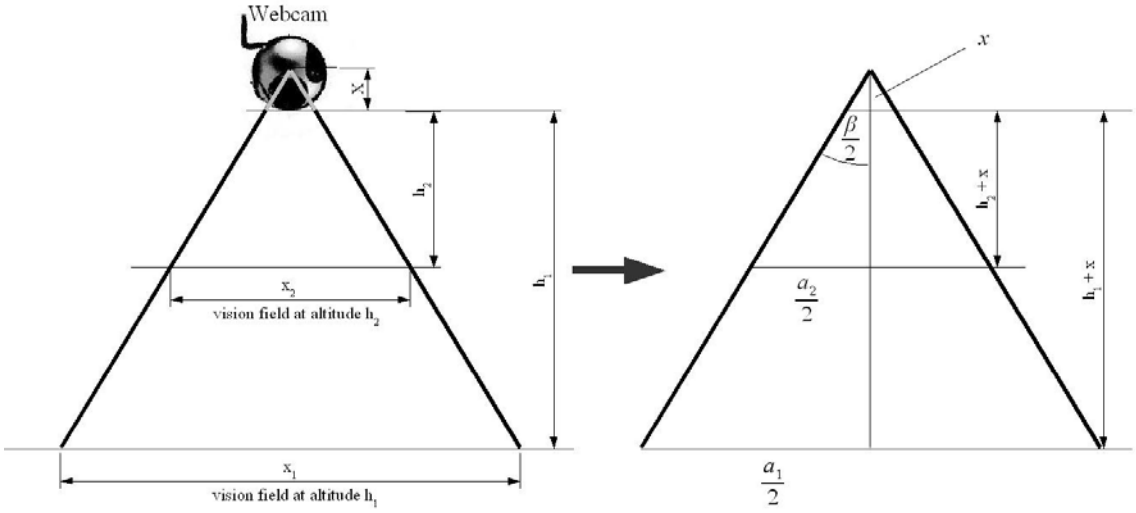


Figure 3.14: Determination of camera beam spread and altitude. By solving the equations obtained by the theorem on intersecting lines the distance x from lens to the virtual intersection point can be determined.

2) Grab image

The JAVA class UDPWebcamClient initializes a TCP/IP connection to UDPWebcamServer class running on the PC with webcam. It grabs an actual picture and transfers it via network.

The image is saved as global variable, converted into a gray scale image and binarized with the current threshold value.

3) Adjust threshold for best binarization result

By moving the slider the threshold value is changed and the result is visualized immediately (when the binary visualization is selected). The automatic threshold is calculated by the IP toolbox function graythresh using Otsu's method.

4) Mark pattern objects in right order

By pressing the mouse over the corresponding object a red cross will be set. Order and number of the objects has to be the same as in the data structure.

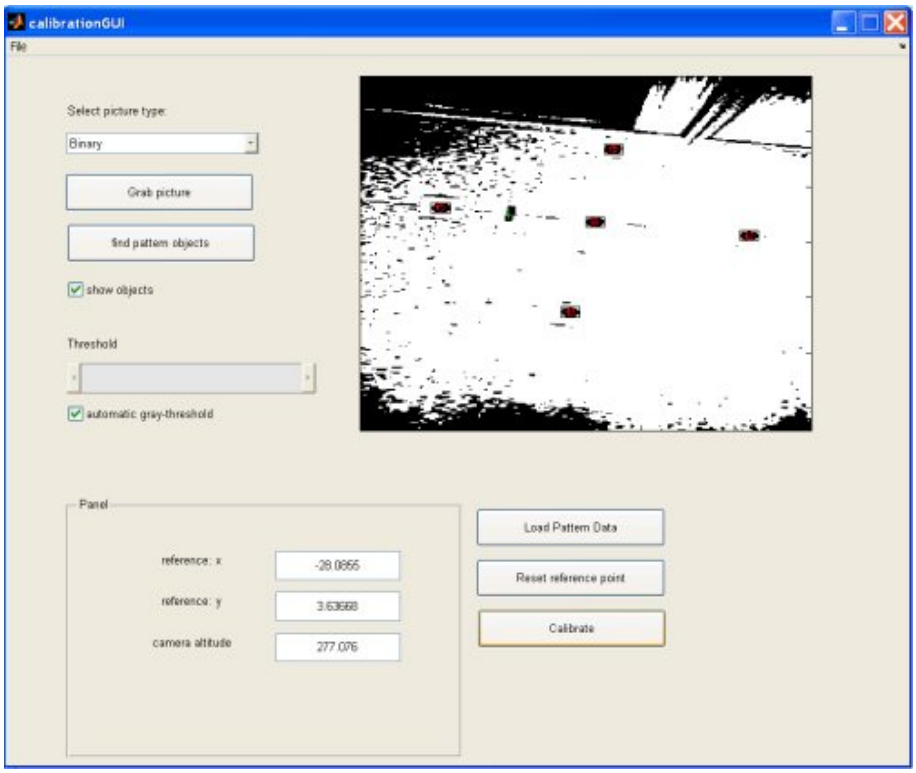


Figure 3.15: Front end of the Calibration GUI. It provides a simple and fast possibility to calibrate the webcam. Different patterns can be used.

5) Search pattern objects

By pressing the equally name button, a blob detection algorithm searches for the pattern discs. If activated, the bounding boxes can be displayed to see if the threshold has to be adjusted.

6) Mark a reference point if desired

Press the mouse one more time over the desired reference position. A green cross will appear.

7) Start calibration ('Calibrate' button)

After performing step 7 transformation matrix T and camera altitude will be calculated and stored as global variable. Furthermore the real-world coordinates of the reference point will be displayed.

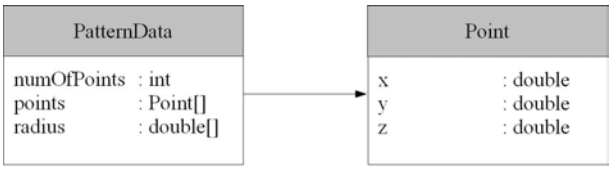


Figure 3.16: Data structure of calibration pattern data. It includes the number of pattern circles and for each circle its center position and radius in defined order.

3.2.3 Communication scheme

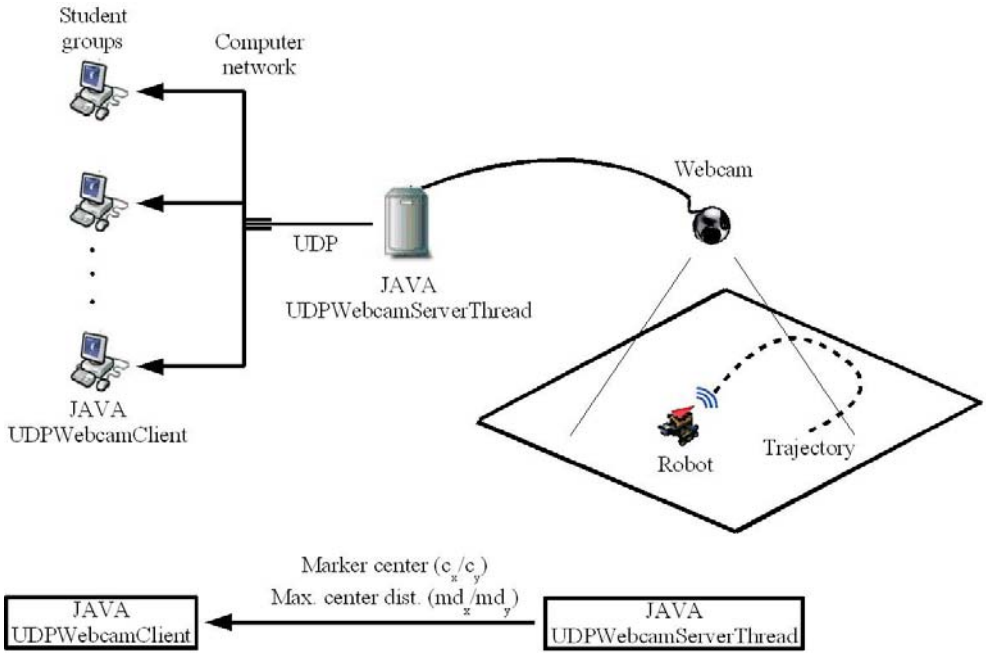


Figure 3.17: Webcam communication scheme. The images are continuously processed and the reduced data (marker's center of gravity and marker point with maximum distance to center) is sent via network to the PC(s) running the controller.

Figure 3.17 shows the communication concept used for the webcam data. On the PC equipped with webcam the JAVA class UDPWebcamServer is running and continuously processing image data. When a marker is found for each processed picture a defined dataset

is send via UDP (Ethernet) to the JAVA class UPDWebcamClient running in MATLAB. This class provides the actual values to the MATLAB controller. The dataset includes the marker's center of gravity and the coordinates of the marker point with maximum distance from the center. Due to the marker's form this point can be used to calculate the vehicle's orientation.

Image processing is followed by a short pause to save computing and network resources. This also has a positiv effect on the distribution of processing times. The longer the sleeping interval the smaller gets the distribution whereas the risk that long samples cross the timeout barrier rises. Processing time and resulting delay distribution are shown in figure 3.18.

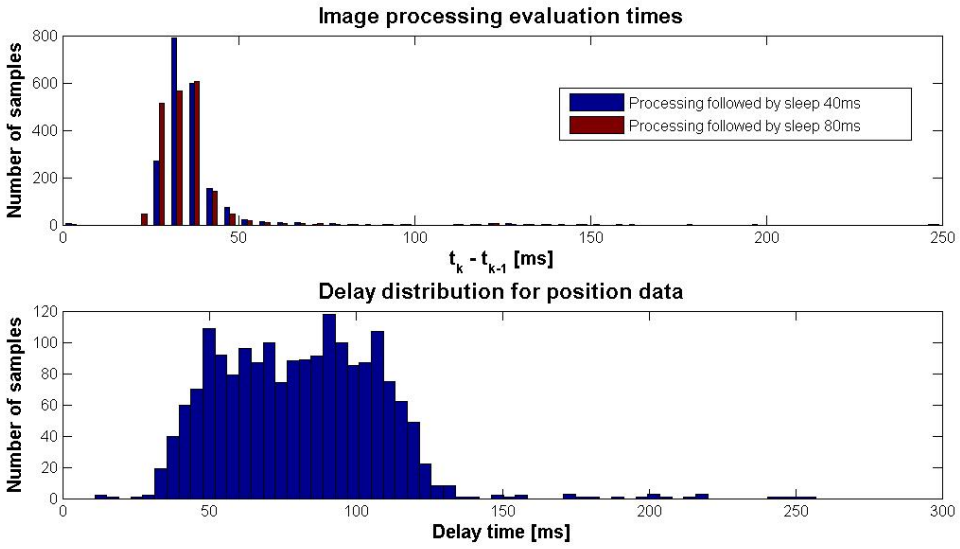


Figure 3.18: Distribution of evaluation times and delays for the encoder signal. The distribution of sample times (top) shows the intervals needed for the image processing routine while the distribution of delays (bottom) represents the actual delays at the moment of the controller action. Sample times were measured with two different pause durations.

3.3 Delay estimation and network synchronization

The measurements discussed in the previous sections only can be obtained with variable delays which are significant compared to the controller sample time.
They are induced by transmissions over ethernet and/or infrared/Bluetooth networks and computational delays. Furthermore the broad distribution of sample times showed in 3.7 and 3.18 are caused by using non-realtime systems on both sides. Especially the RCX firmware only provides a pseudo-multi-threading system where a process gets calculation time depending on its priority. If a high priority process has to do time consuming calculations without pauses between, lower priority threads won't run. For example, if the process of the PID controller gets the focus it will complete all calculations (about 16-20ms) before passing the focus to the thread sending the sensor data, who's priority is lower. In addition background processes as sensor evaluation or incoming transmissions have influence on the processing time.
However the main reason for the broad distribution are the asynchronous processes which lead to shifting delays even in theoretic case of constant sample times.
For assuring synchronous processes with constant sample times and delays, the sampling time would have to be chosen extremely large which means a high loss of performance.
In this application the loss of performance would have been to high, so it was decided to use asynchronous processes. Yet a better knowledge of the delays can help enhancing the used observer (see chapter 5) and reduce their negative effects.

3.3.1 Clock synchronization

To determine the delays between measurement and controller evaluation the easiest way is to attach a timestamp to each measurement. The difference between this timestamp and the current time of the controller action is equal to the overall delay and includes all kind of transmission and computational delays.
Using this method on the same PC can be done straight forward. But in case of measurement and controller on different PC's within a computer network requires additional effort

as in general the system time on both PC's is different which leads to the problem of clock synchronization.
Since computer networks evolved very quickly in the last decades and nowadays connect billions of computers in the whole world, synchronizing the clocks of two or more computers became necessary. For actual financial applications and distributed databases running parallel on several PC's it is essential to reconstruct the temporal order of accomplished actions or perform tasks simultaneously.
In the 1980s Dave Mills of the University of Delaware therefore developed a protocol called NTP (Network Time Protocol) to synchronize the clocks of two connected PC's. Until today it is a common standard with high precision when dealing with variable transmission delays.[22]

3.3.2 Synchronization algorithm

NTP's basic functional principle to calculate transmission time and offset between the two system clocks was also used in this application, implemented in the JAVA class TimeSynchrThread. In fixed intervals and alternating order the two computers exchange an array containing three timestamps, timestamp of the previous sender's last outgoing, and timestamp of the current senders last incoming and outgoing. The receiving computer additionally remembers the arrival time of the current incoming packet. The scheme in in figure 3.19 shows this communication sequence and the 4 timestamps available for the computer that got the last transmission.
The timestamps on one line (representing one PC or it's time axis) are represented in the corresponding system time which is in general different from the second one.
Using these four timestamps and introducing the helper variables

$$a = t_{k-2} - t_{k-3} \tag{3.23}$$

$$b = t_{k-1} - t_k \tag{3.24}$$

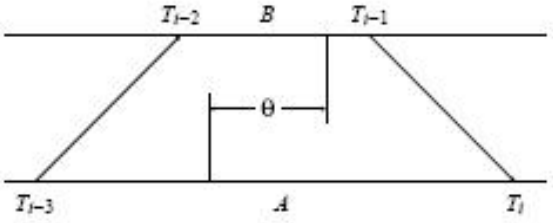


Figure 3.19: NTP scheme. Each horizontal line represents a computer clock running in its own system time. The four timestamps are needed to calculate offset θ and roundtrip time δ. (Source [35])

the offset θ and transmission time δ can be determined by

$$\theta = \frac{a+b}{2} \tag{3.25}$$

$$\delta = a-b \tag{3.26}$$

The advantage of this method is that each PC can calculate the current offset and transmission time for itself and there is no need for a client-server like system architecture. But there also exists a major drawback as the method assumes the transmission delays to be equal in both directions, which may not be the case in some applications, for instance when using a DSL connection.
However in the used application the network traffic was low and almost symmetric such that the assumption holds.

3.3.3 Overall communication scheme

The overall communication scheme for the laboratory setup is shown in figure 3.20. It combines the schemes showed in figures 3.5 and 3.17, supplemented by the clock synchronization. By using this concept the costs can be lowered considerably as only one webcam and one BT adaptor are needed. Compared to the Bluetooth connection the ethernet induced delays are small. In a 100MBit network with moderate traffic the roundtrip times lay between 1 and 5ms. Additionally clock synchronization provides the possibility to work with timestamps and estimate network delays.

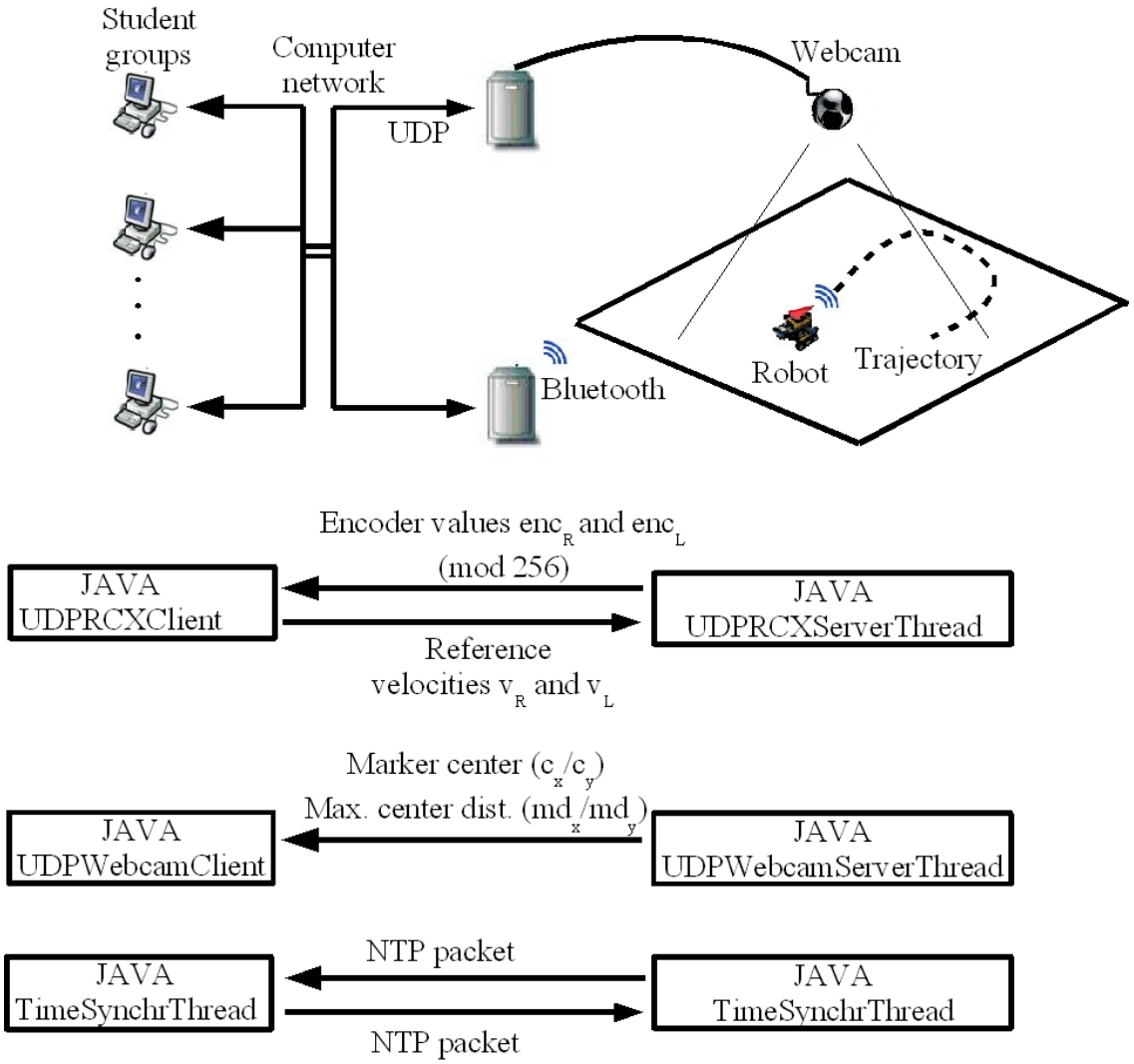


Figure 3.20: Complete laboratory communication setup. Two PCs are resposilbe for relaying measurements and the vehicle's reference track speeds, one each for the webcam and the Bluetooth adaptor. For better delays estimation a timestamp is attached to each data packet which is sent over the ethernet. Therefore additionally a small clock synchronization system is running in background (class TimeSynchrThread).

Chapter 4

Implementation of the RCX velocity controller

Using the kinetic model to control a vehicle is a very common method.[24] Thus the control problem is separated in two levels, a kinematic and a dynamic part. This corresponds to a cascade control, where the kinematic level represents the outer loop and the dynamic part the inner loop. Often for the kinematic controller a more complex controller type can be found whereas the much faster controller of the dynamics is a basic PI or PID controller. The same principle also is used in this application, with the difference that inner and outer controller are connected via network and Bluetooth. Directly on the RCX two PID controllers are used to control the track speeds using the velocity signal provided by the firmware (see chapter 3.1.1).

Due to the little computational power of the RCX the controller only could be realized with integer values what already takes 16-20ms for evaluation. A further challenge is given by the bad quality of the velocity signal provided by the LEGO rotational sensors.

Before discussing the details of controller design and parameterization, the dynamic parameters and characteristics of the system (the vehicle) will be analyzed.

4.1 Parameter identification and controller design

4.1.1 Stationary behaviour

Firstly the static behaviour of the vehicle shall be analyzed according to [11]. Therefore a speed/input characteristic has been recorded. For 11 different value of u (motor input) the resulting mean speed was measured in a test drive. As the motors are bidirectional with symmetric behaviour, the characteristic only was measure for positive values and mirrored in the negative half plane. The resulting curve is presented in figure 4.1. It shows a clear saturation behaviour for high inputs and a wide dead zone due to static friction.

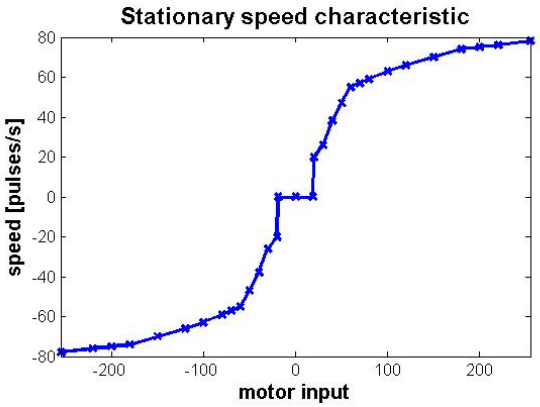


Figure 4.1: Static speed characteristic.

