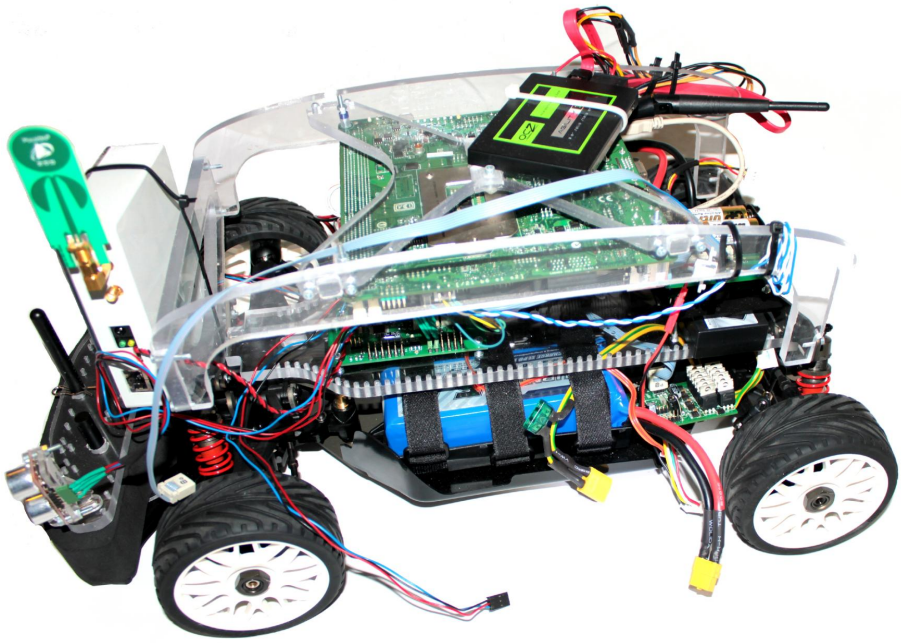
CHALMERS



Gulliver: Design and Implementation of a Miniature Vehicular System

Master of Science Thesis in the Programme Communication Engineering

BENJAMIN VEDDER

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, November 2012

Gulliver: Design and Implementation of a Miniature Vehicular System

Benjamin Vedder

Examiner: Elad M. Schiller

Chalmers University of Technology University of Gothenburg Department of Computer Science and Engineering SE-412 96 Göteborg Sweden Telephone + 46 (0)31-772 1000

Cover: Photo of the Gulliver 1:8 Car with the 9-degrees-of-freedom sensor removed.

Department of Computer Science and Engineering
Göteborg, Sweden, November 2012

Abstract

Traffic increases all over the world and problems such as toxic emissions and congestion are consequences of that. To encounter those problems and to optimize traffic throughput and safety, a miniature vehicular test platform that can be used together with simulation tools can be very useful.

This project deals with the construction, control, communication and navigation of the autonomous 1:8 scale miniature vehicles used in the Gulliver project. The goal is to make the vehicles drive along a generated route, do lane changing between routes and to write software to interface with these vehicles.

Part of the work has been spent on a custom motor controller used to control brushless DC motors, for which many safety mechanisms were implemented. Other areas covered include the fusion of several sensors with Ranging and Communication Modules to achieve indoor localization and algorithms to drive the vehicle autonomously along a pre-defined route while accepting high-level commands such as stop, go and change lanes from external sources. Another part of the project was to evaluate the high-level commands using MICAz motes that run a virtual service agent to act as virtual traffic lights.

At the end all of the goals were achieved and the vehicles were able to drive along a route and accept commands from the MICAz motes. This is very useful as many details that not always are covered by pure simulations can be included while it is much easier and less costly than the same implementation with full-scale vehicles.

Acknowledgements

I would like to thank Elad Michael Schiller and Roger Johansson for their guidance during this project, it would not have been possible for me to achieve the same result without their help. Further, I would like to thank Mohamed Mustafa and Mitra Pahlavan for their help with the MICAz motes and their participation with the evaluation of the system. It is thanks to them that the entire system could be evaluated with a simple realistic scenario at all.