4.1.2 System dynamics

After analyzing the static input/output behaviour now the system dynamics can be regarded closer. A typical method to analyze local system dynamics is to measure the system's step response. The form of the step response often gives important insight in system structure and dynamic order and allows to estimate dynamic parameters. For instance if an overshoot occurs, the system order has to be two or higher.

Figure 4.2 shows the evaluated step response of the vehicle velocity. Neglecting the oscillations caused by the periodic friction the response shows a typical PT_1T_t form (fig.

4.6). However a major part of the time delay originates from the sensor and decreases for ascending velocity. The sensor dynamics are also part of the transfer function and so also have to be attended. In this case we can reduce the time delay slightly to the benefit of a slower system dynamic as the normal working point will be at velocities greater zero, where the delay is smaller.

Assuming a PT_1T_t-structure, the following parameters

$$k_p = \frac{y_s}{u_s}, \quad T = t_2 - t_1, \quad T_t = t_2 - T \tag{4.1}$$

can be extracted from figure 4.2

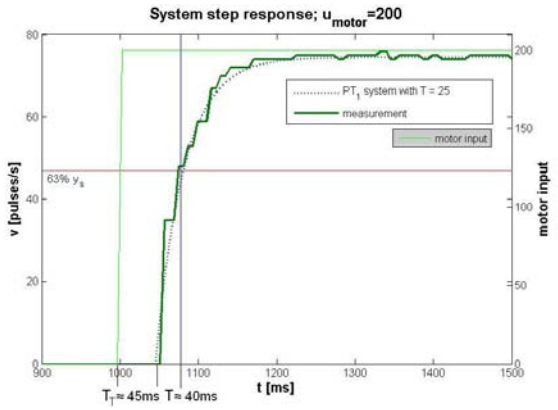


Figure 4.2: PT_1T_t approximation for the LEGO vehicle.

4.1.3 Motor and friction characteristic

To identify the non-linear part leading to the characteristic in 4.1.1 further measurements had to be done. As the sum of forces must be zero for constant velocity, in such a state friction force (of the complete system) and motor force are equal. Friction forces are difficult to measure in contrary to the motor force, which can be obtained by a relatively simple buildup (fig. 4.5). The resulting characteristic (fig. 4.3) indicates a linear dependency between the motor input and the force provided by the pulse width modulated motor. Using a linear least-square approximation of the motor force for every point of the

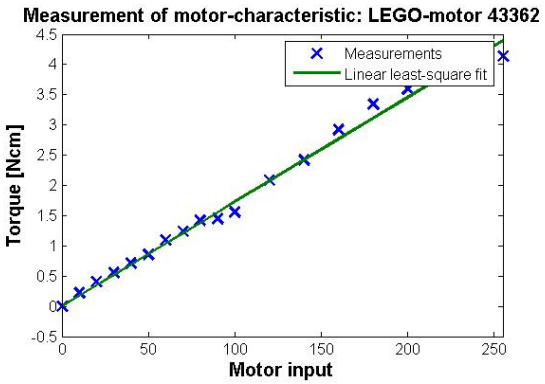


Figure 4.3: Torque characteristic of the motor as function of motor input.

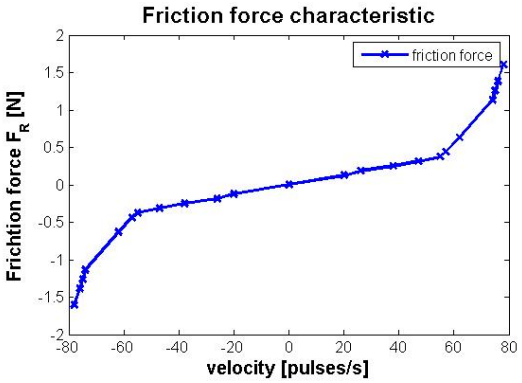


Figure 4.4: Friction characteristic as function of v.

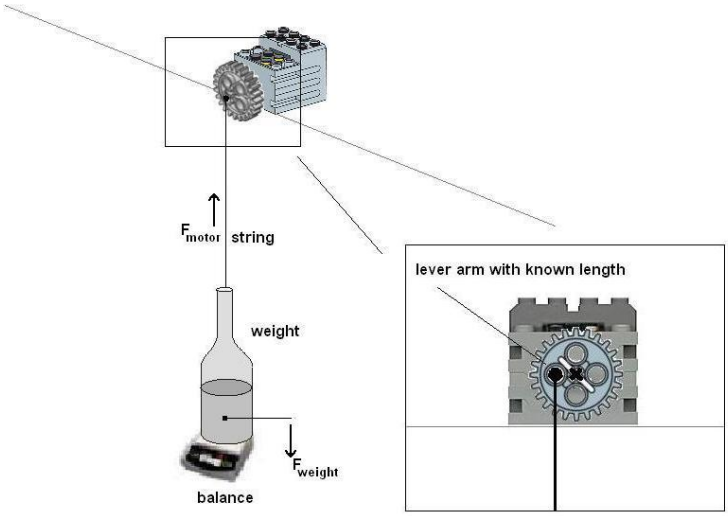


Figure 4.5: Measurement of the motor force characteristic. The motor is fixated on a table and a big cogwheel is attached to its axis. A light string pinned at a defined distance from the axis holds a weight lying on a balance. The string has to be adjusted such that the cogwheel position matches the position in the zoomed picture. Thus the lever arm length is defined. For measuring the characteristic the motor input value is increased in steps and the remaining weight is measured. The difference between measured weight and real weight corresponds to the motor force.

measured static velocity profile 4.1 the friction force can be calculated and related with the corresponding velocity. Thus the velocity/friction force characteristic showed in figure 4.4 can be determined. Based on the previously identified parameters now the mathematical models and controller design can be discussed closer.

4.2 PID design using Ziegler-Nichols method

Many systems having a deadbeat transfer function can be approximated by the following PT_1T_t-model:

$$G(s) \approx \hat{G}(s) = \frac{k_p}{Ts+1} e^{-sT_t} \tag{4.2}$$

The three model parameters k_p,T and T_t can be obtained by the step response as shown in figure 4.6 which shows the same characteristics as the step response of the LEGO vehicle.

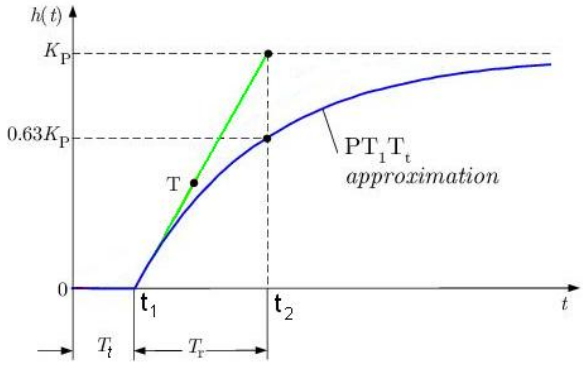


Figure 4.6: PT_1T_t approximation used for the Ziegler-Nichols method.

k	T	T_t
0.37	40 ms	45 ms

In case of relatively low performance demands on the control system, parameters for P, PI or PID controllers can be calculated by empirical rules, using the three previously mentioned system parameters.

Therefore the best known method was developed by Ziegler and Nichols. The calculation rules for the controller parameters are summarized in table 4.1. They were determined empirical such that the closed control loop will converge fast with a moderate overshoot.[34] Furthermore Ziegler and Nichols developed a second method by determining the critical proportional gain and period at the point, the system begins periodic oscillations. As this method is not applicable to the LEGO system it won't be discussed further.
To apply the Ziegler-Nichols methods the system must exist (or at least a good simulation model) and be available for the step response experiments.
Due to the non-linear motor characteristic presented in figure 4.1 parameter estimation turns out to be difficult as it will deliver different values for step responses of different magnitude (step size). The slowest response should result from a step to the maximum input value (u=255) which was slightly reduced (u=200) because of the friction oscillations. The resulting step response is shown in 4.2.
Therefore the parameters already were determined in 4.1 and the following controller parameters can be obtained:

$$k_p = 0.37, \quad T_i = 0.09s, \quad T_d = 0.022s$$

For discrete control systems the controller gains k_i and k_d depend on the sampling time T_s and can be calculated as follows.

$$k_i = k_s \frac{T_s}{T_i} \tag{4.3}$$

$$k_d = k_s \frac{T_d}{T_s} \tag{4.4}$$

4.3 Parameter tuning by simulation

Results with the parameters appraised by the Ziegler-Nichols method were very promising but the velocity still showed significant oscillations for low speeds and had to be improved.

C. type	k_p	T_i	T_d
P	$k_p = \frac{1}{k_s}\frac{T}{T_t}$		
PI	$k_p = \frac{0.9}{k_s}\frac{T}{T_t}$	$T_i = 3.33T_t$	
PID	$k_p = \frac{1.2}{k_s}\frac{T}{T_t}$	$T_i = 2T_t$	$T_d = 0.5T_t$

Table 4.1: Adjustment rules for controller parameters by Ziegler and Nichols. (Source [34])

Here obviously the errors caused by the non-linear friction characteristic, periodic friction by the gear and especially the bad quality of the velocity sensor are the reason for the system to be outside the validity area of this rule.
So the parameters had to be adjusted empirical and by hand, using the Ziegler-Nichols proposal as starting point.
Testing parameter sets in experiments is very time consuming. First the RCX has to be programmed, then the run is performed and at last the logged data has to be transmitted via the slow IR connection using the USB tower or Bluetooth adaptor. So the idea came up to use a simulation to find better parameters what works much faster and thus allows a larger range to be tested. So only promising parameter sets have to be testes on the real system. In the following section model and experimental results of this approach will be presented.

4.3.1 Model

Due to the relatively long sample time, the controller was tested with (30-70ms), and a first attempt to use a discrete model, it became obvious that the resulting discretization was too unprecise for sensor and drive model and often caused numeric instabilities. Thus it was decided to use a hybrid model, discrete for the PID controller and continuous for the remaining parts. This modelling approach turned out to be much more stable and reflected the measured effects.

Vehicle submodel

The most difficult modelling part including most of the model error is given by the drives. Here for simplification, both motors and tracks are reduced to one resulting force in the vehicles direction such that no steering is possible. This generalization neglects cross effects but should be adequate for identification of controller parameters.
The vehicle's movement is modelled using Newton's laws of motion. Acceleration is determined by the resulting force on the vehicle and its mass. By integrating the acceleration the velocity is obtained.
The resulting force represents the difference between force emanating from the motors and the counteracting friction force.
Identification of these forces was treated in 4.1.3. For integration in the model, motor force was approximated by a linear function whereas the friction characteristic from 4.1.3 was used as a lookup table with linear interpolation between the points of the characteristic. Outside the interval the characteristic is defined on, no extrapolation is performed.
Furthermore a simple error model is included trying to describe the observer oscillation of the vehicles speed by periodic friction. The friction appears to result from the gear, so its period is proportional to the velocity v. Figure 4.7 shows a velocity/frequency diagram for the measured points, including a linear least-square approximation. The resulting dependency is

$$f(v) = 0.0632v \quad [\frac{Hz}{pulses/s}\, pulses/s] \tag{4.5}$$

This corresponds to one error period every 16 pulses (one motor rotation), what lets conclude that this error is caused by the inner motor gear.
By using the magnitude of the velocity oscillations, the oscillation of the friction force can be determined form the friction characteristic for the analyzed points. Although the oscillations get worse for low speed the magnitude of the friction fluctuation seems to get smaller, as shown in figure 4.8 in a quadratic sense. Thus a quadratic least-square approximation was used to describe the velocity dependence of this magnitude.

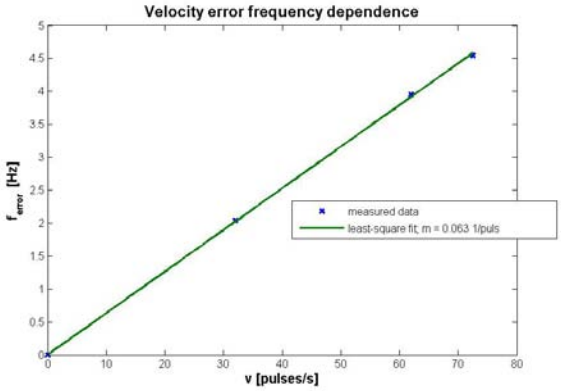


Figure 4.7: Linear approximation for error frequency.

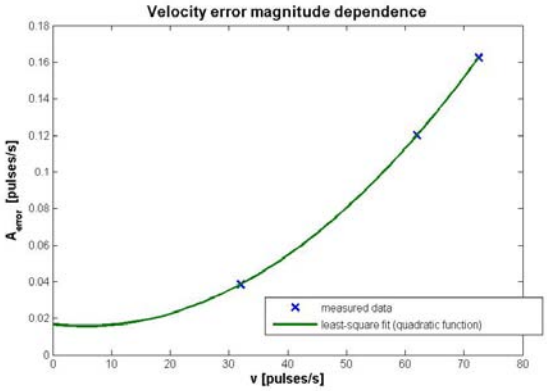


Figure 4.8: Quadratic approximation for error magnitude.

The value for the friction force returned by the lookup table summed with the periodic superposition builds the overall friction force. The complete model structure can be seen in figure 4.9. The s-function finally sums up all forced and returns the resulting force to the vehicle. Moreover it treats the special case of static friction in case of $v = 0$ such that the resulting force f can be written as:

$$f_{res} = \begin{cases} 0 & v = 0 \wedge \sum f < f_{static} \\ f_{motor} - f_{fric} & v \neq 0 \vee \sum f > f_{static} \end{cases} \tag{4.6}$$

Finally the speed obtained by the motion equations has to be converted to the desired unit of $pulses/s$.

Sensor submodel

The encoder used to determine the rotational velocities only updates this value when a new pulse is measured. This highly non-linear effect gets more severe the smaller the velocity is and is assumed to be the main reason for the bad controller performance. Therefore it has to be modelled. Moreover the systematic error described in chapter 3.1.1 deteriorates this problem and has to be included in the model too.

Both can be done by imitating the function principle of the velocity calculation on the RCX,

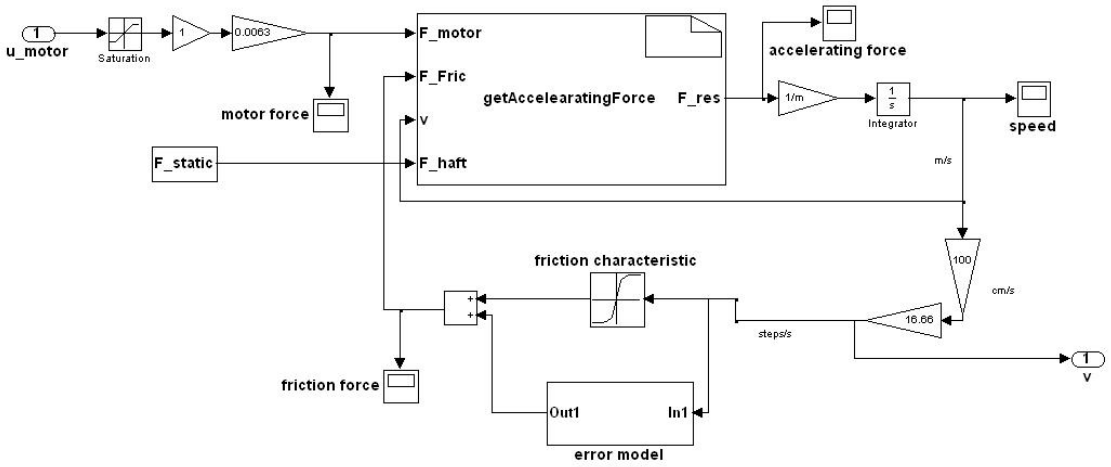


Figure 4.9: Simulink model for the vehicle speed.

meaning to use the same pulses as the firmware does. As can be seen in the Simulink plan (fig. 4.10), therefore a periodic signal with input proportional frequency is utilized which triggers pulses at fix points within the period. This signal can be generated by combining an integrator who's output is connected with a sinus function. If the integrator input increases the frequency of the sinus will rise proportional as time 'runs faster'. Triggering a pulse at defined angles in the sinus will create the same behaviour as the rotational sensor, even the unequal ratio of the interval sizes listed in table 3.1.

The pulses are analyzed by a s-function imitating the firmware. It is used to memorize the timestamps of the last pulses and calculate the velocity (equ. 3.1) when a new pulse occurs. Also the timeout is incorporated here.

The complete Simulink plan can be seen in figure 4.10. It shows the integrator connected to the sinus function by a gain block to adjust the frequency. The sinus output is connected with four hit crossing detectors, each corresponding to one of the intervals' pulses. Evaluation is done by the s-function. The feedback loop only is used by the s-function to determine if the next integrator step is entered as it may be triggered several times within one interval, depending on the used integrator.

The s-function also has to mind the case of a pulse lasting two intervals because hit crossing

detection may keep the value true for two consecutive samples if the input takes exactly the value of the threshold.
It has to be mentioned that the described model is only valid for positiv velocities. However for the desired objective to find better parameters by simulation, negative velocities are not required and would make the model unnecessary complex.

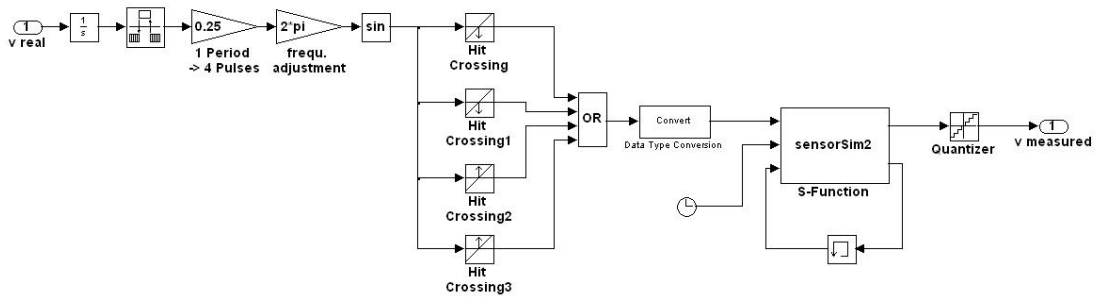


Figure 4.10: Simulink model for velocity measurement.

Full model

Based on the previously discussed submodels, the complete simulation model combines them and the control loop is closed by a discrete PID controller as can be seen in figure 4.12. The vehicles output (its velocity) is connected to the sensor who generates the measured velocity. The PID controller is inserted surrounded by a rate transition to imitate the time discrete implementation with the sample time used for the controller on the RCX.
Normally a discrete PID controller can be written as follows:

$$u(t_k) = k_p e(t_k) + \frac{k_p T_s}{T_i} \sum_{i=0}^{k} e(t_i) + \frac{k_p T_i}{T_d}(e(t_k) - e(t_{k-1})) + u_0 \tag{4.7}$$

By using the equation for the previous controller output

$$u(t_{k-1}) = k_p e(t_{k-1}) + \frac{k_p T_s}{T_i} \sum_{i=0}^{k-1} e(t_i) + \frac{k_p T_i}{T_d}[e(t_{k-1}) - e(t_{k-2})] + u_0 \tag{4.8}$$

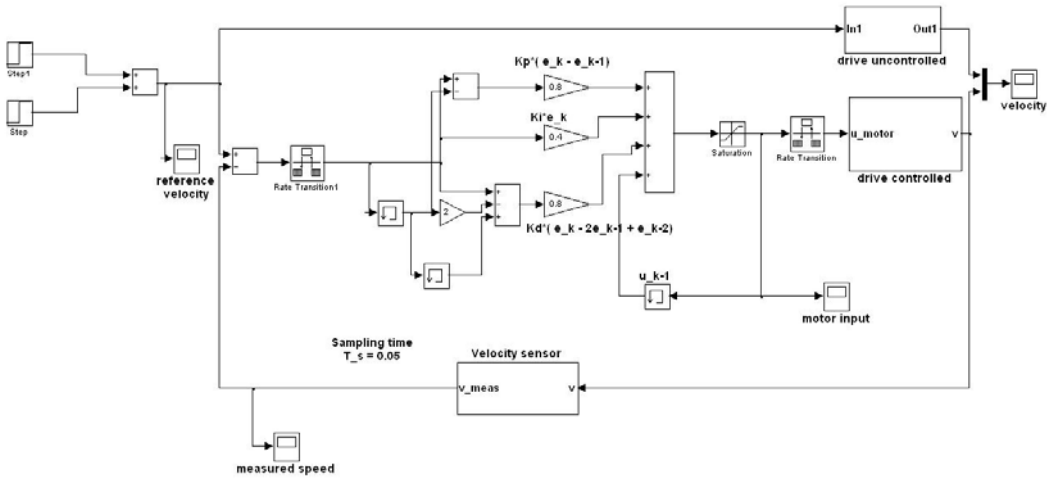


Figure 4.11: Simulink plan for the complete system.

and building the difference 4.7-4.8, the following differential form (also called velocity form [1]) can be obtained:

$$u(t_k) = u(t_{k-1}) + k_p[e(t_k) - e(t_{k-1})] + \frac{k_p T_s}{T_i} e(t_k) + \frac{k_p T_i}{T_d}(e(t_k) - 2e(t_{k-1}) + e(t_{k-2})) \quad (4.9)$$

Both structures are equivalent. As shown in the Simulink plan, the velocity form was used for the controller.

4.3.2 Results

The model showed in figure 4.12 was tested with several controller parameter sets to find alternative sets that can handle the bad signal quality. It is shown that for the parameters proposed by the Ziegler-Nichols method the controller performs relatively well-tempered but much too slow. During more experiments it became obvious that a faster reaction only can be reached by higher oscillations for low speeds. Figure 4.12 shows two alternative parameter sets with much higher integrator and derivative gains which perform satisfactory. The oscillations observed in this simulations turned out to be much smaller in the real system and must be a result of the quadratic error magnitude approximation.