There were also many bachelor students involved in this project, and I would like to express my gratitude to all of them:

Alexander Altby
Tobias Boström
Erik Dahlgren
Johan Grundén
Daniel Gunnarson
Nadia Holtryd
Anmar Khazal
Timur Sibgatullin
Karl Stjärne
Viktor Swantesson

Further, I would like to thank everyone at Chalmers Robotics Society (CRF) who participated in the construction of the Gulliver vehicles:

Peter Kaldén
Erik Sternå
Nigul Ilves
Mikael Tulldahl
Michael Nilsson

I would also like to thank Henk Wymeersch and Gabriel Garcia for their expertise regarding the localization of the vehicles. Further, I would like to thank Brandon Dewberry from TimeDomain for visiting us at Chalmers and for holding a seminar about the P400 RCMs.

It was privileged and honoured for me to work with so many brilliant people during this project.

Benjamin Vedder - Wednesday 14th November, 2012

Glossary

ACK acknowledgement. 26

ADC Analog-to-Digital Converter. 13

BEMF Back Electro-Motive Force. 6, 13

BLDC Brushless Direct Current. 5, 6, 13, 14

Bluetooth is a wireless technology standard for exchanging data over short distances. 24

commutation is the act of switching the polarity between different windings in an electric motor to make the electromagnetic force contribute to the same angular direction at all times. 5, 6, 9, 15

CRC Cyclic Redundancy Check. 16, 17, 25, 26

DC Direct Current. 5

dead reckoning is (in this project) the process of estimating a vehicles position by advancing the previous position based on data from encoders in the wheels and the steering angle. 6, 7, 10, 21

DOF Degrees of Freedom. 10, 21, 24

encoder is a device that converts mechanical movement into electrical pulses. i, 5, 9

GUI Graphical User Interface. ii, 5, 7, 8, 25, 26

IDE Integrated Development Environment. 8

Linux (or GNU/Linux) is a Unix-like computer operating system that uses the Linux kernel. 7

MAC Media Access Control. 3

MATLAB (matrix laboratory) is a numerical computing environment. 13, 15

MICAz is a 2.4 GHz Mote module, which is able to exchange data with other MICAz motes. 4, 11, 24, 28, 31

mini-ITX is a 17x17 cm motherboard form factor. 9, 11, 24

MOSFET Metal–Oxide–Semiconductor Field-Effect Transistor. 5

PCB Printed Circuit Board. 13

PID Proportional–Integral–Derivative. 15

PWM Pulse Width Modulation. 13–15

Qt is a cross-platform application framework that is used for developing software with a Graphical User Interface (GUI). In addition to the GUI, Qt also provides functionality in many other areas, such as Transmission Control Protocol (TCP) connections (see http://qt.nokia.com/). 4, 7–9, 18, 23, 25, 26

RC Radio Controlled. 1, 5

RCM Ranging and Communications Module. 2, 3, 7, 11, 12, 16, 21, 24, 30

RF Radio Frequency. 24

route is in this document defined a set of points that define a path for vehicles to follow. 2–4, 21–23

TCP Transmission Control Protocol. ii, 7, 11, 24–26

UART Universal Asynchronous Receiver/Transmitter. 10, 11, 13, 15–17, 24

USB Universal Serial Bus. 24, 25

UWB Ultra Wideband. 11

Wi-Fi is a technology that allows devices to exchange data wirelessly over a computer network. 25

Contents

1

Introduction

As traffic increases all over the world it is required to increase the capacity of the roads. The traditional way to do this has been to build more roads, but this alone will not be enough in the long run, which is why new approaches are required. One way to conquer this is to simulate different traffic scenarios for which we could implement traffic coordination algorithms to increase traffic safety and throughput, however, simulation alone will not be enough to cover every detail that has to be considered before implementation on real traffic. The next step, after simulation, is traditionally to test and verify the simulation on physical full-scale vehicles. This tends to be very expensive in many aspects, making this step impossible for most organizations due to cost and space restrictions.

The difficulty in going from simulations to physical full-scale vehicles was addressed in [1], where a method to make this process easier was proposed: the creation of a platform with physical *miniature* vehicles where large scale experiments could be conducted together with simulations. Thus, this project deals with the construction of miniature vehicles for a miniature vehicular system.