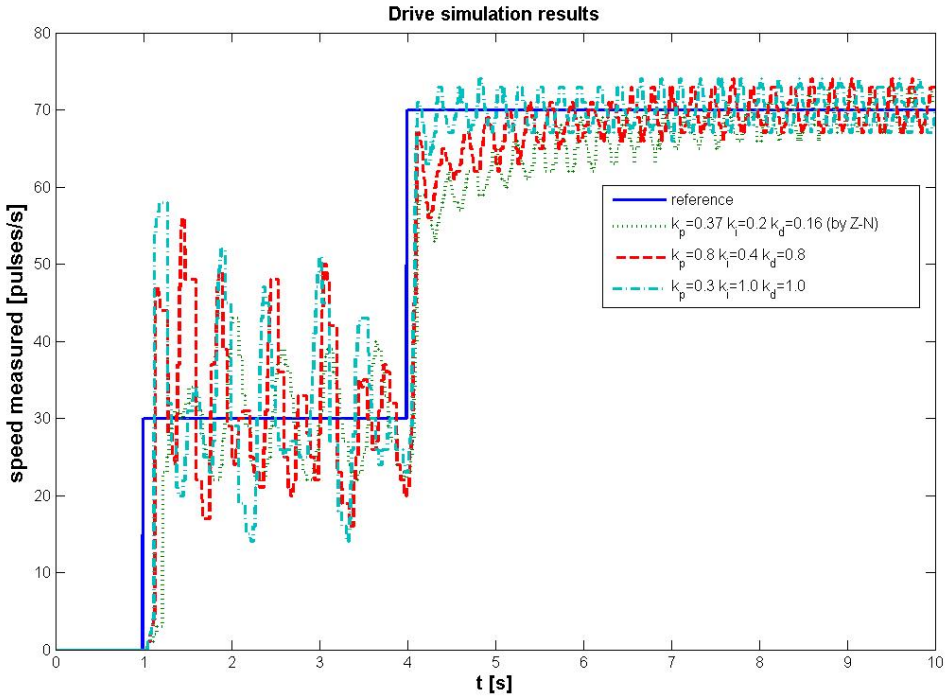


Figure 4.12: Simulation results for different parameter sets. For the parameters proposed by the Ziegler-Nichols-method the system is converging very slow.

4.4 Controller implementation and results

Controller timing

Achieving a correct sampling time on the RCX turned out to be problematic when using the multi-process structure (one thread for the controller, a second for sending sensor data). Within the controller thread `msleep()` became unprecise over long intervals and often returned to late (supposable due to other processes in charge). An empty loop running until the desired system time is reached needs to much computation time and slows down or completely interrupts other processes. So it was tested to use a loop only performing a 1ms sleep which lead to much better results but often awakes 1-2ms to late. Finally a compromise was made for exact timing: Waiting is performed in 2 steps, one loop performing a 4ms sleep until $t_{desired} - t \leq 4ms$ and afterwards an empty loop. In this way exact timing was achieved.

Punishment term

Analyzing the velocity profiles achieved with the PID controller, it stroke that especially in case of low velocity the downwards deviations are more severe than the upwards deviations. This can be explained by the friction which is in an intermediate region between static and dynamic friction for low velocities. Here the reaction of the controller is too slow. Thus an additional punishment term p was introduced. If the error gets above a certain threshold (here 8 pulses/s, only if too slow) the punishment term is added to the motor input. Because of the used differential form of the PID it is important not to add p to u_t (see eq. 4.9) since it is added to the integral part then. The final form of the controller can be seen in figure 4.13.

Result on the real system

Supported by the controller parameters obtained in simulation and with the Ziegler-Nichols method the controller was implemented on the RCX. Some results for two different sampling times T_S can be seen in figure 4.13. Sample times smaller than 50ms turned out to be critical since in case of low speed the controller may run several times until a velocity measurement update is available. Interestingly the high derivative gain (compared to

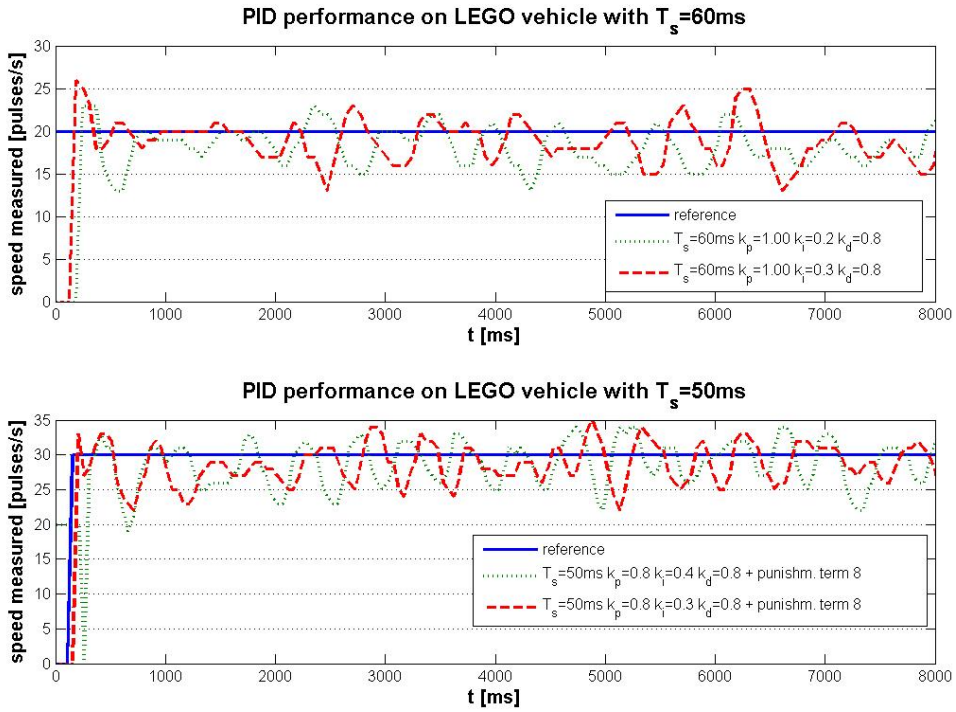


Figure 4.13: Experimental results on the LEGO vehicle.

Ziegler-Nichols) is not destabilizing the system much here and dampens the oscillations caused by the integral part of the control input. This gain has to be chosen high for achieving fast convergence. Here the proposed gain by Ziegler-Nichols turned out to be too small.

In the upper plot the mean value of the oscillating velocity seems smaller than the reference value. This is caused by the non-linear behaviour of the motor characteristic in this region. Therefore the previously discussed punishment term was used in the experiment shown in the lower plot. Here the deviation of the mean value appears less albeit it is not totally compensated.

The presented results show that the quality of the velocity control is not very high. But due to the quality of the sensor signal and the system friction no further enhancement could be obtained. Therefore the outer control loop has to compensate these errors.

Chapter 5

Vehicle model and observer design

The previous chapters discussed experiment buildup and the inner control loop. Furthermore in chapter 3 the sensor readings provided for the controller were treated. These readings include a major part of the system state (position (x,y)) and the encoder values as independent states. As we will see in chapter 7 this readings are not sufficient and further states are needed by the controllers. Especially the vehicles orientation (heading), velocity and angular velocity are important here. Since these values can't be measured directly, an observer was designed to estimate them.
Several different approaches of observer design for non-linear system can be found in literature (e.g. [29]). Regarding the noise of the sensors and the relatively simple system structure, the best choice appear to be an Extended Kalman Filter (EKF) as described in [29] and [20]. Due to its stochastic part it works as observer and filter at the same time. Design, parameterization and validation of this observer will be treated in the following sections, after introduction of two models used for validation.

5.1 Evaluation models

For validation of the EKF two models with different complexity were used. Furthermore the different measurement and transmission delays were incorporated to evaluate the observer performance under conditions that are very close to real.

In chapter 8.1 the following models also will be used for simulations of the closed loop system.

5.1.1 Ideal model

The more simple model only consists of the kinematic model (as in [11]), complemented by two additional integrators representing the encoders.

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \end{bmatrix} = \begin{bmatrix} \cos(x_3)\frac{v_R+v_L}{2} \\ \sin(x_3)\frac{v_R+v_L}{2} \\ \frac{v_R-v_L}{2r} \\ v_R \\ v_L \end{bmatrix} \tag{5.1}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} v_R \\ v_L \end{bmatrix} \tag{5.2}$$

This model represents the ideal system without any errors. Velocity changes immediately when the input changes (no dynamics) and slip effects are neglected. These normally causes a gap between the state of the encoders and the really covered distance which is less.

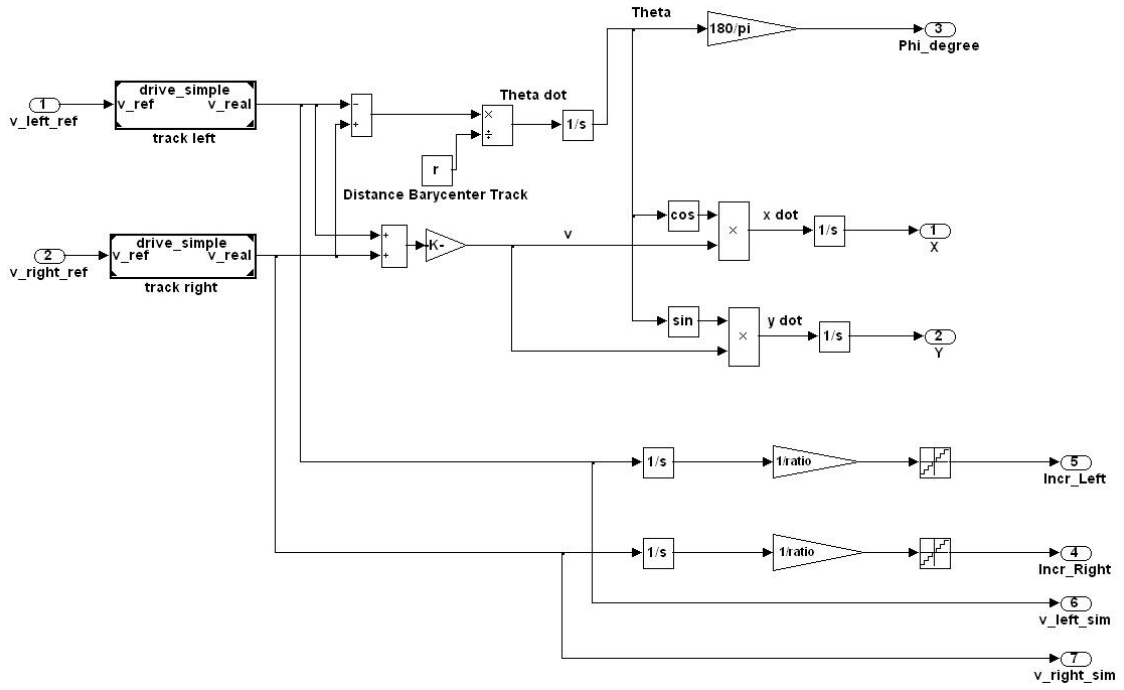


Figure 5.1: Ideal model for a tracked vehicle.

5.1.2 Extended model including errors

To test the EKFs under more realistic conditions a second model has been developed, containing some of the identified errors as slip, process noise and the periodic velocity oscillations described in 4.3.1. It has the same basic structure as the previous model but additionally includes two linear PT_1 subsystems for the tracks. For them also periodic speed oscillations are modelled similar to 4.3.1 with a velocity-dependent frequency and a quadratic dependency between v and the oscillation magnitude. Furthermore a slight white noise is added to the output, representing effects within gears and motors.

Slip is modelled proportional to the velocity difference of the two tracks. It is subtracted from the track with higher speed and added to the slower one such that a compensation occurs. Furthermore on both sides a slight white noise representing friction effects is added. The error modelling was meant to describe the effects qualitatively to make the simulation more real, thus all parameters were determined empirical.

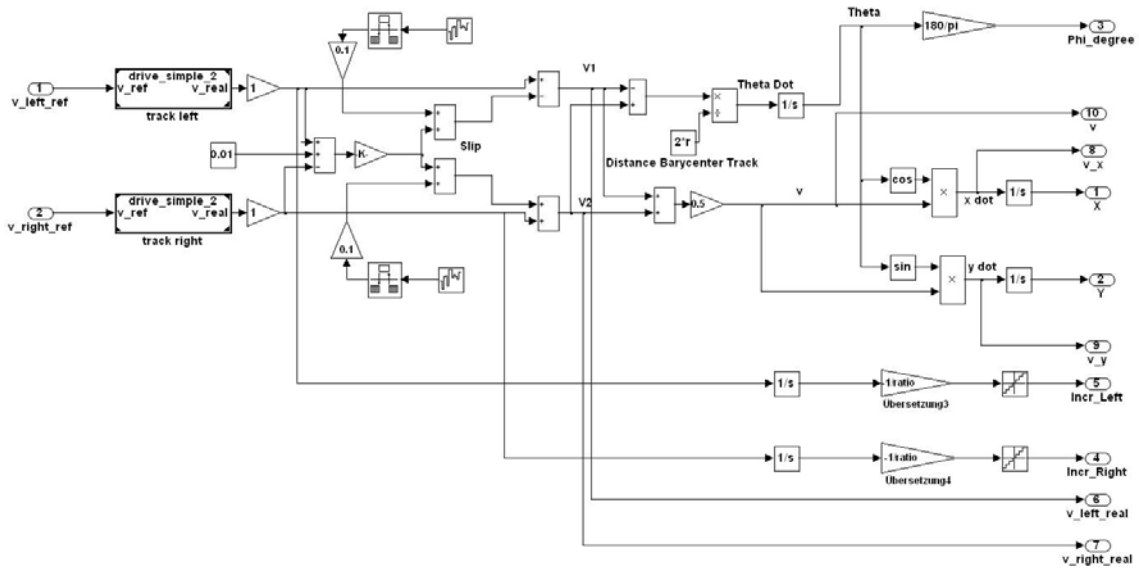


Figure 5.2: Extended vehicle model. Track speeds are modelled as PT_1 subsystems. Slip effects and noise represent friction errors.

5.1.3 Transmission modelling

To analyze the effects of transmission delays on the observers' performance, two different models for transmission delays were designed. Both models were included in one Simulink subsystem (fig. 5.3) where input 3 is used to switch between the two delay types.

Constant delays

Constant delays can be modelled directly by a Simulink block called 'Transport Delay'. Here a 'Variable Transport Delay' was used to allow adjustment of the delay value via an input. By adding Rate Transitions (see fig. 5.3) the discrete behaviour of the measurements can be imitated.

Stochastic random delays

A lot more complex to realize are random delays. Here a s-function was developed, supporting three types of delay time distributions:

1. rectangular distribution with adjustable mean value and interval width
2. Gaussian distribution with adjustable mean value and variance
3. Gaussian distribution with adjustable mean value and variance, but additionally a lower boundary such that delays can't get smaller than a minimum value or negative

For each sample the s-function generates a new delay by using the distributions. The current input value (the value to transmit) is saved and set as output after the delay time passed.
By identifying the corresponding parameters from delay distributions measured in experiments, a realistic transmission behaviour can be achieved.

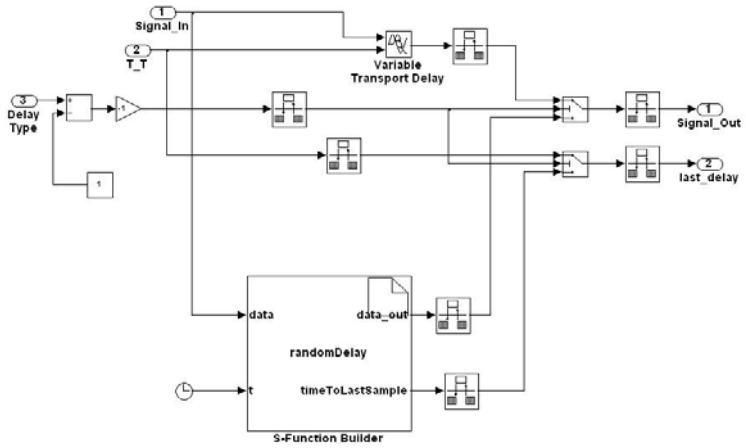


Figure 5.3: Simulink transmission delay model. Two types of delays are possible: constant delays and random delays with various distributions.

5.2 The Extended Kalman Filter

First published in 1960 by R.E. Kalman, the Kalman filter provides a recursive solution to discrete-data linear filtering problems. Due to increasing computational power, in the following decades the KF became subject of extensive research and a standard technique used in control engineering and signal processing today. Further extensions on non-linear

systems, for instance, were following.

The Kalman filter is a set of mathematical equations that provides an efficient computational tool to estimate the state of a process, in a way that minimizes the mean squared error. By allowing to estimate past,present and even future states also if the precise nature of the process is unknown, the KF represents a powerful solution in state estimation.

5.2.1 Discrete formulation of the EKF

Process

In the following we assume the system of interest to be formulated in the non-linear difference and measurement equations

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}); \qquad z_k = h(x_k, v_k) \tag{5.3}$$

where $x \in \mathcal{R}^n$ is the state vector and $z \in \mathcal{R}^m$ the measurements vector. w_k, v_k represent process and measurement noises at time k which have a independent, white, normal probability distribution with mean zero. The exakt values of w_k and v_k at time k are normally unknown.

With w_k and v_k unknown, state and measurements can be approximated by

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0); \qquad \hat{z}_k^- = h(\hat{x}_k^-, 0) \tag{5.4}$$

with $\hat{x}_{k-1}$ being the a posteriori state estimate from a previous time step k-1.

It is important to note that in contrary to the linear case the random variables v and w no longer are normal distributed after a non-linear transformation. This is the mayor drawback of the EKF. Some attempts were done by Julier et al. in developing a variation of the EKF, using methods that preserve the normal distributions throughout the non-linear transformation[19].

Computational realization

For using the filter equations from the linear Kalman filter formulation we define a new

system that linearizes 5.3:

$$x_k \approx \hat{x}_k^- + A(x_{k-1} - \hat{x}_{k-1}) + Ww_{k-1} \tag{5.5}$$

$$z_k \approx \hat{z}_k^- + H(x_k - \hat{x}_k^-) + Vv_k \tag{5.6}$$

with $\hat{x}_k^-, \hat{z}_k^-$ being the estimates from 5.4 and the following system matrices:

$$A_k = \frac{\partial f}{\partial x}|_{\hat{x}_{k-1},u_{k-1},0} \tag{5.7}$$

$$W_k = \frac{\partial f}{\partial w}|_{\hat{x}_{k-1},u_{k-1},0} \tag{5.8}$$

$$H_k = \frac{\partial h}{\partial x}|_{\hat{x}_k^-,0} \tag{5.9}$$

$$V_k = \frac{\partial h}{\partial v}|_{\hat{x}_k^-,0} \tag{5.10}$$

The Kalman filter, like most other observers, uses the so-called innovation or residual $z_k - h(\hat{x}_k^-, 0)$ which reflects the discrepancy between measurement and predicted measurement to update the a priori estimated state. Here the gain matrix K is determined by a probabilistic approach using error and noise covariances.
Therefore we can now define the a priori and a posteriori estimate errors as

$$e_k^- = x_k - \hat{x}_k^- \quad and \quad e_k = x_k - \hat{x}_k \tag{5.11}$$

The a priori and a posteriori estimate of the error covariance are then

$$P_k^- = E[e_k^- e_k^{-T}] \quad and \quad P_k = E[e_k e_k^T]. \tag{5.12}$$

Using this covariances in addition to the measurement noise covariances R, the gain proposed by Kalman is calculated by

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \tag{5.13}$$

K represents the gain that minimizes the a posteriori error covariance. See [20] for further explanation of how minimization can be proofed, the probabilistic background and further literature.
Looking at the properties of equation 5.13 some interesting observations can be made. The

more the error covariances R approach zero the heavier get the weights of K caused by the inversion.

$$\lim_{R_k \to 0} K_k = H^{-1} \tag{5.14}$$

Otherwise, as P_k^- approaches zero, the gain K gets smaller with the limit

$$\lim_{P_k \to 0} K_k = 0 \tag{5.15}$$

Using these preliminaries and the 'linear' system formulation from equation 5.5 we get the complete predictor-corrector-algorithm of the EKF. It is performed in two steps by:

1: Time Update (Prediction)

Project the state ahead:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0); \qquad \hat{z}_k^- = h(\hat{x}_k^-, v_k) \tag{5.16}$$

Project the error covariance ahead:

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \tag{5.17}$$

2: Measurement Update (Correction)

Compute the Kalman gain

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \tag{5.18}$$

Update estimate with measurements

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0)) \tag{5.19}$$

Update error covariance

$$P_k = (I - K_k H_k) P_k^- \tag{5.20}$$

where Q is process noise covariance and P_0, x_0 have to be chosen properly.
An important feature of the EKF is that in equation 5.18 H_k is used to correctly propagate only the relevant component of the measurement information to a certain state.
Here R and Q are also indexed by k which means that they may change with time. However in most applications they are used as constant values.

5.2.2 Plant models

Model 1: Basic System, encoders only

State

$$\begin{aligned} x_{k+1} &= x_k + T\cos(\phi_k)v_k + T\cos(phi)w_{1,k} \\ y_{k+1} &= y_k + T\sin(\phi)v_k + T\sin(\phi_k)w_{1,k} \\ \phi_{k+1} &= \phi_k + T\omega_k + Tw_{2,k} \\ EncL_{k+1} &= EncL_k + T(v_k - r\omega_k) + Tw_{1,k} - Trw_{2,k} \\ EncR_{k+1} &= EncR_k + T(v_k + r\omega_k) + Tw_{1,k} + Trw_{2,k} \end{aligned} \tag{5.21}$$

Input

$$\mathbf{u} = \begin{pmatrix} v_k \\ \omega_k \end{pmatrix}$$

Output

$$\mathbf{y} = \begin{pmatrix} EncL \\ EncR \end{pmatrix}$$

This model only uses kinematics and the encoder states corresponding to equations 5.1 and 5.2. As the measurements contain no information about angle ϕ the initial error for ϕ has to be very small.
It works good for the ideal model, but for the real system or the extended model the errors sum up and lead to an increasing deviation with ongoing time.

Model 2: Basic model using all measurements

State:

$$\begin{aligned} x_{k+1} &= x_k + T\cos(\phi_k)v_k + T\cos(\phi_k)w_{1,k} \\ y_{k+1} &= y_k + T\sin(\phi_k)v_k + T\sin(\phi_k)w_{1,k} \\ \phi_{k+1} &= \phi_k + T\omega_k + Tw_{2,k} \\ EncL_{k+1} &= EncL_k + T(v_k - r\omega_k) + Tw_{1,k} - Trw_{2,k} \\ EncR_{k+1} &= EncR_k + T(v_k + r\omega_k) + Tw_{1,k} + Trw_{2,k} \end{aligned} \tag{5.22}$$

Input Output

$$\mathbf{u} = \begin{pmatrix} v_k \\ \omega_k \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} X \\ Y \\ EncL \\ EncR \end{pmatrix}$$

Unlike the previous model here two additional outputs are used, the x and y coordinates of the vehicles position (more precisely the center of gravity of the marker). Thus for $v \neq 0$ a correction for θ is possible.

However in simulations with the extended model unstable behaviour in form of varying severe static deviations from the correct state were observed. This has two reasons. Firstly the real system input (v_r, v_l or v,ω) differs from the reference values (inner control loop). This corresponds to process noise and can be only partly compensated by the EKF as its distribution normally has a mean different from zero. The second and more severe problem is that slip generates a gap between the velocity measured by the encoders and the real ground speed of the tracks. This gap leads to a contradiction between the measurements of encoders and position and produces oscillations in error covariances and the resulting correction gain.

This especially leads to severe error in the estimate for θ (see 5.2.3).

Model 3: Basic model with practical approach

State:

$$\begin{aligned} x_{k+1} &= x_k + T\cos(\phi_k)v_k + T\cos(\phi_k)w_{1,k} \\ y_{k+1} &= y_k + T\sin(\phi_k)v_k + T\sin(\phi_k)w_{1,k} \\ \phi_{k+1} &= \phi_k + T\omega_k + Tw_{2,k} \\ EncL_{k+1} &= EncL_k + T(v_k - r\omega_k) + Tw_{1,k} - Trw_{2,k} \\ EncR_{k+1} &= EncR_k + T(v_k + r\omega_k) + Tw_{1,k} + Trw_{2,k} \end{aligned} \tag{5.23}$$

Input Output

$$\begin{pmatrix} v_{R,k+1} = \frac{Enc_{R,k+1} - Enc_{R,k}}{T} \\ v_{L,k+1} = \frac{Enc_{L,k+1} - Enc_{L,k}}{T} \end{pmatrix} \Rightarrow \mathbf{u} = \begin{pmatrix} v_{k+1} \\ \omega_{k+1} \end{pmatrix} = \begin{pmatrix} \frac{v_{R,k+1} + v_{L,k+1}}{2} \\ \frac{v_{R,k+1} - v_{L,k+1}}{2r} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} X \\ Y \end{pmatrix}$$

To undergo the drawbacks of the previous model in practice a different approach is used based on the same model and measurements. The encoder values are used to estimate the vehicles speed and thus avoid the contradiction and partly considers the input errors. The EKF based on this model achieved good results.

Model 4: Model with second order dynamics

State:

$$\begin{aligned} x_{k+1} &= x_k + T\cos(\phi_k)v_k + \frac{T^2}{2}\cos(\phi_k)a \\ y_{k+1} &= y_k + T\sin(\phi_k)v_k + \frac{T^2}{2}\sin(\phi_k)a \\ \phi_{k+1} &= \phi_k + T\omega_k + \frac{T^2}{2}\alpha \\ v_{k+1} &= v_k + Ta \\ \omega_{k+1} &= \omega_k + T\alpha \\ EncL_{k+1} &= EncL_k - T(v_k - r\omega_k) - \frac{T^2}{2}(a - r\alpha) \\ EncR_{k+1} &= EncR_k - T(v_k + r\omega_k) - \frac{T^2}{2}(a + r\alpha) \end{aligned} \tag{5.24}$$

Input Output

none $\mathbf{y} = \begin{pmatrix} X \\ Y \end{pmatrix}$

Another approach to avoid the input errors and at the same time get an estimate for v and ω is realized by additional system states. Here v and ω now are states influenced by acceleration a and angular acceleration α. a and α are assumed unknown and thus treated

as process noise.

This EKF also achieved good results but has a slower performance than model 3 due to the velocities that first have to converge after changes. The encoder values are not used.

Model 5: Basic model with slip estimation

State:

$$\begin{aligned} x_{k+1} &= x_k + \frac{T}{2}(v_{R,k+1}\bar{i}_{R,k} + v_{L,k+1}\bar{i}_{L,k})\cos(\phi_k) \\ y_{k+1} &= y_k + \frac{T}{2}(v_{R,k+1}\bar{i}_{R,k} + v_{L,k+1}\bar{i}_{L,k})\sin(\phi_k) \\ \phi_{k+1} &= \phi_k + \frac{T}{2r}(v_{R,k}\bar{i}_{R,k} - v_{L,k+1}\bar{i}_{L,k+1}) \qquad (5.25) \\ \bar{i}_{L,k+1} &= \bar{i}_{L,k} \\ \bar{i}_{R,k+1} &= \bar{i}_{L,k} \end{aligned}$$

(5.26)

with $\bar{i}_X = 1 - i_X$ and $i_X = \frac{v_{meas} - v_{track}}{v_{meas}}$.

Input Output

$$\mathbf{u} = \begin{pmatrix} v_{R,k+1} \\ v_{L,k+1} \end{pmatrix} = \begin{pmatrix} \frac{Enc_{R,k+1} - Enc_{R,k}}{T} \\ \frac{Enc_{L,k+1} - Enc_{L,k}}{T} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} X \\ Y \end{pmatrix}$$

The last tested model works comparable to model 4 but estimates the slip instead of v and ω. Track speeds v_X are estimated using the same approach as in model 3. Slip is assumed to be a factor $\bar{i}_x$ multiplied to the measured track speed to obtain the real track velocity. i_x is static (no differential equation) and estimated by the Kalman equations.[10] Using this approach the estimation errors were more severe compared to the previous two EKF's. The slip seems to change very quickly and so is difficult to estimate. By increasing the corresponding weights of Q it can be smoothed but is leading to much slower observer dynamics.

Delay compensation

As described in chapter 3 all sensor readings are transmitted asynchronous and lead to randomly varying delays. In the standard models above, these delays are part of the measurement noise. By using the knowledge of the delays it was tried to enhance the EKF performance.

A priori estimate

For state prediction all discussed models use the differential form of Euler integration which can be written as:

$$\dot{x} = f(x) = \frac{dx}{dt} = \frac{x_{k+1} - x_k}{dt} = \frac{x_{k+1} - x_k}{T} \tag{5.27}$$

$$\Rightarrow \quad x_{k+1} = x_k + T\dot{x} \tag{5.28}$$

Here and in the previous section the sampling time T was assumed to be constant. Due to varying delay T will be different for every sample and therefore has to calculated from the timestamps. Thus a new form of the difference equation was implemented in the form

$$x_{k+1} = f(x_k, T_k) \quad with\ T_k = t_{k+1} - t_k \tag{5.29}$$

Final prediction

Due to transmission delay τ_k of the measurements the a posteriori state $\hat{x}$ always will be a state in the past at time $t_k - \tau_k$. To get a more accurate estimate a further prediction for the interval τ_k is done using the equation

$$\tilde{x}_{k+1} = f(\hat{x_{k+1}}, \tau_k) \tag{5.30}$$

Here it is important to mention that this estimate only is forwarded and used as current state but not memorized. For the following sample the non-predicted state $\hat{x}$ is used as initial value.

Timeout

In few worse samples the transmission delays get higher than the sample time. In this case no new measurement is available when the EKF is evaluated. This is handled different for

the two sensor signals.

encoder signal

In case of no new encoder value, the previous track speed values are used

webcam data

If no actual position information is available, only the linear prediction 5.3 is performed. Update of covariances and the state are skipped

5.2.3 Results

After introducing the different EKF versions and the models they are based on, we now can try to evaluate the different EKFs and analyze the effects of delay compensation (DC) on the error.
All used filters are listed in table 5.1. Some implementations also contain special features that are mentioned in the description.

Concurring measurements

As already mentioned in 5.2.2 the usage of encoder and webcam measurements for the calculation of the Kalman gain and state correction destabilizes the observer. This originates from a gap between distance measured by the encoders and the really covered distance that leads to fluctuation error covariances and Kalman gains and a severe error in estimation of θ.
This effekt is shown in figure 5.4 for EKFs 2 and 3. Other than EKF 2, EKF 3 doesn't use the reference input for prediction but the velocity calculated by encoder values. Obviously this only brings little improvement since EKF 2 performs almost equal to EKF 3.
Also the estimation error of EKF 1 caused by slip is shown. Due to missing position information no correction for position and angle is possible.
Thus for the following analysis only models 3 to 5 are used.

Name	Description	Delay comp.
EKF 1	Model 1	no
EKF 2	Model 2	no
EKF 3	Model 4; Encoders and webcam used for update	no
EKF 4	Model 3	no
EKF 5	Model 5; With alpha → discarded	no
EKF 6	Model 5; Additional output v by differentiation	no
EKF 7	Model 5; Additional output v by states	no
EKF 8	Model 5; Delay compensation: only prediction	yes
EKF 9	Model 5	yes
EKF 10	Model 4	no
EKF 11	Model 4	yes
EKF 12	Model 3	yes

Table 5.1: List of tested EKFs.

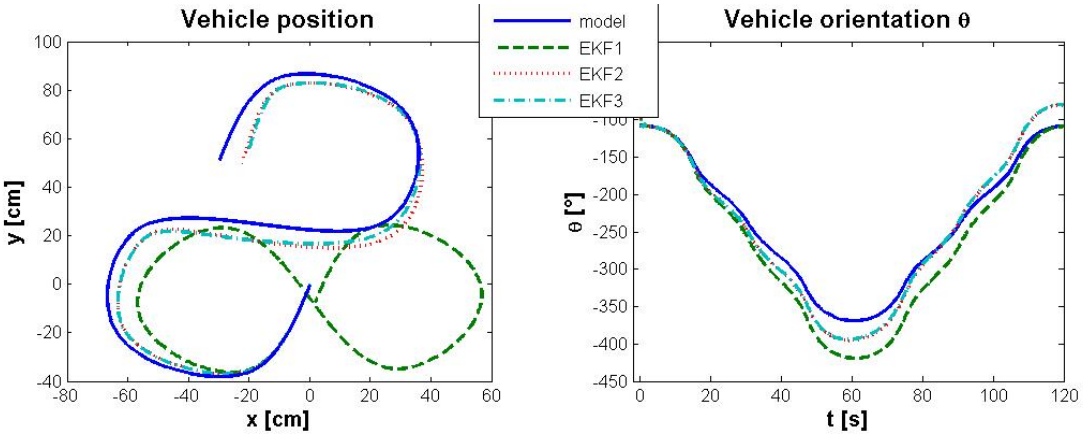


Figure 5.4: Comparison of EKF 1-3. EKF 1 estimates movement of ideal model since no update measurements are available. EKFs 2 and 3 show almost equal behaviour with severe static errors in angle and position.

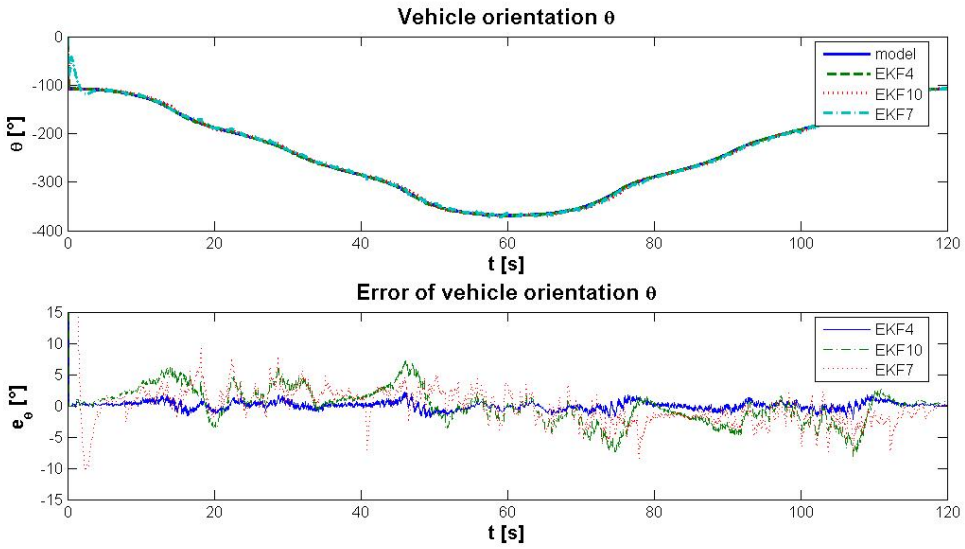


Figure 5.5: Estimation of θ by EKF 4, 7 and 10. The deviation from the real value (bottom plot) is significantly smaller for EKF 4.

Estimation of θ

For three selected EKF's figure 5.5 shows the estimate of the vehicle orientation in an example simulation.

The result is very promising with an error of less then 10° after short initial oscillations. These oscillations are extremely severe for EKF 7 which is critical as it can destabilize the controller in the initial phase.

An extremely good and stable estimate with an error of less than 3° is achieved by EKF 4, although it has the most simple model.

Position estimation

For position estimation the results are also satisfiying with an accuracy of less than 1.5cm for all EKFs.

Here significant high frequency oscillations can be observed which are caused by the direct measurements of this state and the resulting more intense correction by the Kalman equations.

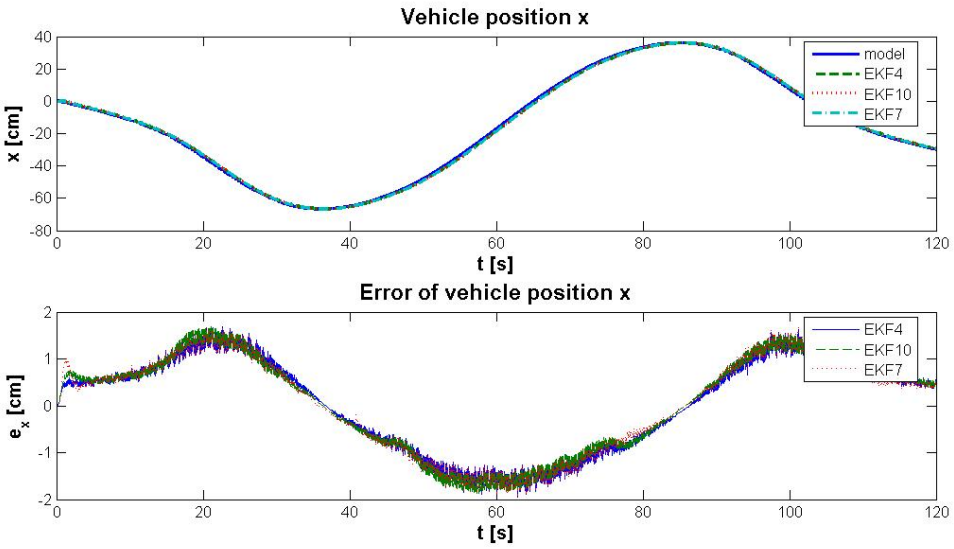


Figure 5.6: Estimate for X component of vehicle positon.

The sinusoidal general form of the error can be explained by the transmission delays that are responsible for the major part of this error. They cause the observer state to be late which leads a changing sign of the error depending on the motion direction.
Estimate and error of the three EKFs for the x coordinate are shown in figure 5.6. Here no filter shows clear superiority but again EKF 4 is the most stable without initial osciallations and a slightly better performance.

Velocity estimation

A further value needed by the controllers is the vehicle velocity v. It is only provided directly by EKFs working with models 4 and 5. Figure 5.7 shows the result based on the same experiment as before. Additionally v is determined by discrete differentiation and low-order (order 2-4) mean filtering.
EKF 10/11 show very slow convergence. This could be accelerated by augmenting the process noise covariance values, but leads to worse behaviour for the other estimates.
EKF 7 and 9 are much faster but also are oscillating much and with high magnitude. However EKF 9 would give the best compromise between convergence speed and error.

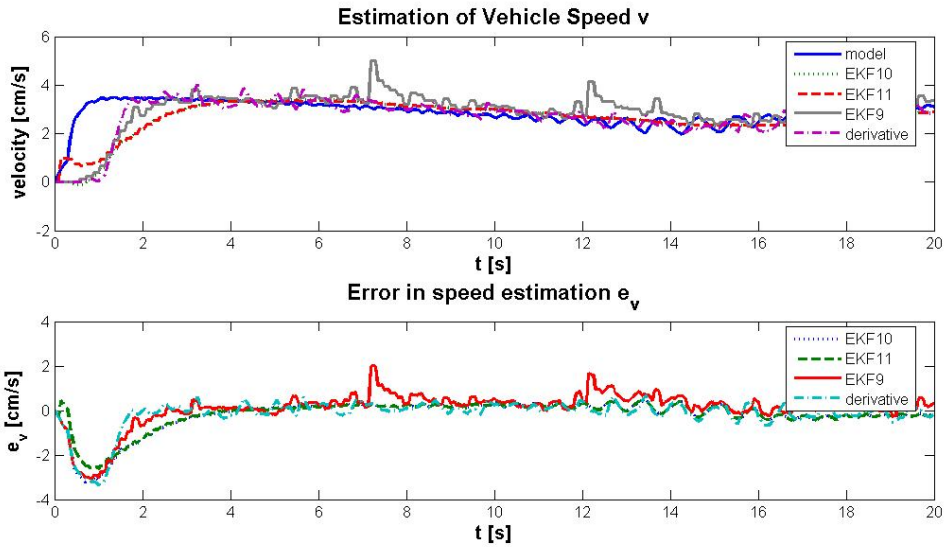


Figure 5.7: Estimate for vehicle velocity v. Differentiation and EKF 9 show the fastest convergence, while EKF 10 and 11 are much to slow. The error peaks occurring in EKF 9 a caused by missing measurement samples (timeout).

For the analyzed simulations, differentiation of the velocity signal even performed almost equally fast and with less fluctuations. If this also holds in real experiments has to be analyzed.

Effect of delay compensation

Figures 5.8 and 5.9 show the estimation error in x and θ for the three filters used before and their pendant with delay compensation.

A real enhancement only is visible for the position estimate and little for the orientation in case of EKF 10/11 and EKF 7/9. It seems like in all three types, but especially for model 3, the change in θ is caused mainly by the Kalman correction and not by the differential equation.

Since in all cases exists no differential equation with known input for v, the prediction here has no effect (as can be seen in fig. 5.7). On the other hand the error in v causes

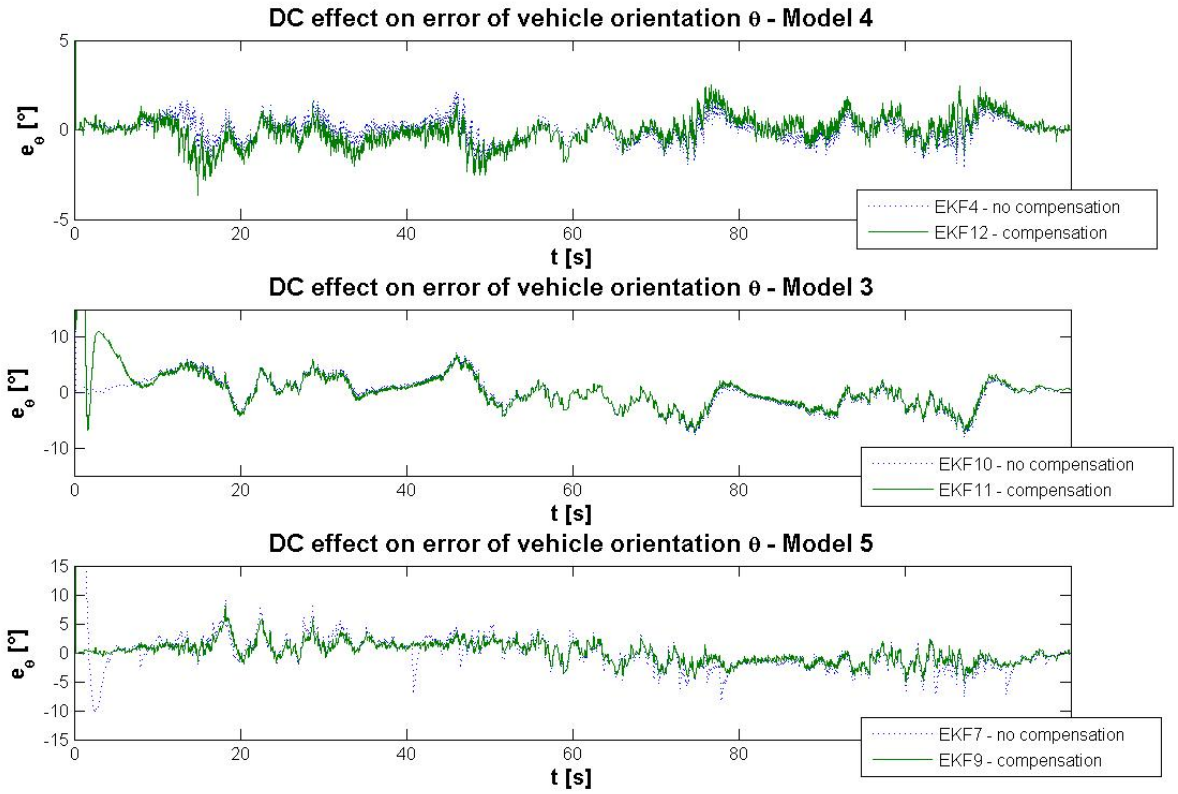


Figure 5.8: Comparison of orientation estimation between DC and non-DC realization for all three filter types. Only in case of EKFs 7 and 9 a slight enhancement can be observed.

the major part of the controller response as we will see in the following chapters. Thus only little improvement on the controller stability and performance can be expected. The assumption that a mayor part of the error is caused by delays can be partly confirmed here. Shape and sign of the error are equal to their pendant without DC but the magnitude is less, however the difference is smaller than expected. Corresponding the angle θ most of the error appears to originate from the observer and observer dynamics respectively.