1.1 Goal of this Work

The goal of this project is to participate in the construction of a small scale platform with physical miniature vehicles to test different traffic scenarios. These vehicles are common 1:8 scale [1] common Radio Controlled (RC) cars modified for this purpose, where the modifications include, but are not limited to, the following points:

[1] This scale refers to the way common radio controlled model cars are scaled.

- Localization for all the vehicles that is accurate and fast enough to make it possible to drive the vehicles autonomously.
- The vehicles should be able to accept high level commands such as stop, go and change lane. Moreover, remote programming of main functionality should be allowed.
- Possibility to remote control the vehicles and to record routes should be present. This means that one should be able to drive one vehicle and record the driven path, and store it as a route.
- It should be possible to upload the routes to a computer, change and download them to the vehicle again. It should also be possible to download routes to the vehicle that are completely generated by software on a computer.
- On-board sensors on the vehicle should avoid driving into obstacles even if the navigation and control software claim that there is nothing in the way. This reduces the necessity to externally monitor the area where the vehicles drive to avoid unexpected objects.

1.2 Previous and Related Work

Two vehicles were constructed, assembled and partly tested in another project [2, see Appendix A] in collaboration with the work of this project. Parallel to this project, other projects on different aspects of these vehicles and their integration into other systems were also carried out, where one project focused on improving the localization of the vehicles [3] and the other one focused on integrating the Gulliver vehicles into other systems [4].

1.2.1 The Initial Construction of the Gulliver Vehicles

Part of this project was to participate in a project where the initial construction of the Gulliver vehicles was conducted [2, see Appendix A]. In this project, two vehicles were constructed and very basic software was written to test the functions of the different systems on the vehicles. At the end of this project, one vehicle was able to follow a hard-coded route to some extent.

While this was a great step in the progress of constructing vehicles as proposed in [1], there were many things left to be done in order to reach the goal. The following list points out a number of those issues, which this project report will focus on:

- When two vehicles were driving at the same time they interfered with each other and the accuracy for the position decreased. When switching on another one of the Ranging and Communications Modules (RCMs), effectively adding another

vehicle to the ranging network, the localization got too much interference due to collisions that navigation became impossible. This issue will mainly be addressed in [3], however, there was much collaboration between this project and their work.

- The localization had a very simple algorithm that did not consider the quality of the measurements from the different sources even if the information was there. For instance, the RCMs provide an indication about the quality of each range measurement [5] that was not taken into account. Like the previous issue, this will be addressed in [3] with collaboration from this project.
- There was no way to upload new routes to the vehicles while they were driving, so in order to change the route the firmware had to be changed by connecting a programming cable and uploading it. As the routes (and the firmware) were changed often during the experiments much time was used for picking up the vehicles, reprogramming them, and putting them back on the floor. This is one of the areas that was focused on during this project by implementing the necessary communication links and computer programs to remotely reconfigure the vehicles.
- An RF-transceiver was used to communicate with one vehicle at a time. There was no way to receive data from more than one vehicle at a time and the transmission power was low enough to cause lost packets frequently. Like the previous point, this was addressed during this project.
- The motor controller didn't have enough "sanity checks" for the received commands, which caused defective electronic components when certain errors were present in the program of the main controller on the vehicle. One of the main tasks of this thesis was to improve the motor controller and fix these issues.

1.2.2 The Localization Project

The localization project [3] was carried out in parallel to this project. This project's purpose was to improve the localization on the vehicles in two different areas: improving the algorithm for the position estimation and making the localization work with many vehicles at the same time.

In order to make the position estimation more accurate a Kalman filter [6] was used. This filter merged data from the RCMs and the on-board sensors of the vehicle to estimate the vehicle current position and a confidence interval around that position.

To coordinate the ranging of the vehicles in a way that avoids collisions was the other major part of this project. For this purpose a custom Media Access Control (MAC) layer was created that uses one or multiple communication units on the vehicle to keep a global clock and coordinate the ranging operations. The so-called timeslot manager adapted the timing scheme for the ranging dynamically when new vehicles entered or left the area.