Concluding the previous results, EKF 4/12 seem to be the best choice for estimation of position and orientation. Since they don't provide an estimate for v here additionally EKF 9 or a filtered differentiation has to be used. Both has be tested in experiments and will be discussed in chapter 8.

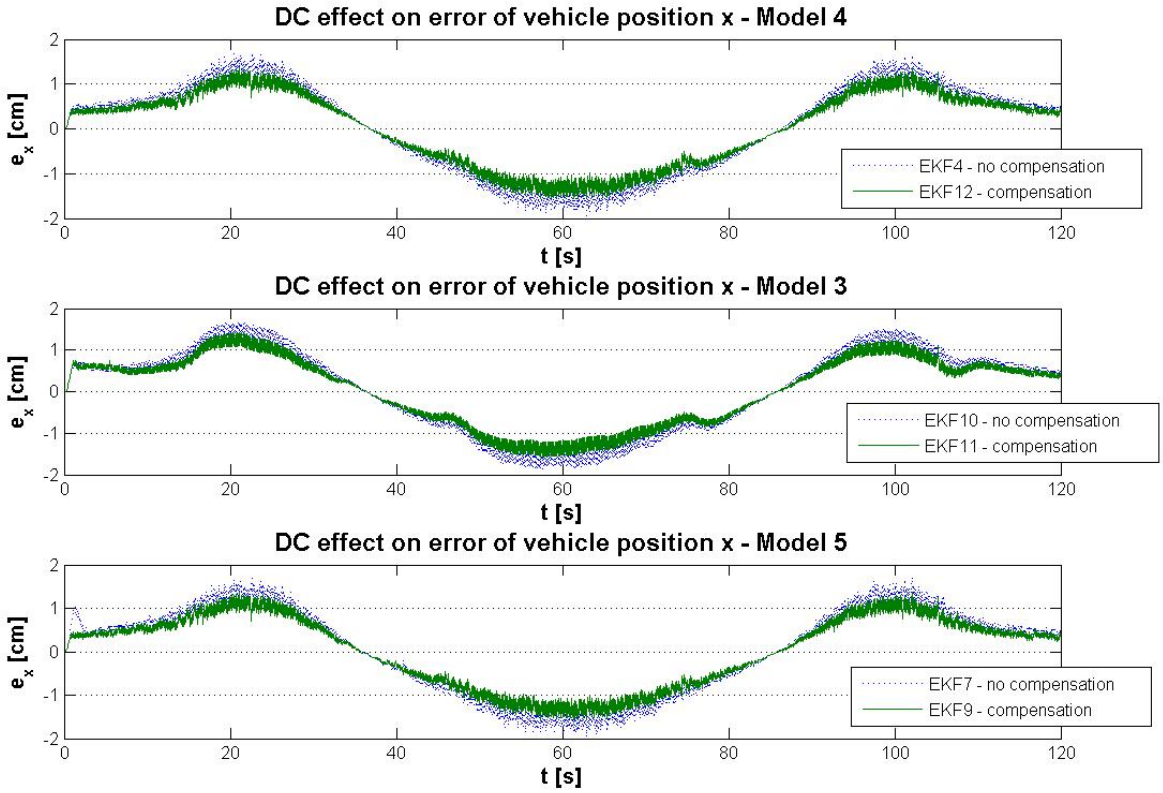


Figure 5.9: Comparison of position estimation between DC and non-DC realization for all three filter types. In all cases the effect is comparable and reduces the error up to 25%.

Chapter 6

Trajectory generation and path planning

After treating the all technical requisites for the control experiment, in the following section the generation of the reference trajectory and the feedforward control will be discussed. Therefore we'll first have a look at how to generate a time-dependent trajectory through a given set of points and then discuss methods for building these sets in combination with the task to avoid obstacles.

6.1 Trajectory generation

When control theory got accretively important in the beginning of the 20th century, most applications were restricted to stabilizing the control system around a certain set point. But with further knowledge and new control techniques, applications like set point changes or process starts and shut-downs became control objective and required to generate time dependent set points, so-called reference trajectories. Furthermore applying a smooth trajectory between two set points avoids stress to the actuators.
Depending on the system, trajectory generation was (and still is) a very challenging problem and required new mathematical and numerical methods and tools. One example are high-dimensional systems like industrial robots which have to perform precise movements

with a high number of degrees of freedom and include many constraints and the problem of multiple possible solutions on the other hand.
A more simple problem, restricted to two dimensions, is given in mobile robotics, when a time dependent trajectory through a map of given points (x_i, y_i) in the plane is required. As this is one major aspect of the control problem treated in this thesis, the generation of reference trajectories will be discussed closer in this chapter.

6.1.1 Common methods

For trajectory generation in the meantime a wide variety of methods have been developed, where the following section should give a brief overview.

Linear movements

Here the trajectory consists of a straight line between the last way-point (x_i, y_i) and the following one (x_{i+1}, y_{i+1}) . The time parameterized representation of the trajectory can be written as follows:

$$\begin{aligned} x &= x_i + sign(x_{i+1} - x_i)k \cdot (t - t_i) \\ y &= y_i + k \cdot m \cdot (t - t_i) \end{aligned} \tag{6.1}$$

with $m = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$ and $v = k \cdot \sqrt{1 + m^2}$.
The velocity v can be set by the parameter k, depending on the slope m. Important properties of this method are constant speed and no rotation between two way-points. The drawback are jumps in velocity and singularities in acceleration reference values at the way-points. The possible trajectories with this method are very limited, or a high number of way-points has to be chosen to approximate more complex curves.
Furthermore, for example parabolic blending can be used to overcome singularities in the way-points, as described in [17]. It generates a constant acceleration around the way-points, but is less precise than the following techniques as it may not pass them.

Polynomials

Generation of more smooth curves can be done by using polynomial approximations. These are performed for a one dimensional problem, meaning the approximation of a function f(t) described by a set given points (t_i, y_i). For n defined points a unique polynomial in time t of degree n-1 can be found that includes all points (assuming all t_i being different).

First the time vector $\mathbf{t}$ has to be generated which for example can be done by a cumulative sum of the euclidian distances between the way-points.

For a polynomial in the form $p(t) = c_1 t^{n-1} + c_2 t^{n-2} + \ldots + c_{n-1} t + c_n$ the following equation system has to be solved to obtain the polynomial coefficients c_i.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & t_1 & \ldots & t_1^{n-1} \\ 1 & t_2 & \ldots & t_2^{n-1} \\ \vdots & & & \\ 1 & t_n & \ldots & t_n^{n-1} \end{bmatrix} \begin{bmatrix} c_n \\ \vdots \\ c_2 \\ c_1 \end{bmatrix} \tag{6.2}$$

The left side represents the values of one component of the way-point coordinates. As t_i and y_i are known values this equation system is linear with a unique solution for all t_i being different. This polynomial approximation has to be done separately for every dimension in case of higher dimensional systems resulting in n polynomials $p_i(t)$, each representing one element of time parameterized coordinate $\mathbf{p}$(t).

Drawback of this method is the high polynomial orders for trajectories with many way-points which can lead to unpredictable variations especially on larger intervals. So it is mostly used for local function approximations.[16]

Splines

The most common method and a special case of polynomial approximations are piecewise polynomial functions of low order, which were called splines by their inventor I.J. Schoenberg.[16]

The interval $[a, b]$ on that the function f is approximated, is divided into sufficiently small intervals $[\xi_i, \xi_{i+1}]$, such that on each interval a polynomial p_j of relatively low

degree can provide a good approximation to f. This is done in such a way that the polynomial pieces blend smoothly, i.e., so that the resulting patched or composite function s(x) that equals $p_j(x)$ for $x \in [\xi_i, \xi_{i+1}]$,in all j, has several continuous derivatives. The points x_i that subdivide the intervals are called via points or breaks.

This method is more adequate for building vehicle trajectories although it doesn't allow to consider constraints (e.g. in velocity=slope) directly. In case of restrictions the generation has to be performed iteratively.

Optimal control methods

Optimal control methods, such as model predictive control methods, use a dynamic model of the system in combination with techniques like linear or non-linear programming for calculation and prediction of a trajectory between two states of the system.

Thereby they return the one trajectory that minimizes a certain performance index Q. They also allow to include constraints for system inputs or system states, for instance.

This method mostly is used directly as a controller since it returns the optimal system input additionally to the system trajectory. In general optimal control methods are computationally very expensive and little demonstrative for the given problem.

6.1.2 Cubic splines

Within the methods described in the previous section, splines turned out to be the best choice for generating a simple and smooth vehicle trajectory in a plane. In contrary to optimal control methods that premise advanced knowledge in programming, systems theory and optimality concepts, they only require little mathematical knowledge to be understood by students in practicals.

The piecewise polynomial form (ppform) of a k'th order polynomial spline (for a cubic

spline k=4) can be written as

$$p_j(x) = \sum_{i=1}^{k} (x - \xi_j)^{k-i} c_{ji}, \qquad j = 1 : l \tag{6.3}$$

, where j is the index of the interval defined by the breaks $\xi_1, ..., \xi_{l+1}$. c_{ji} represent the local polynomial coefficients of the l pieces.

It should be mentioned that there exists a second spline representation called B-form that is used mostly during the spline construction. As this representation is less clear but equivalent to the ppform it won't be treated further at this point.

The popularity of splines can be explained by it's convenient properties like smoothness and continuity in function and derivatives. Especially cubic splines, meaning piecewise polynomials of order t^3, posses the well-tempered properties of being twice continuously differentiable which means a continuous velocity and acceleration profile without jumps.

Continuity of the function and it's k-1 derivatives is obtained during the spline construction by constraints for the values of function and derivatives at the interval borders. These constraints are needed for uniqueness of the solution. They can be influenced by a property called knot multiplicity that means how often a knot (here value in the time vector) appears. For a knot multiplicity greater than one, constraints for derivatives at this knot are neglected by the rule

$$knot\ multiplicity + condition\ multiplicity = polynomial\ order \tag{6.4}$$

where the polynomial order is fixed. In this way it is possible to soften the constraints regarding continuity of derivatives.

6.1.3 Using MATLAB spline toolbox

For working with splines the MATLAB spline toolbox offers a wide variety of generation, manipulation and utilization functions. It offers 4 different methods to generate cubic splines.

csapi

generates a spline interpolation f(t) through the given via points

csape

generates a spline interpolation f(t) through the given via points; additionally various types of end conditions (derivatives at first and last point) can be demanded.

cscvn

generates natural or periodic splines. In case of equal first and last point, periodic end conditions are applied (equal first and second derivative in end points), otherwise variational end conditions are used (second derivatives equal to zero in end points). The time parametrization of the knots is calculated by a cumulative sum of the euklidic distances between the points (all dimensions). So the parameter value t(j) for the j'th knot is given by $\sum_{i=1}^{j} \sqrt{x_{i+1} - x_i}$.

csaps

generates a smoothing spline f(t) that approximates the data values. This is done by minimizing the the performance index $p \sum_{j=1}^{n} w(j)|y(:,j) - f(x(j))|^2 + (1-p) \int \lambda(t)|D^2 f(t)|^2 dt$ where w is the vector of error weights. λ is called roughness measure and represents a weighting function for the second derivative of f(t). p is called smoothness factor and influences the ratio between smoothness (derivatives) and approximation precision inside the performance index.

All methods produce a spline structure containing all information on the polynomial pieces. With methods for manipulation points can be added or removed easily without the need of re-creating the complete spline.
Furthermore the user doesn't have to concern about polynomials or intervals since functions as *fnval(spline,t)* directly evaluate the spline at the required times **t**. The spline toolbox also provides the function *fnder(spline)* and *fnint(spline)* that generate the splines derivative or integral function.

Especially the derivatives are needed for calculation of the feedforward control by inverse kinematics which is treated in the following section.

6.1.4 Inverse kinematics

The created splines give us a time-parametrization x(t) and y(t) for the reference trajectory. Using the described functions we also can obtain the reference velocities and accelerations $\dot{x}(t)$, $\dot{y}(t)$, $\ddot{x}(t)$ and $\ddot{y}(t)$ by differentiation.

These can directly be used as reference values for the flatness based controller design described in 7.3.

However the linear controller developed in 7.1 needs a reference trajectory for angle θ, velocity v and angular velocity ω of the vehicle.

By inverting the forward kinematics

$$\begin{bmatrix} \dot{x}_{ref} \\ \dot{y}_{ref} \\ \dot{\theta}_{ref} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{ref}) & 0 \\ \sin(\theta_{ref}) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{ref} \\ \omega_{ref} \end{bmatrix} \tag{6.5}$$

the inverse kinematics result as follows:

$$\begin{bmatrix} v_{ref} \\ \omega_{ref} \end{bmatrix} = \begin{bmatrix} \sqrt{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} \\ \dot{\theta}_{ref} \end{bmatrix} \tag{6.6}$$

As θ results from the two components of the overall velocity $\mathbf{v}= \begin{bmatrix} v_x \\ v_y \end{bmatrix}$ by

$$\theta_{ref} = \arctan \frac{\dot{y}_{ref}}{\dot{x}_{ref}} \tag{6.7}$$

the angular velocity ω can be obtained by differentiating θ by t.

$$\omega_{ref} = \dot{\theta}_{ref} = \frac{\dot{x}_{ref}\ddot{y}_{ref} - \dot{y}_{ref}\ddot{x}_{ref}}{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} \tag{6.8}$$

Finally the inverse kinematics can be written as

$$\begin{bmatrix} v_{ref} \\ \omega_{ref} \end{bmatrix} = \begin{bmatrix} \sqrt{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} \\ \frac{\dot{x}_{ref}\ddot{y}_{ref} - \dot{y}_{ref}\ddot{x}_{ref}}{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} \end{bmatrix} \tag{6.9}$$

or directly as references for the track speeds (feed-forward control)

$$\begin{bmatrix} v_{R;ref} \\ v_{L;ref} \end{bmatrix} = \begin{bmatrix} v_{ref} + r\omega_{ref} \\ v_{ref} - r\omega_{ref} \end{bmatrix} = \begin{bmatrix} \sqrt{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} + r\frac{\dot{x}_{ref}\ddot{y}_{ref} - \dot{y}_{ref}\ddot{x}_{ref}}{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} \\ \sqrt{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} - r\frac{\dot{x}_{ref}\ddot{y}_{ref} - \dot{y}_{ref}\ddot{x}_{ref}}{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} \end{bmatrix} \tag{6.10}$$

Time scaling

As the MATLAB function for constructing natural splines generates the time parametrization automatically it is necessary to perform a time scaling to slow down or speed up the trajectory to the desired period time. Also for the other generation functions this is useful as the spline doesn't have to be re-built when changing the period duration.

Time scaling can be achieved by introducing a new time

$$t_{new} = \frac{T_{SPLINE}}{T_{DEMAND}} t = k_{scale} t \tag{6.11}$$

. So the spline $\mathbf{f}$(t) defined on $t \in [0, T_{SPLINE}]$ can be transformed in the spline $\hat{\mathbf{f}}(t) = \mathbf{f}(t_{new}(t))$ defined on $t \in [0, T_{DEMAND}]$. Furthermore the scaling has to be extended to the derivatives by

$$\begin{aligned} \hat{\dot{x}}(t) &= k_{scale}\dot{x}(t_{new}(t)); \quad \hat{\dot{y}}(t) = k_{scale}\dot{y}(t_{new}(t)); \quad \hat{\dot{\omega}}(t) = k_{scale}\dot{\omega}(t_{new}(t)) \\ \hat{\ddot{x}}(t) &= k_{scale}^2\ddot{x}(t_{new}(t)); \quad \hat{\ddot{y}}(t) = k_{scale}^2\ddot{y}(t_{new}(t)) \end{aligned} \tag{6.12}$$

In case of periodic splines shall be used outside the interval $[0, T_{DEMAND}]$ one additionally has to perform a modulo calculation

$$t_{new} = k_{scale} mod(t, T_{DEMAND}) \tag{6.13}$$

because the spline function f is only periodic within the defined interval (meaning same derivatives in the two boundary points). The property f(t + T) = f(t) of a real periodic function doesn't hold outside in this case.

6.1.5 Examples

Using the basics treated in the previous sections some simple trajectories were constructed for the LEGO vehicle. Therefore different construction methods of the spline toolbox were

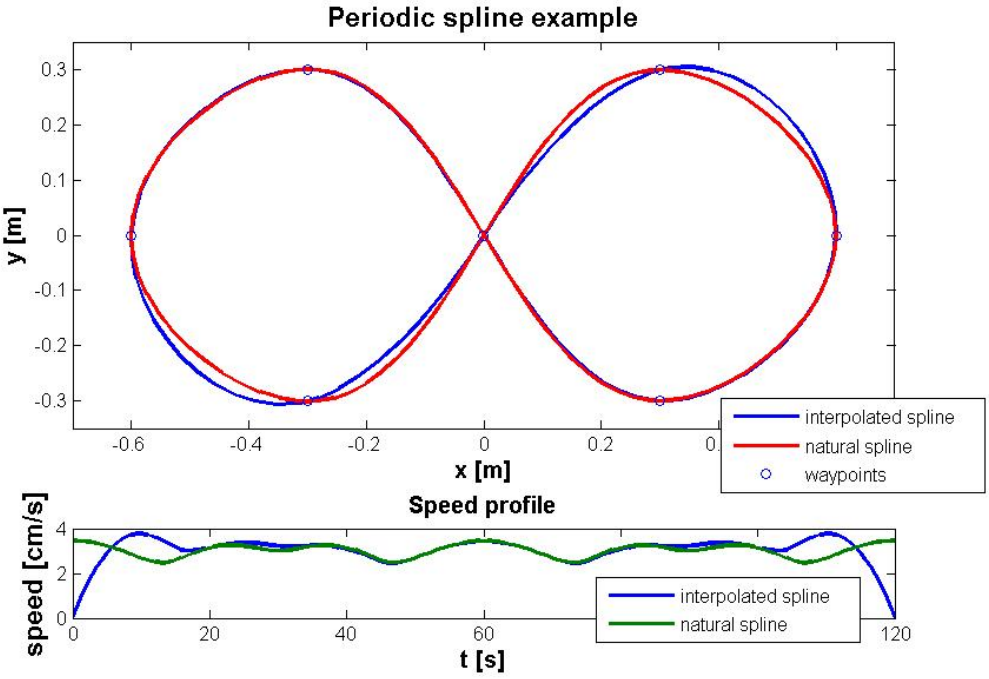


Figure 6.1: Periodic example trajectory with speed profile. The function cscvn delivers a periodic spline that is exactly symmetric. The advantage of csape (interpolated) are the zero velocity end conditions.

used and compared for two types of trajectories, periodic and non-periodic trajectories. In all cases a knot multiplicity of 1 was chosen to generate the trajectories as smooth as possible.

Figures 6.1 and 6.2 show two unscaled trajectories with the corresponding speed profiles. As can be seen in Fig. 6.1 the natural periodic spline returns a perfectly symmetric trajectory whereas the interpolation trajectory gets asymmetric in the first and last curve due to the velocity constraints $v_0 = v_{end} = 0$.

The second figure shows two non-periodic trajectories created by cscvn (natural spline) and csaps (smooth spline) with a very low smoothness parameter p=1e-5. It shows very clearly the smoother behaviour of the second spline, only possible by not exactly crossing the via points.

Recapitulating the results and experiences made within this work natural splines and

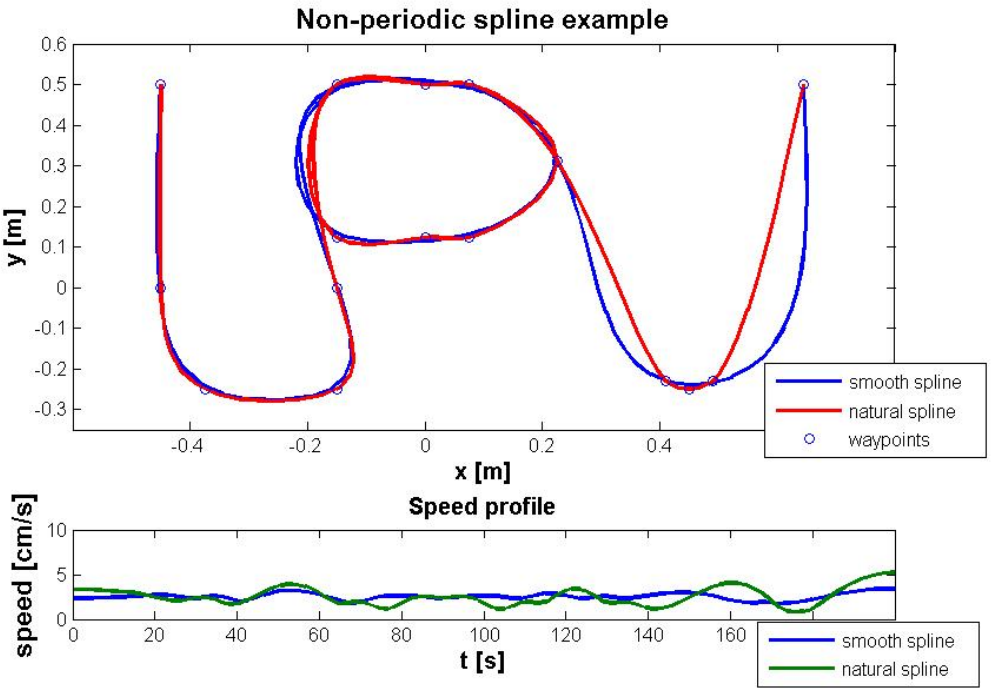


Figure 6.2: Non-periodic example trajectory with speed profile. Using natural splines sometimes generates tight curves. By using smoothed spline this effect can be reduced by allowing the spline to not pass the waypoints exactly.

interpolating splines turned out to generate the best trajectories, depending on if constraints in the form of initial and final speeds are needed.
The results show that splines, especially in combination with the MATLAB spline toolbox, offer an easy and fast method for vehicle trajectory generation.

6.2 Path search and path generation

Besides trajectories consisting of simple geometric forms a small example application was created to combine the trajectory generation with path search algorithms in form of navigating between two points and avoiding obstacles. Therefore firstly a small overview of common methods will be given, followed by the description of the used method.

6.2.1 Common methods

Road map methods

Road map approaches reduce the robot's free space to a network of 1D curves or lines, called road maps. Therefore mostly as set of knot points is generated and connected to neighbour points by certain rules. This is done by straight lines and a local planner that checks for obstacle collisions.
After construction of this road map it is used for the robot's path planning which is done by connecting the initial and goal positions of the robot to the road network and then searching for a series of roads (path) from the starting position to the robot's desired position.
The challenge is to construct a set of roads that together enable the robot to go anywhere in its free space, while minimizing the number of total roads.
Examples for this kind of path generation techniques are probabilistic road maps, Voronoi diagrams, visibility graphs and rapidly-exploring random trees.[37],[28]

Cell decomposition methods

Cell decompositions methods work by dividing the target area into small connected geometric areas or cells, discriminating if the cells are free or occupied by obstacle objects. As a next step for all free cells it is determined to what cells they are connected and a connectivity graph is created.
By searching the cells that include the initial and goal point a path that connects them is searched in the connectivity graph. From the sequence of cells found with an appropriate searching algorithm, a path within each cell can be computed, for example, passing through the midpoints of the cell boundaries or by a sequence of wall-following motions and movements along straight lines.
An important aspect of cell decomposition methods is the placement of the boundaries between cells. If the boundaries are placed such that the decomposition is lossless, the method is called exact cell decomposition. If the decomposition results in an approximation of the actual map, the system is called approximate cell decomposition.
Examples for this kind of methods are fixed decomposition (rasterization), or exact cell decomposition[37].

Potential field methods

Potential field methods use an artificial potential field defined on the robot's map to lead it to it's target. This is achieved by constructing a potential field function $U(\mathbf{x})$ such that obstacles represent local potential peaks whereas the target marks the global minimum. The robot is treated as a point under influence of this potential field and is driven in direction of the potential fields gradient like a ball rolling into a valley. So the target acts as an attractive force on the robot (the valley) whereas obstacles represent repulsive forces (hills).
Such an artificial potential field smoothly guides the robot towards the goal while simultaneously avoiding known obstacles. It has to be mentioned that this method not only is a path planning algorithm but simultaneously can generates a control input for the robot

when used dynamically. If the robot can detect it's position in the map it can calculate the reference direction using the potential field and react to model or input errors in that way. It also can react to moving obstacles by updating the potential field function with the obstacle data (positions, dimensions).

6.2.2 Path generation by rasterization

For demonstration of path search algorithms, a small example application was created in the course of this thesis. Therefore colored markers similar to the marker of the vehicle were posed on the ground, representing imaginary obstacles.
For path generation it was decided to use a cell decomposition method with a fixed cell size (rectangular) which also is called rasterization. This method divides the vehicles map (here the vision range of the camera) into equal cells by a grid of predefined size. It is verified for each cell if it contains or is part of an obstacle using image processing routines. In case of a detected obstacle the cell is blocked. Though a possible path form initial point to the target only is possible along the free cells.
Utilizing this method the robots movement will be generated along the grid (or better cell centers). As rasterization is working with constant cell dimensions these only have to be chosen once. Here the robot dimensions and dynamics have to be taken into account for choosing the cell size sufficiently big for no collisions can occur. As the dynamic of the vehicle used for this experiments is very fast (movement after stop signal at max speed $<$ 2cm) and turning is possible on the spot the raster length can be chosen slightly more than the vehicles diagonal (here 25cm).

In the example application image processing is done directly in MATLAB using the webcam client described in chapter 3.2 to obtain starting position of the vehicle. A current webcam image also can be receiver via TCP by use of the webcam clients function getPicture(). The binary image is generated similar to the method for detection of the vehicles marker described in chapter 3.2.1. For each point the expression
$g - b - r \geq threshold$

is evaluated resulting in a boolean array of the pictures dimensions. Figures 6.3 and 6.4 show the result of the segmentation applied to an example picture. After generation of the

Figure 6.3: Raw picture for a path search example. The green objects are virtual obstacles the vehicle has to avoid.

map containing the obstacles the following algorithm is used to determine the path.

in image coordinates

- determine position of vehicle
- calculate raster where vehicle position is in the center of a cell; neglect uncomplete boundary cells
- create reduced array (cell indices) and determine if field is free or obstacle

- determine indices of initial point and target point
- calculate for each field the distance to initial field
- apply reverse search from target point to initial point. Create path of cells with distance decreasing in steps of one

in real world coordinates

- create path by cell centers and transform it into real-world coordinates using the transformation matrix obtained by the camera calibration in chapter 3.2.2
- generate spline with cell centers as via points
- adapt trajectory time T by T = k·cells to go

The result of the algorithm is a spline that can be directly used as trajectory for the robot. As this kind of trajectory is non-periodic the interpolating spline with zero-velocity end conditions turns out be be a suitable generation method. Figure 6.4 shows the result of the path search algorithm applied on the example of Fig. 6.3 using a smooth spline.
It also shows one of the major drawbacks of this method. Small obstacles can block a complete cell and thereby prohibit a possible passageway even if there is enough space for the vehicle. This problem gets more severe the bigger the cell size gets compared to the obstacles. In the presented example this is clearly the case as the vision field of the camera is very limited. In this case it also depends on the initial position what paths are feasible as some paths only are possible for a certain offset range of the grid. So a possible extension to this method would be to alter the x- and y-offset of the grid to find alternative better or shorter paths.
Furthermore the presented method and algorithm won't generate an optimal path in the sense of path length as the vehicle moves along the grid. Since this application only was developed for demonstration purposes this also would be a possible extension for future works or student projects. Alternative path search methods could be used or diagonals

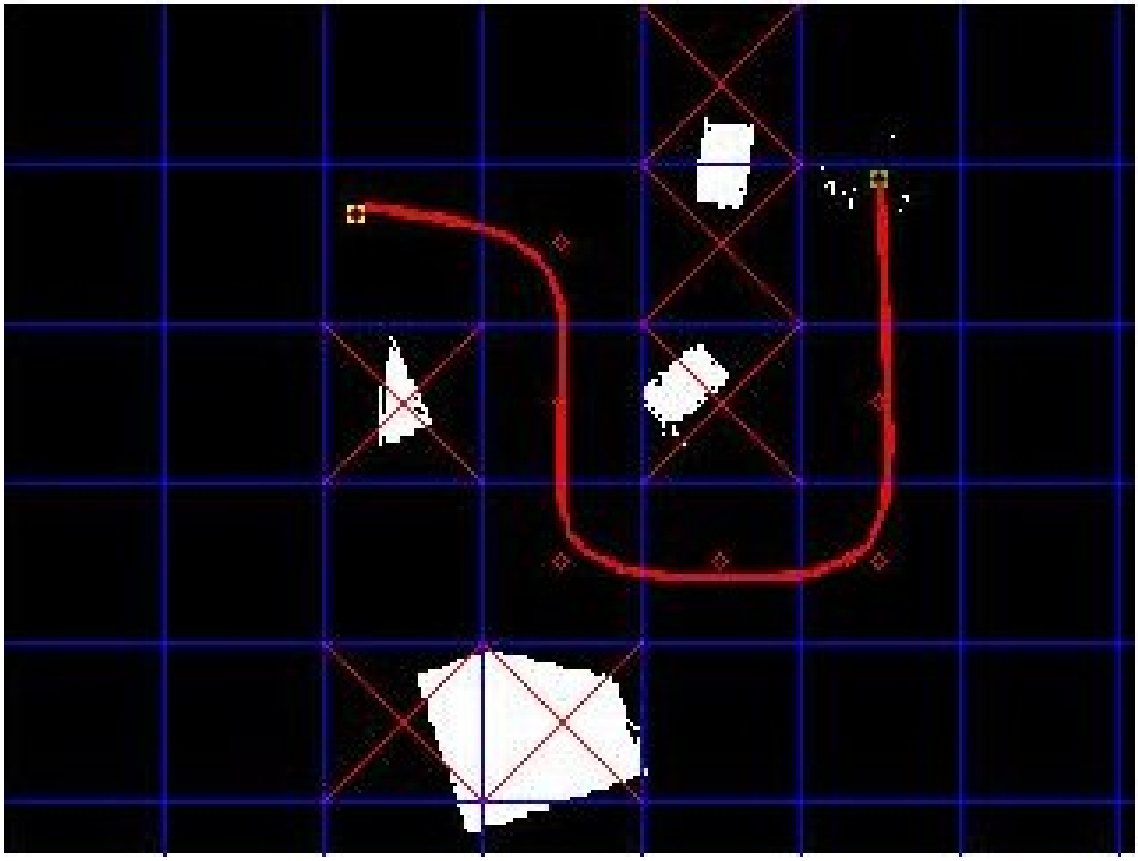


Figure 6.4: Result of the path search algorithm applied to Fig. 6.3

could be introduced in the existing method.

Chapter 7

Trajectory controllers

In the following section, two different trajectory controller types will be developed and applied on the system. Both are based on the kinematic model defined in 5.1.1. Friction and slip error as well as deviation of the PID controlled track speeds have to be compensated by this controllers.

For simplification of the model equations, instead of the track speeds v_{left} and v_{right}, the overall speed v and the angular velocity ω will be used as system inputs. They can easily be converted into each other by

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{v_{right}+v_{left}}{2} \\ \frac{v_{right}-v_{left}}{2r} \end{bmatrix} \tag{7.1}$$

and

$$\begin{bmatrix} v_{right} \\ v_{left} \end{bmatrix} = \begin{bmatrix} v + r\omega \\ v - r\omega \end{bmatrix} \tag{7.2}$$

with r being the distance between barycenter and tracks.

7.1 Linear state-feedback control by error linearization

The first approach used to control the vehicle along a given trajectory is a linear state-feedback controller. In the following section the derivation of the feedback law will be

discussed.

7.1.1 Feedback law

For the reference trajectory the vehicle kinematics are given as

$$\begin{bmatrix} \dot{x}_{ref} \\ \dot{y}_{ref} \\ \dot{\theta}_{ref} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{ref}) & 0 \\ \sin(\theta_{ref}) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{ref} \\ \omega_{ref} \end{bmatrix}. \tag{7.3}$$

We define the error vector

$$\mathbf{e} = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \\ \theta_{ref} - \theta \end{bmatrix} \tag{7.4}$$

in global coordinates and transform it into the vehicles local coordinates by

$$\begin{bmatrix} e_{\hat{x}} \\ e_{\hat{y}} \\ e_{\hat{\theta}} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{ref}) & \sin(\theta_{ref}) & 0 \\ -\sin(\theta_{ref}) & \cos(\theta_{ref}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \\ \theta_{ref} - \theta \end{bmatrix}. \tag{7.5}$$

Differentiation of time leads to the following error dynamics:

$$\begin{aligned} \dot{e}_{\hat{x}} &= -\omega_C(x_{ref} - x)\sin(\theta) + (\dot{x}_{ref} - \dot{x})\cos(\theta) \\ &+ \omega_C(y_{ref} - y)\cos(\theta) + (\dot{y}_{ref} - \dot{y})\sin(\theta) \qquad (7.6) \\ \dot{e}_{\hat{y}} &= -\omega_C(x_{ref} - x)\cos(\theta) - (\dot{x}_{ref} - \dot{x})\sin(\theta) \\ &- \omega_C(y_{ref} - y)\sin(\theta) + (\dot{y}_{ref} - \dot{y})\cos(\theta) \qquad (7.7) \\ \dot{e}_{\hat{\theta}} &= \omega_{ref} - \omega \qquad (7.8) \end{aligned}$$

By substituting 7.3 into equations 7.6 - 7.8 and using addition theorems we can simplify the expressions to:

$$\begin{aligned} \dot{e}_{\hat{x}} &= -v_C + v_{ref}\cos(\theta_{ref} - \theta) + \omega_C e_y = -v_C + v_{ref}\cos(e_\theta) + \omega_C e_y \qquad (7.9) \\ \dot{e}_{\hat{y}} &= -\omega_C e_x + v_{ref}\sin(\theta_{ref} - \theta) = -\omega_C e_x + v_{ref}\sin(e_\theta) \qquad (7.10) \\ \dot{e}_{\hat{\theta}} &= \omega_{ref} - \omega_C \qquad (7.11) \end{aligned}$$

As next step we define the input **u** as

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} v_{ref}\cos(e_\theta) - v_C \\ \omega_{ref} - \omega_C \end{bmatrix} \tag{7.12}$$

and replace the corresponding terms in equations 7.9-7.11. Thus we obtain the almost linear error dynamics

$$\begin{bmatrix} \dot{e}_{\widehat{x}} \\ \dot{e}_{\widehat{y}} \\ \dot{e}_{\widehat{\theta}} \end{bmatrix} = \begin{bmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_{\widehat{x}} \\ e_{\widehat{y}} \\ e_{\widehat{\theta}} \end{bmatrix} + \begin{bmatrix} 0 \\ \sin(e_\theta) \\ 0 \end{bmatrix} v_{ref} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u}. \tag{7.13}$$

Now the system can be linearized around the working point $u_1 = 0, u_2 = 0, e_x = 0, e_y = 0, e_\theta = 0$ and by applying the feedback law

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} -k_1 e_x \\ -k_2 sign(v_{ref}) e_y - k_3 e_\theta \end{bmatrix} \tag{7.14}$$

we obtain the following error dynamics for the closed-loop system:

$$\begin{bmatrix} \dot{e}_{\widehat{x}} \\ \dot{e}_{\widehat{y}} \\ \dot{e}_{\widehat{\theta}} \end{bmatrix} = \begin{bmatrix} -k_1 & \omega & 0 \\ -\omega & 0 & v_{ref} \\ 0 & -k_2 sign(v_{ref}) & -k_3 \end{bmatrix} \begin{bmatrix} e_{\widehat{x}} \\ e_{\widehat{y}} \\ e_{\widehat{\theta}} \end{bmatrix} \tag{7.15}$$

For the final control law we obtain

$$\begin{bmatrix} v_C \\ \omega_C \end{bmatrix} = \begin{bmatrix} v_{ref} cos(e_\theta) + k_1 e_x \\ \omega_{ref} + k_2 sign(v_{ref}) e_y + k_3 e_\theta \end{bmatrix} \tag{7.16}$$

by joining equations 7.12 and 7.14.

7.2 Choosing parameters

Now the error dynamics can be estimated values for the the proportional feedback gains have to be found.

It can be shown that the closed-loop system is stable for any combination of k_i in case of all

$k_i > 0$. Stability can be proofed with the Hurwitz criterion by building the characteristic polynomial:

$$Det(A - \lambda I) = \lambda^3 + (k1 + k3)\lambda^2 + (k1k3 + k2v^2 + \omega^2)\lambda + (k1k2v^2 + k3\omega^2) = 0 \quad (7.17)$$

For all $k_i > 0$ all coefficients are positive which is necessary but not sufficient. In case of a third order system additionally the condition $a_2a_1 > a_0a_3$ or $a_2a_1 - a_0a_3 > 0$ must hold for the polynomial coefficients a_i. Evaluating this expression leads to

$$a_2a_1 - a_0a_3 = k_1^2k_3 + k_2k_3v^2 + k_1(k_3^2 + w^2) \quad (7.18)$$

which always is positive for all $k_i > 0$.

As the algebraic solutions for λ_i as function of k_i get very complex, empirical studies about the parameters' influence on the error dynamic's eigenvalues were performed. The parameters were varied around the values proposed before. The following tendencies could be observed:

- the higher k1 and/or k3 the higher the real-part of all three eigenvalues
- the higher k2 and ω the higher the oscillations and the imaginary part of the pair of complex eigenvalues

For finding concrete parameters in [24] a building rule is proposed by

$$k_1 = k_3 = 2\zeta\sqrt{\omega_{ref}^2 + \beta v_{ref}^2} \quad (7.19)$$

$$k_2 = \beta|v_{ref}| \quad (7.20)$$

with $\beta, \zeta > 0$. With this parameter set the system behaves like a second order system, affected by the parameters β and ζ.

The proposed parameter values represent a good initial configuration for the controller. It has to be pronounced that caused by the high delay times too high proportional feedback can destabilize the closed-loop system. This will be analyzed more closely in the following chapter.

7.3 Feedback linearization by non-linear state feedback

A totally different approach for trajectory control of the given kind of system can be realized with a flatness based feedback linearization by non-linear state feedback. Before turning to the explicit controller design firstly a view basics on flat systems shall be discussed.

7.3.1 Differential flatness

Flatness, mostly known as a geometric or organizational property, is associated with a certain level of simplicity. In analogy to this flat systems also posses a flat structure that simplifies the understanding of functional dependencies and control of such kind of systems. The concept of flatness was introduced in 1992 by Fliess, Lévine, Martin and Rouchon. It describes a system property that is characterized by the existence of a specific virtual or real system output, called flat output.

By means of this output a flat system can be described like linear systems in linear spaces by a set of suitable coordinates. It also possesses controllability properties similar to linear systems. Furthermore it is possible to describe all system states and inputs with these coordinates. The new coordinates are linked to the original coordinates by a formally unique invertible non-linear interrelation.

Because of all these properties flat systems turn out to be very good natured in respect of feedforward and feedback control. In many cases trajectories also can be planed easily.

As already mentioned a flat system is characterized by the existence of a flat output

$$\mathbf{y} = \Phi(\mathbf{x}, u_1, ..., u_1^{(\alpha_1)}, ..., u_m, ..., u_m^{(\alpha_m)}). \tag{7.21}$$

This output normally is situated 'as far as possible' from the inputs. It has as many differential independent components $y_i, i = 1, .., m$ as the input $\mathbf{u}$ (dim $\mathbf{y}$ = dim $\mathbf{u}$). All states and inputs of the system can be described as a function of the y_i and a finite number of derivatives $y_i^{(k)}$, $k \geq 1$.

$$\mathbf{x} = \mathbf{\Psi_1}(y_1, ..., y_1^{(\beta_1)}, ..., y_m, ..., y_m^{(\beta_m)}) = \mathbf{\Psi_1}(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}, ..., \mathbf{y}^{(\beta)}) \tag{7.22}$$

$$\mathbf{u} = \mathbf{\Psi_2}(y_1, ..., y_1^{(\beta_1+1)}, ..., y_m, ..., y_m^{(\beta_m+1)}) = \mathbf{\Psi_2}(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}, ..., \mathbf{y}^{(\beta+1)}) \tag{7.23}$$

Using this functions a feedforward control can directly be applied for given trajectories of the flat output. If the trajectory is given only for the real system output, generation of the trajectory for the flat output can become very challenging. The complete feedforward control can be calculated offline and applied to the system as a lookup table, for example. Also feedback control can be applied very easily as flat systems always can be linearized by a static state feedback. This results in freely selectable linear error dynamics (assuming unconstrained inputs and outputs).

7.3.2 Controller design

How the previously described concept can be applied on the trajectory control of the LEGO vehicle will be demonstrated in the following section.

Flatness analysis

Controller design is based on the simplified kinematic model

$$\dot{\mathbf{x}} = \begin{bmatrix} cos(x_3)u_1 \\ sin(x_3)u_1 \\ u_2 \end{bmatrix} \tag{7.24}$$

with the state and output

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}; \qquad \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.25}$$

described in 5.1.1. Here x and y are the coordinates of the vehicles center of gravity whereas θ is the vehicles orientation angle. $\mathbf{y}$ at the same time represents a flat output for the system what can be shown by building the derivatives $\dot{y_1}, \ddot{y_1}, \dot{y_2}$ and $\ddot{y_2}$ and resolve the resulting equations for $\mathbf{x}$, $\mathbf{u}$ and $\dot{u}$.

$$
\begin{aligned}
y_1 &= x_1 & x_1 &= y_1 \\
y_2 &= x_2 & x_2 &= y_2 \\
\dot{y}_1 &= u_1 \cos x_3 & x_3 &= \arctan \tfrac{\dot{y}_2}{\dot{y}_1} \\
\dot{y}_2 &= u_1 \sin x_3 \quad \Leftrightarrow & u_1 &= \tfrac{\dot{y}_1}{\cos(\arctan \frac{\dot{y}_2}{\dot{y}_1})} \\
\ddot{y}_1 &= \dot{u}_1 \cos x_3 - u_1 u_2 \sin(x_3) & u_2 &= \psi_1(y_1, y_2, \dot{y}_1, \dot{y}_2, \ddot{y}_1, \ddot{y}_2) \\
\ddot{y}_2 &= \dot{u}_1 \sin x_3 + u_1 u_2 \cos(x_3) & \dot{u}_1 &= \psi_2(y_1, y_2, \dot{y}_1, \dot{y}_2, \ddot{y}_1, \ddot{y}_2)
\end{aligned}
\tag{7.26}
$$

As can be seen this relations are only valid locally ($x_3 \neq 0$). Furthermore x_3 and the inputs depend on the term $\frac{\dot{y}_2}{\dot{y}_1}$. In case of the vehicles velocity getting 0, singularities appear in this expressions because the system is no longer observable. But experience tells us that a vehicle can be stopped at any time, what means that these singularities don't matter in practice, but have to be eliminated in the implementation of the equations.