1.2.3 The Integration Project

In the integration project [4], the Gulliver vehicles were connected to MICAz motes used to interact with the SUMO simulator [7]. This was carried out in parallel to this project and the external interface to the vehicles was tested together with this project. One part of this project was to drive simulated vehicles together with the physical vehicles on the same virtual map and make the simulated and physical vehicles aware of each other. At the end this worked quite well and a demonstration of this was made [4].

Another part of this project was to work on a map editor to create routes for the vehicles. This was done in close collaboration with the work on the external interface written in Qt during this project.

1.3 Delimitations

The miniature vehicle proposed in [1] leave room for a large number of possibilities, and obviously not all of them can be implemented in full detail the scope of this masters thesis. Therefore, it is important to make it clear that only the most important features necessary to show the basic concept of the vehicles were implemented. Also, the features that were implemented, such as the localization and navigation algorithms, leave room for further development. Further, the system will not be tested with a large number of physical vehicles due to resource and time limitations.

Another thing to point out is that this paper does not cover all the details of the implementations made, but only the concepts and the most important details. There are approximately 25000 lines of C/C++ code written for the microcontrollers and the desktop computer programs. The entire design of the printed circuit boards, source code and a construction tutorial can be retrieved via the Gulliver website [1].

[1] www.gulliver-testbed.net.

2

Theory and Related Studies

This chapter covers fields and studies related to this project. Many decisions regarding the design and implementation of different systems on the vehicles were made based on the material that is reviewed and compared here. More precisely, this chapter covers: background about the Gulliver platform, Brushless Direct Current (BLDC) motors, localization and Graphical User Interface (GUI) programming.

2.1 The Gulliver Platform

In order to determine what is required by the Gulliver vehicles, it is important to study the environment in which they are going to operate. As is described in [1], the vehicles are going to work together with a *motion manager* (the MICAz mote, see Chapter 6 for further details) which will provide high-level commands to perform different maneuvers. Therefore, it is important that the vehicles are aware of their own location and have algorithms to perform motion planning. To make it convenient to work with the simulator mentioned in [1] and to debug the vehicles, it is also important to have an interface where maps and routes can be edited and visualized. It is also important to plot the vehicles on these maps in real-time to see how they perform and what is observed from their perspective. The GUI that is developed in this project to control the vehicles can be compared to the *base station* described in [1].

With this considered, the additions required for a common RC car to suite the Gulliver platform are:

- A good motor controller to give precise control of the maneuvers of the vehicles.

- Hardware and software able to carry localization and navigation in real-time.
- The necessary interface software to control and debug the experiments made with this platform.

How all of this fits together will be presented in this project report.

2.2 Basics of Brushless Direct Current Motors

For model RC cars and other smaller motor-driven applications it is common to use brushed Direct Current (DC) motors because they are easy to control, however, they have a few drawbacks compared to BLDC motors:

- They require more maintenance because the brushes wear out.
- Their power density and efficiency is lower.
- Since the commutation is done mechanically, brushed DC motors cannot be used as pulse encoders. The angular speed of the motor can only be estimated by measuring the current and voltage, unless an external encoder is used.

In comparison, the BLDC motor does not suffer from those issues. With a custom motor controller one has good knowledge about the angular position of the rotor and the motor requires very little maintenance since there are essentially no parts that wear out. The drawback is that those motors are more difficult to control. The reason for that is that instead of mechanically switching between the different windings for the commutation, as is the case with the brushed motor, the windings of the motor are stationary and the commutation is done electronically with Metal–Oxide–Semiconductor Field-Effect Transistors (MOSFETs). In order for this to work the position of the motor has to be known at all times, which is a problem that has two possible solutions:

1. **Sensored commutation:** sensors mounted in the motor are used to detect its position.
2. **Sensorless commutation:** as one of the connections of the motor is always floating, i.e., is not connected to any of the power rails, it can be used to detect the position of the motor by measuring the Back Electro-Motive Force (BEMF) of the motor.