Feedforward control

Using the equations in 7.26 the feedforward control of the vehicle can be applied straight forward. As the spline trajectories directly deliver the functions for $y_1, \dot{y_1}, \ddot{y_1}, y_2, \dot{y_2}$ and $\ddot{y_2}$ the control inputs u_1 and u_2 can be calculated using the corresponding equations from 7.26. Furthermore the resulting control input can be transformed into the track speed references by using equation 7.2.

Feedback control

The feedforward control for trajectory described in the previous section can be calculated offline and applied easily to the open loop system in form of a lookup table.
If the system is stable, the initial state is known sufficiently precise and only small errors affect the system, open-loop control can achieve good results. If these conditions can't be guaranteed, a feedback control gets necessary to compensate the deviations. Because the treated LEGO vehicle control system includes errors like slip, gear friction, model errors

and random transmission delays such a feedback control in form of a state feedback was designed. In case of the described system there exist the following two possibilities: a quasi-static state feedback and dynamic state feedback, which introduces an additional state to the closed-loop system.

Quasi-static state feedback

In case of a feedback control the pursuit movement has to be stabilized. Therefor the new inputs

$$v_1 = \dot{y}_1, v_2 = \ddot{y}_2 \tag{7.27}$$

are introduced. By inserting them into equations for u_1 and u_2 in 7.26 the quasi-static state feedback law

$$u_1 = \frac{v_1}{\cos x_3} \, (\cos x_3 \neq 0) \tag{7.28}$$

$$\dot{u}_1 = \ddot{y}_1 \cos x_3 + \ddot{y}_2 \sin x_3 = \dot{v}_1 \cos x_3 + v_2 \sin x_3 \tag{7.29}$$

$$u_2 = \frac{v_2 - \dot{u}_1 \sin x_3}{u_1 \cos x_3} \, (\cos x_3 \neq 0) \tag{7.30}$$

can be obtained. An asymptotic follow-up control for the reference trajectory $\mathbf{y}_R(t)$ will be achieved by the control law

$$v_1 = \dot{y}_{1,R}(t) + p_{10}(y_{1,R}(t) - y_1) \tag{7.31}$$

$$\dot{v}_1 = \ddot{y}_{1,R}(t) + p_{10}(\dot{y}_{1,R}(t) - v_1) \tag{7.32}$$

$$v_2 = \ddot{y}_{2,R}(t) + p_{21}(\dot{y}_{2,R}(t) - \dot{y}_2) + p_{20}(y_{2,R}(t) - y_2) \tag{7.33}$$

with $y_{1,R}, y_{2,R} \in \mathcal{C}^2$.

By choosing p_{10}, p_{20} and p_{21} the error dynamics

$$\dot{e}_1 + p_{10}e_1 = 0, \; \ddot{e}_2 + p_{21}\dot{e}_2 + p_{20}e_2 = 0 \tag{7.34}$$

can be adjusted to the desired values.

The number of system states stays unchanged by using this approach.

Dynamic state feedback

The second possibility to design the feedback is given by a dynamic state feedback. Therefore a new state x_4 and two new inputs $\bar{u}_1$ and $\bar{u}_2$ are introduced.

$$x_4 = u_1,\ \dot{x}_4 = \bar{u}_1,\ \bar{u}_2 = u_2 \tag{7.35}$$

By choosing

$$\bar{v}_1 = \ddot{y}_1,\ \bar{v}_2 = \ddot{y}_2 \tag{7.36}$$

analog to 7.27 the following dynamic state feedback results by solving the equations for $\ddot{y}_1$ and $\ddot{y}_2$ in 7.26.

$$\dot{x}_4 = \bar{u}_1 \tag{7.37}$$

$$\bar{u}_1 = \bar{v}_1 \cos x_3 + \bar{v}_2 \sin x_3 \tag{7.38}$$

$$\bar{u}_2 = -\frac{\sin x_3}{x_4^2}\bar{v}_1 + \frac{\cos x_3}{x_4^2}\bar{v}_2 \quad (x_4 \neq 0) \tag{7.39}$$

Asymptotic follow-up control for the reference trajectory $\mathbf{y}_R(t)$ is guaranteed by the following control law for the new inputs $\bar{v}_1$ and $\bar{v}_2$.

$$\bar{v}_1 = \ddot{y}_{1,R}(t) + p_{11}(\dot{y}_{1,R}(t) - \dot{y}_1) + p_{10}(y_{1,R}(t) - y_1) \tag{7.40}$$

$$\bar{v}_2 = \ddot{y}_{2,R}(t) + p_{21}(\dot{y}_{2,R}(t) - \dot{y}_2) + p_{20}(y_{2,R}(t) - y_2) \tag{7.41}$$

Other than the quasi-static state feedback, the dynamic state feedback controller possesses one state what expands the state of the closed-loop system by one. This leads to the following error dynamics

$$\ddot{e}_1 + p_{11}\dot{e}_1 + p_{10}e_1 = 0,\ \ddot{e}_2 + p_{21}\dot{e}_2 + p_{20}e_2 = 0 \tag{7.42}$$

which also can be chosen freely. Furthermore the error dynamics can be set symmetric by choosing $p_{1x} = p_{2x}$.

Application to the real system

For the controller used in the laboratory experiments it was decided to work with dynamic

state feedback as it provides the possibility to set the error dynamics symmetric for x and y. Furthermore it is easy to avoid singularities in the equations for u_1 and u_2 by introducing a lower limit > 0 for the integrator state of x_4. Since the used trajectories always demand the vehicles speed being positiv and greater zero this is constraint is justified. Additionally an upper integrator limit can be used as an anti windup measure.

Another requirement for the application to the real system are the values of v_x und v_y. These sizes can't be measured directly and therefore have to be estimated, either by a state observer or by numerical differentiation. Both methods were used and compared in experiments what will be discussed in chapter 8.1.2.

Chapter 8

Simulation and real experiments

8.1 Simulation experiments

8.1.1 PI controller

The PI controller described in the previous chapter only uses the system state for calculating the input correction. As showed in chapter 5.2.3 EKF 12 gives the best position and angle estimate and therefore was used.
Normal simulations were run with random delay, parameterized with near-to-real distributions (see chapter 5.1.3). In the following section the delays' influence on the closed-loop stability will be analyzed. Therefore a constant delay was used.

Delays and stability

Systems with high time delays mostly are difficult to control because the delays are destabilizing the closed loop. In the underlying control system time delays are caused by sensor-to-controller and controller-to-actuator data transmissions and thus part of the system.
To evaluate their influence, simulations with increasing constant delays were performed. For controller 1 (PI) the results are presented in figure 8.1. The left plot shows results for the ideal model whereas on the right side the extended, error afflicted model was used.
A delay of 0.11s turned out to be the critical value for both models, any higher delays

destabilized the system. The value of 0.6 reflects an approximated mean value of the real system and shows a slightly bigger error than the system without delay. Obviously most of the error originates from the observer dynamics and the high sample time of the controller. The last dataset in both plots was obtained with a significantly bigger delay and a system with saturated ($v_max \approx 4.5$cm/s) inputs. This kind of saturation corresponds to the real system as the maximum reachable velocity is about 4.5 cm/s. Interestingly this saturation stabilizes the system even for higher delays. For the ideal model the corresponding error even is smaller in mean and magnitude than for the unsaturated system with 50% less delay. For the extended model the mean is comparable but still the saturation dampens the oscillation magnitude.

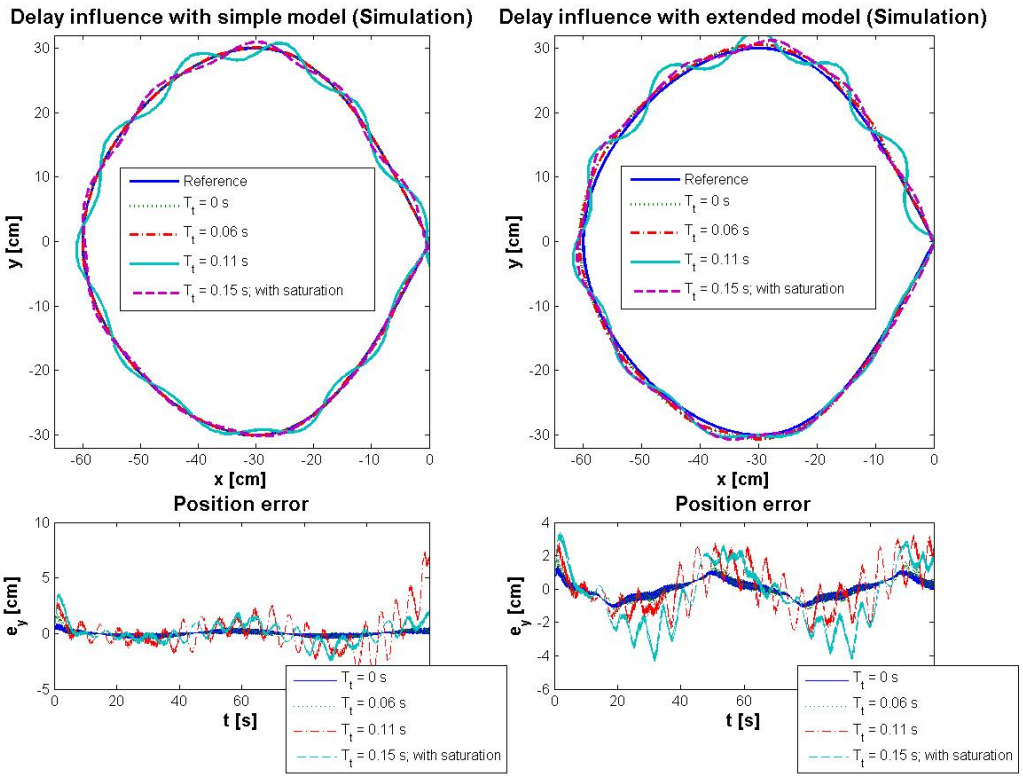


Figure 8.1: Delay influence on closed-loop system with controller 1. For both vehicle models 0.11s seems to be the stability border for the unsaturated system. Introducing an input saturation provides a wider stability region.

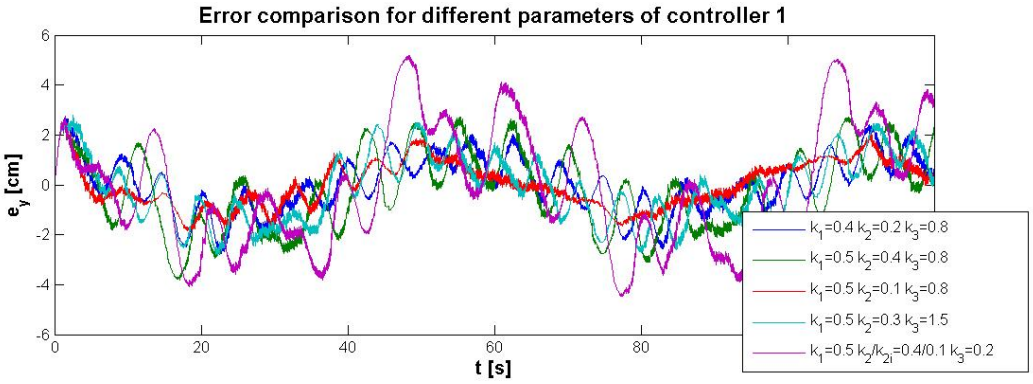


Figure 8.2: Position error (y component) for different controller parameters. k_2 seems to be mainly responsible for the oscillating behaviour and needs a sufficiently hight k_3 for dampening. The best result was achieved with $k_1 = 0.5$, $k_2 = 0.1$ and $k_3 = 0.8$.

Controller dynamics analysis

For approximating a good starting point of the controller parameters a simulative study was made. Several parameter combinations were tested and the resulting position error can be seen in figure 8.2. It turned out that k_2, correcting lateral errors, has to be chosen small since it easily leads to oscillations, even on straight trajectories. Here the factor k_3 dampens these oscillations and has be be chosen sufficiently high. When set too high ($k_3 > 1$) it also leads to oscillations due to the overshoot it is generating. k_1 appears to be a very robust parameter but achieved the best results around a value of 0.5.

8.1.2 Dynamic state feedback

Like the PI controller setup, position and orientation estimation is done by EKF 12. Furthermore the flatness-based controller needs vertical and horizontal speed v_x and v_y as input. These are determined either by EKF 9 or numerical differentiation with running average filtering.

Also in this case we first try to evaluate the delay influence on the closed-loop stability.

Delays and Stability

Also for the second controller simulations with constant increasing delays were performed for both models and can be seen in figure 8.3.
Interestingly the ideal model shows significant errors without delay, perhaps resulting from the large sampling interval of 150ms. For delays > 0 a small error can be observed which is rising with increasing delays. For the ideal model the critical delay could be identified to approximately 0.15 whereas for the extended model a delay of 0.12s leads to significantly large errors. In contrary to controller 1 destabilization here not emerges in form of oscillations but a growing lateral deviation. This also explains why the saturation here shows a negative effect on the performance.

Controller dynamics analysis

The two different methods for velocity estimation were compared in figure 8.4. The comparison was performed for two different controller dynamics which both showed only a slight dominance of the differentiation. In both cases the resulting controller error was comparable but due to the additional observer dynamics the EKF appears to cause bigger oscillations.
Furthermore various controller dynamics were tested on the extended model (fig. 8.5). Eigenvalues of -0.6 seemed to be critical as higher values lead to growing errors and oscillations. Here the strong corrections combined with delays lead to overshoots. The combination of -0.1/-0.2 and -0.3/-0.4 show the smoothest movement but lead to significant overshoot due to the slow controller reaction. The combination -0.1/-0.5 turned out to be the best parameter set.

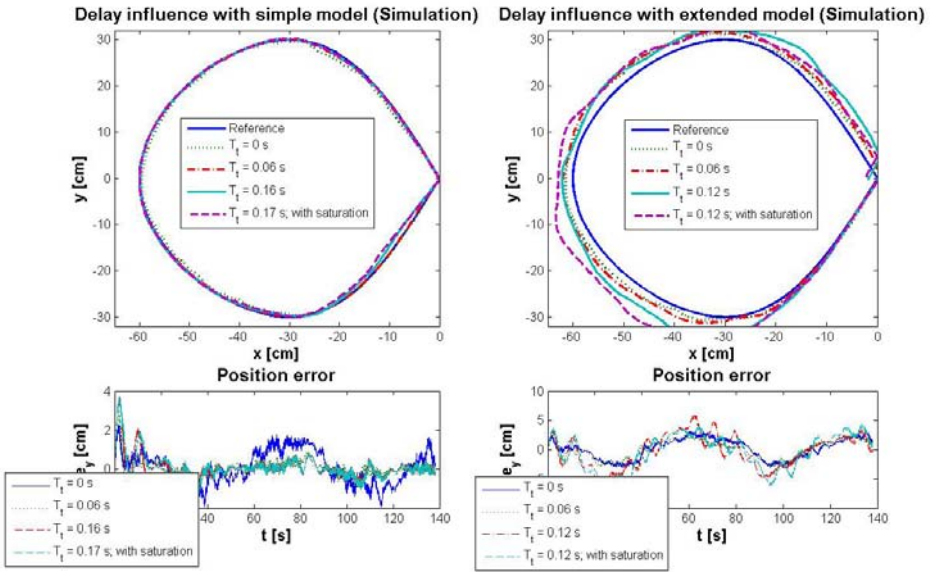


Figure 8.3: Delay influence on closed-loop system with controller 2. For the ideal model the controller can handle much higher delays than controller 1 (>0.16s). Applied to the extended model it shows severe deviation for delays > 0.1s. Here the system isn't getting unstable but reacts with a big lateral error. Furthermore saturating the input shows no positive effect.

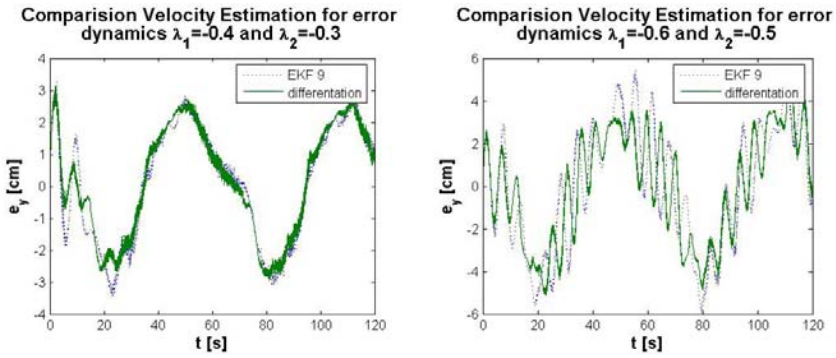


Figure 8.4: Error of the closed-loop system using differentiation and EKF 9 for velocity estimation. For the slower controller dynamics (left) almost no difference is visible. However for higher controller dynamics differentiation appears to produce less error, supposably due to it's faster convergence.

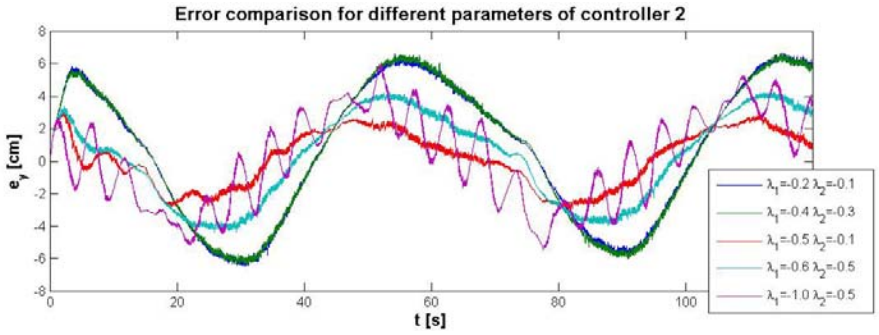


Figure 8.5: Position error (y component) for different error dynamics. Error dynamics with one $\lambda < -0.4$ seem to reduce the error significantly. For one $\lambda = -0.6$ and smaller the system begins to oscillate. The best parameter set seems to be $\lambda_1 = -0.5$ and $\lambda_2 = -0.1$.

8.2 Real experiments

8.2.1 PI controller

Parameters determined by simulations turned out to be a good starting point for practical experiments. Only the factor k_2 had to be set bigger (≈0.3-0.4). A small integral part for the lateral deviation seemed to compensate for static lateral errors.

Compared to simulation the oscillations were less severe supposably due to the speed limits and slip effects. Figure 8.6 shows two example results compared to the simulation. The second parameter set shows a better performance and less oscillations. Also the initial error is compensated with 2 overshoots. The second parameter set performs better than the simulation but shows significant overshoots in some situations.

8.2.2 Dynamic state feedback

Also for controller 2 experiments were performed. Figure 8.7 shows results for two different trajectories, the left one was chosen very smooth whereas the right trajectory includes many tight turns and varying curvature.

In the experiments the maximum stable controller dynamics were identified at eigenvalues of -0.5/-0.4. Furthermore differentiation turned out to be very critical in the initial phase

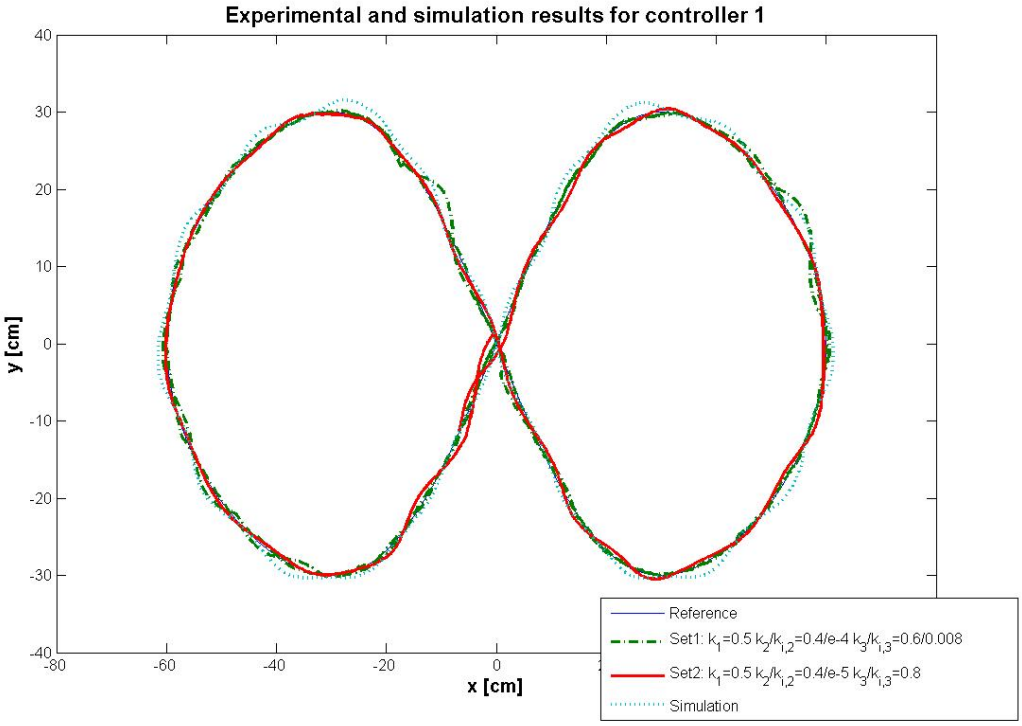


Figure 8.6: Experimental results with controller 1. Compared to simulation results with equal controller parameters, the real system produces less error. Especially set 2 shows only slight oscillation around the trajectory.

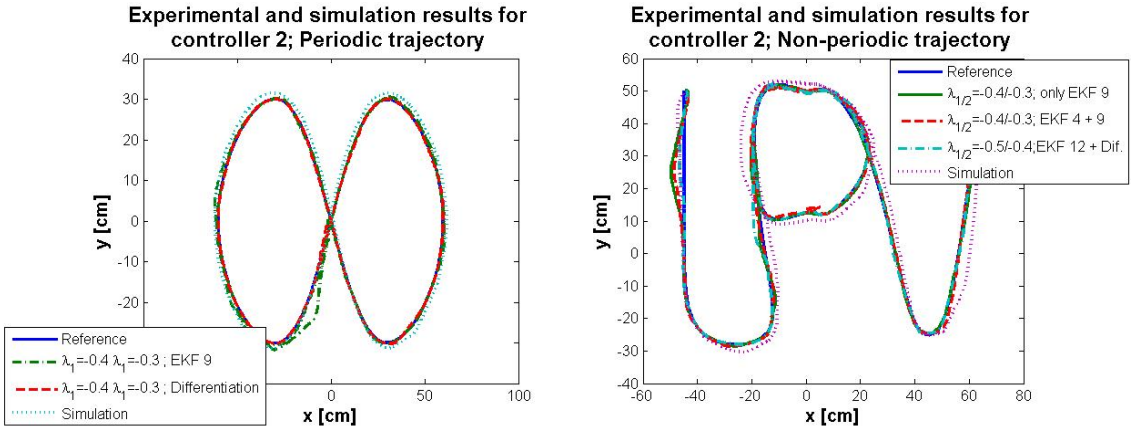


Figure 8.7: Experimental results for periodic and non-periodic trajectories with controller 2. Again the real system performs better than the simulated system. Using controller 2 the initial oscillations of the EKFs in some cases lead to a high initial overshoot. As trajectory 1 (left) is much smoother here a better performance can be observed.

and often lead to extreme overshoots (see left; set 1) that destabilized the closed-loop system.

The constant lateral deviation observed in simulation for these controller dynamics couldn't be verified (fig. 8.7 left). Especially the second parameter set with an EKF for velocity estimation showed a very good performance despite a significant initial error.

For the second trajectory the best performance was reached in experiments 1 and 3. In experiment 1 a severe overshoot because of initial EKF oscillation was observed. In experiment 3 a faster controller dynamic and differentiation for velocity estimation was tested and also performed very good. The initial error was compensated much faster and only slight deviations from the reference appeared. Only in one case a small overshoot was observed. Here again the big lateral error appearing in the simulation didn't occur.

8.3 Pathsearch experiements

To test and demonstrate the path-search algorithm discussed in chapter 6.2.2 some experiments with marked obstacles were made. Therefore plane obstacles of different shapes were

made by green paper and positioned in the vision field. Then the path search algorithm was started and tried to determine the shortest reachable path from the robots position to a given end position.
Here working with natural or interpolation splines turned out to be critical as the resulting curves are to tight what causes to high rotational speeds. As a consequence the reference track speeds increase above the possible maximum. Furthermore the rotation is too fast for the controller dynamics such that a big overshoot occurs.
To avoid these tight curves, smoothed splines were used und parameterized for creating a trajectory as close as possible to the waypoints with sufficiently smooth turns. The results of two tests can be seen in figure 8.8.

Regarding the given system, the presented results showed a surprisingly good controller performance and clear dominance of the non-linear controller. On the other hand problems caused by delays and the slow controller dynamics get more severe for trajectories with tight curves.
The mean speed with $\approx$ 3cm/s is not very fast but could not be increased more because peak velocities were close to maximum. A further increase would generate a reference speed above the saturation limit and thus cut the feedback. Moreover the delays' destabilizing effects get more severe with rising speed.

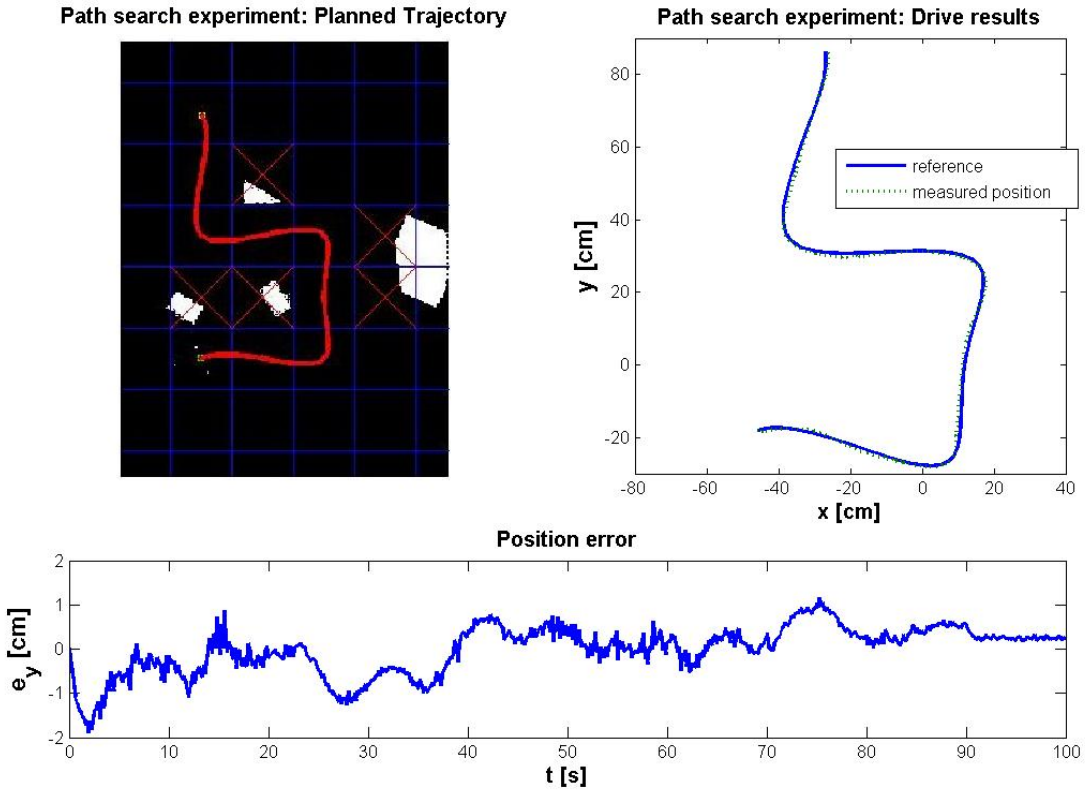


Figure 8.8: Experimental result for path search and trajectory control. The vehicle followed a trajectory along the calculated path (top,left). The deviation from the reference (top,right) was within a 1.5 cm range.

Chapter 9

Resumption and outlook

In course of the thesis presented in the previous chapters a control experiment was developed, using a Bluetooth-to-infrared adaptor developed at the Universidad Politecnica de Valencia to establish a cascade trajectory control of a LEGO mindstorms vehicle. Therefore a PID controller for the track speeds was implemented on the vehicle, whereas trajectory planning and two more complex controllers supported by an Extended Kalman Filter are running on a laboratory PC.

To accomplish this, adjustments for the firmware were made to provide reliable velocity measurements under consideration of the encoders systematic error. A tracked LEGO vehicle was constructed and programmed to control its track speeds and communicate with a PC via Bluetooth. Here simulation was used for searching PID parameters, supported by approximated friction and error modells.

Further firmware adjustments and a physical separation of the two infrared channels were made to achieve sufficiently fast sampling times for the measurements. For additional measurement of the vehicles position a JAVA image processing framework was created to detect a marker on the vehicle with a webcam. To transform the image data to real-world coordinates a calibration algorithm and GUI was developed in MATLAB.

For establishing trajectory control several types of EKFs were tested and evaluated supported by simulation. It lead to the awareness that the EKF based on the most simple kinetic model delivers the best results. Also attempts to compensate the errors induced

by transmission delays were tested, leading to slight improvements of the observer performance.
Two controllers were tested in simulation and lab experiments with good results. Due to the delays, performance in trajectories with sharp curves is problematic but smooth trajectories could be passed with an error of less than 1.5cm at a mean speed of 2.5-3 cm/s.

The project was developed with the objective to use it for educational purposes, as for example low-cost experiments in control engineering practicals. Therefore a framework was created to be used by student groups easily. So with little time consumption students can use the experimental setup to test their own controllers, image processing routines or velocity controllers. Some possible tasks students can solve by using the presented setup will be presented in the following.

Proposals for student labs

Preprogrammed movements with following error correction

A simple event-based system can be developed using preprogrammed basic movements as 'turn' and 'move'. After preforming the action, the error has to be determined and corrected by a correction action. Here LNP can be used and interesting tasks as model identification (e.g. encoder-distance ratio) have to be done. Since little knowledge about real-time control or control theory is needed this kind of work is perfect for undergraduate courses.

Event based trajectory control

Using the same principle as in the previous proposal, but working with much smaller intervals and including the error into the next movement, a kind of P/PI control can be realized as a next step.

Realtime trajectory control

As demonstrated in this thesis real-time control experiments can be performed, too. Student groups can develop and test different kind of controllers and observers and

have to deal with high uncertainties and delays. This kind of work is adequate for courses in advanced control engineering and mobile robotics.

Static collision avoidance

As also demonstrated here, trajectory planning can be combined with obstacle avoidances. Here again image processing on a less time-critical but perhaps more precise level can be the objective. Furthermore different kinds of path-search algorithms can be tested.

Dynamic collision avoidance

Broadening the previous problem to moving obstacles again leads to more time-critical and predictive algorithms and may be a problem formulation in (advanced) mobile robotic courses.

Time-critical image processing algorithms for position detection

In signal or image processing practicals simple but effective algorithms for position detection of the vehicle can be developed.

Robust velocity or position PID control for systems with high uncertainties

PID control as a typical problem in control engineering can be trained here on a very noisy systems with problematic measurements. This also is adequate for basic control engineering courses.

Moreover some projects can be proposed to enhance the described system or its abilities.

Proposals for further works or projects

Separation of the IR channels

By extending the BT adaptor to two IR channels with different frequencies the tube can be made redundant and sporadic signal noise can be reduced.

Path search

For path search alternativ methods can be implemented and tested. Furthermore the presented method can be extended to continuous sampling and a followed distance

verification as in the presented method the spline function may get too close to the obstacle such that the vehicle may collide while turning.

Camera calibration

Camera calibration can be automated with an algorithm for searching the calibration pattern automatically. This will need further image processing techniques as edge detection and following for instance.

Interactive systems

The system can be extended to multiple vehicles which demands a certain level of interaction. Agent systems can be used to achieve this and perform common tasks as for example area sweep or team games.

This thesis showed that also with 'low quality' and low-cost equipment as LEGO Mindstorms serious control experiments can be realized. Especially the new Mindstorms generation 'NXT' with better sensors should widen the range of possibilities a lot. Thus this provides a cheap and interesting way to bring a little more practise into university everyday life and motivate students. I hope a lot of students will enjoy working with my experiment buildup and thereby learn much about control systems.

Appendix A

CRC-Example

A.0.1 a worked out calculation

Appendix B

Controller implementation

Figure B.1 shows the Simulink plan used for the real-time control. The MATLAB function getWebcamData and getRCXData used the Java clients to obtain the received data. The webcam data then is processed further to get the real world coordinates by using the transformation matrix MT.

For preventing Simulink to run as fast as possible and in realtime a freeware tool called RTBlock (XXX source) was used. This tool can be set into the plan like any Simulink block and parameterized with priority and sample time.

For sending the reference speed to the vehicle a UPD-block, included in the MATLAB xPC-Toolbox was used. First the data has to be packed in the corresponding form (here the signed bytes) into a UDP datagram. Then it is send by the UDP-sender block to the declared IP address, the address of the PC running the JAVA RCXServer class.

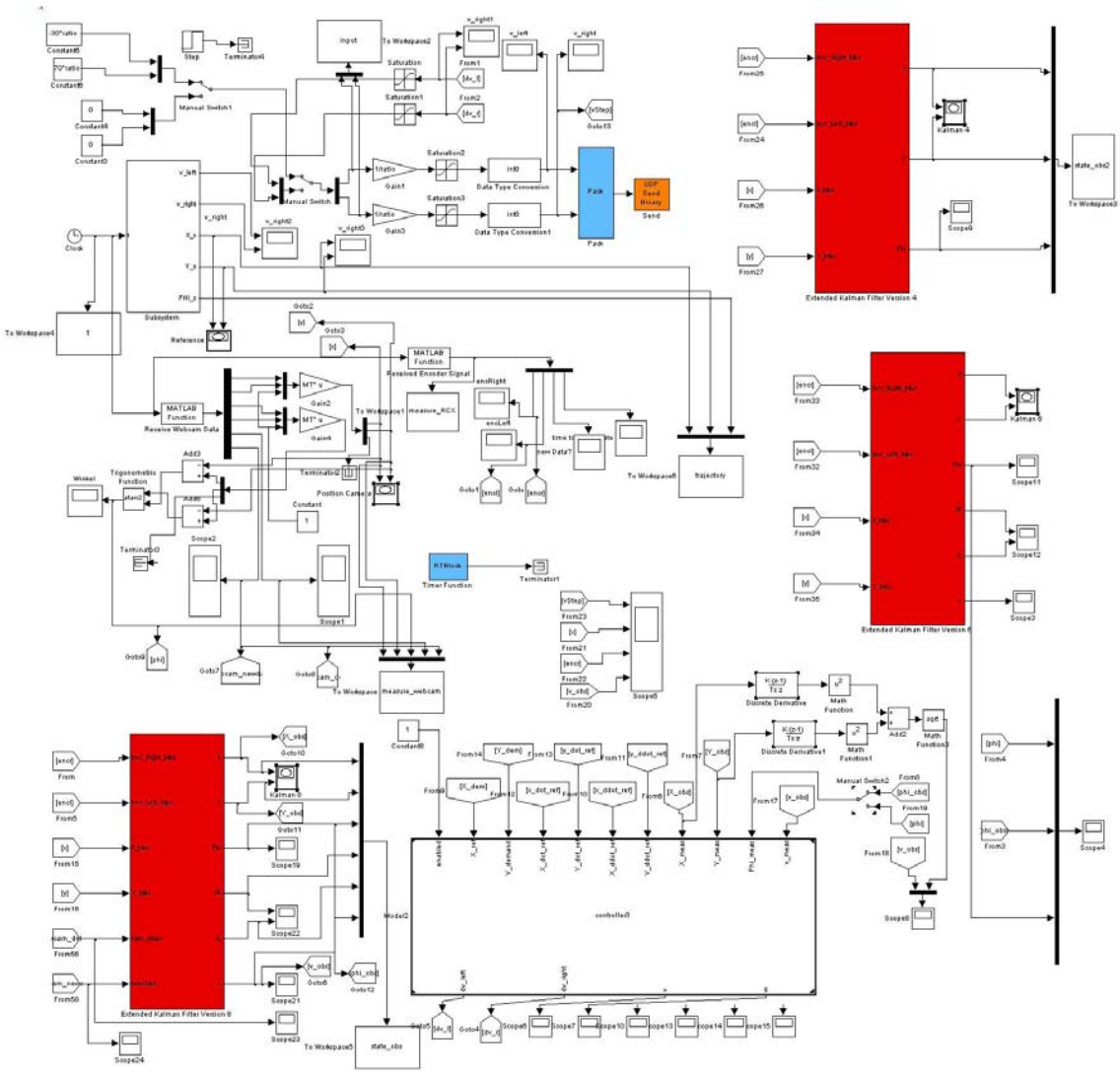


Figure B.1: blabla

List of Figures

List of Tables

Bibliography

[1]

[2] Brickfilms.com message board: Details lego webcam, http://www.brickfilms.com/phpbb2/viewtopic.php?p=484&. Stand: 01.03.2006.

[3] Brickos homepage and manual, sourcefourge , http://brickos.sourceforge.net/. Stand: 01.12.2005.

[4] Bricx command center homepage, sourcefourge , http://bricxcc.sourceforge.net/. Stand: 01.12.2005.

[5] Cisco systems, atm over e1 framing formats on ima interfaces; http://www.cisco.com/warp/public/121/e1framing.html#crc4. Stand: 10.02.2006.

[6] Desarrollo y programación de aplicationes con la plataforma lego mindstroms, course note. Stand: 5 / 2005.

[7] Hitachi, single-chip microcomputer h8/3292 hardware manual, 3rd edition. Stand: 01.03.2006.

[8] Homepage of philippe hurbain: Lego sensor details, http://www.philohome.com. Stand: 01.12.2005.

[9] Hypermedia image processing reference, department of artificial intelligence university of edinburgh, uk , http://www.cee.hw.ac.uk/hipr/html/label.html. Stand: 01.04.2006.

[10] kalman tracked vehicle. Stand: 01.03.2006.

[11] Lego Mobile Robot.

[12] Lego press releases,
http://www.lego.com/eng/info/default.asp?page=pressdetail&contentid=17278&countrycode=20
Stand: 01.03.2006.

[13] Lego robotics internals,
http://www.crynwr.com/lego-robotics. Stand: 01.12.2005.

[14] Lego robotics internals,
http://www.nabble.com/re:-rcx-question-t606566.html. Stand: 01.03.2006.

[15] legos manual by luis villa,
http://brickos.sourceforge.net/. Stand: 01.12.2005.

[16] Matlab spline toolbox documentation. Stand: 01.03.2006.

[17] Robotics - part 7: Trajectory generation, course notes ,
http://users.rsise.anu.edu.au/ chen/teaching/robotics_engn4627_2005/lecturenotes/engn4627-part07.pdf. Stand: 01.03.2006.

[18] Rsa laboratories, what is a hash function;
http://wwwrsasecurity.com/rsalabs/node.asp?id=2176. Stand: 10.02.2006.

[19] siehe welch & bishop. Stand: 01.03.2006.

[20] Welch & bishop. Stand: 01.03.2006.

[21] Wikipedia: Affine transformation ,
http://de.wikipedia.org/wiki/affine_transformation. Stand: 01.03.2006.

[22] Wikipedia: Network time protocol , http://en.wikipedia.org/wiki/network_time_protocol.
Stand: 01.03.2006.

[23] Wikipedia.de :zyklische redundanzprüfung, http://de.wikipedia.org/wiki/zyklische_redundanzpr%c3%bcfung. Stand: 01.03.2006.

[24] Xxx ,. Stand: 01.03.2006.

[25] M. Vallez A. Valera, M. Weiss and J.L. Diez. Control of Mobile Robots using Mobile Technologies.

[26] M. Vallez A. Valera, M. Weiss and J.L. Diez. Proposal of a Low-Cost Mobile Robot Control Laboratory Experiment.

[27] L. Akin. Investigation of Cooperative Behavior Among Small Mobile Robots, project-report, http://www.wam.umd.edu/ evren/robotics/index.html. Master's thesis, Bogğaziçi University, Computer Engineering Department, 1999.

[28] N. Amato. Coursenotes to Randomized Motion Planning - Section: Probabilistic Roadmap Methods. University of Padova, 2004.

[29] XXX Übersicht nichtlin. Beobachter. .

[30] Ralph Hempel Dave Baum, Michael Gasperi and Luis Villa. *Extreme MINDSTROMS: An Advanced Guide to LEGO MINDSTORMS*. Springer Verlag, Berlin, Heidelberg, 2000.

[31] A. Encimas. Bluetooth rcx adaptor. Universidad Politechnica de Valencia,, 2005.

[32] P.J. Gawthrop and E. McGookin. A lego-based control experiment. *IEEE Control Systems Magazine*, 24(5):43–56, October 2004.

[33] Eddins S.L. Gonzalez R.C., Woods R.E. *Digital Image Processing Using MATLAB*. Prentice Hall, Upper Saddle River, New Jersey, 2004.

[34] Jan Lunze. *Regelungstechnik 1, 3. Auflage*. Springer Verlag, Berlin, Heidelberg, 2001.

[35] David L. Mills. Internet time synchronization: The network time protocol. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press*. 1994.

[36] C. García Morey. Diseño, implementación y validación de controladores sobre sistemas empotrados. Aplicación a robots móviles. Master's thesis, Universidad Politechnica de Valencia, Esculea Technica Superior de Ingenieria Industrial, 2005.

[37] I. R. Nourbakhsh R. Siegwart. *Introduction to Autonomous Mobile Robots*. Campridge MIT Press, MA, 2004.

[38] H. Stetter. Messtechnik and Maschinen und Anlagen vorlesungsskript. University of Stuttgart, 2000.

[39] H. Wehlan. Digitale Bildverarbeitung - Zusammenfassung zur Vorlesung. University of Stuttgart, 2002.

[40] H. Wehlan. Echtzeitdatenverarbeitung - Zusammenfassung zur Vorlesung. University of Stuttgart, 2002.

[41] R. N. Williams. A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS. Rocksoft^{TM}Pty Ltd., 1993.