2.2.1 Sensored Compared to Sensorless Commutation

Sensored commutation works very well from startup of the motor until high speeds without glitches and is relatively easy to implement on the motor controller; which is because the sensors directly provide the position of the rotor and it is easy to read them. The problem is that it only works for BLDC motors with sensors, which are

a bit more difficult to find and more expensive. On the contrary, a motor controller designed for sensorless BLDC motors works with any motor and requires less wires and less components. The drawback is that it is harder to implement the motor controller and most methods used to control the motors have limitations during startup and low speeds [8, 9]. This is because the BEMF created by the motor, used to detect its position, is proportional to the rotational speed of the motor and hence non-existent when the motor is not moving.

The most common way to start sensorless BLDC motors, as proposed in [8, 9], is to use a known sequence of commutations based on knowledge about the motor and its load, and then switch to closed loop control when the motor speeds up. There are also more advanced ways to start BLDC motors that rely on heavier calculations and more accurate measurement of the BEMF [10] where the start is smoother.

2.3 Localization and Navigation in other Mobile Robot Platforms

Indoor localization and motion planning are common issues when dealing with mobile robot platforms. One platform where these issues were encountered by the developers is the autonomous robot Blanche [11]. Blanche is a three-wheeled cart designed to navigate autonomously in structured office environments. Blanche navigates based on dead reckoning and on-board sensors which are combined to estimate its current position. It is assumed that a path, consisting of straight lines and circle segments, is provided in advance for the cart to follow. The cart controller will continuously update the motor output power and steering angle based on the reference position and feedback from the motor odometry and a model for the movement of the cart.

The concept of combining different sensors to estimate the current position will be implemented for the Gulliver vehicles as well. The dead reckoning will be attempted almost identical to the way it is done on Blanche, however, the RCMs will provide a more accurate position than the other sensors on Blanche.

2.3.1 Motion Planning

The path for the Gulliver Vehicles consists of a set of points and the vehicles themselves make decisions on how to drive to the next point based on the current position, performing simple motion planning. One way to do motion planning, as suggested by [12], is to calculate a number of paths to reach the goal and pick the best one based on minimizing the cost to of the selected path. This optimization takes the distance, speed and energy consumption into account and many details on how to do this are provided. Even though the areas covered in this paper are relevant for the Gulliver project, most of them are not implemented during the work in this project as it would require considerably more

time than there is available. It should however be noted that more advanced motion planning, as suggested in this paper [12], would be very useful for work on the Gulliver vehicles in the future.

2.4 GUI and Network Programming

As a major part of this project involves writing a GUI to control the vehicles and communicating over a Transmission Control Protocol (TCP) network, it is important to choose a toolkit that can handle those tasks. The work will be performed using the Linux operating system, so it is important that the chosen toolkit will run on Linux. Three modern and common such toolkits are GTK+ [13], Qt [14] and WxWidgets [15]. These toolkits will be compared in the areas of functionality and how easy they are to use.

2.4.1 The Functionality of the GTK+, Qt and WxWidgets Toolkits

Regarding the ability to draw GUI components (or widgets), all toolkits are able to do the job. Qt and GTK will draw widgets by themselves trying to emulate the behaviour of the operating system, while WxWidgets will use the native widgets provided by the operating system. For comparison, a few aspects of the languages will be considered:

Multi-threading, which is important because network communications often take relatively long time, is supported on GTK+ and WxWidgets to some extent; however it has to be handled carefully and can cause many problems [13, 15]. On the contrary, Qt has been designed with multi-threading in mind and threads can easily be synchronized with not only mutexes, but also with signals and slots [14]. This gives Qt an advantage regarding multi-threading.

Network Support, which is built into WxWidgets and Qt. GTK+ lacks built-in libraries with network support. However, external libraries could be used to provide this functionality [1]. When comparing the network support in Qt with WxWidgets, Qt's network classes are more feature rich and provide better multi-threading support [14, 15]. This, again, gives Qt an advantage.

2.4.2 Ease of Use and Documentation for the GTK+, Qt and WxWidgets Toolkits

Both GTK+ and Qt have excellent and updated documentation available online which makes it obvious where to get started [2]. Qt also has an Integrated Development Environ-

[1] see http://developer.gnome.org/gnet/.

[2] see http://doc.qt.nokia.com/ and http://www.gtk.org/documentation.php.

ment (IDE) with built-in documentation and examples, called QtCreator [3], which makes it very easy to find documentation during the development process. On the contrary, WxWidgets lacks in some areas of the documentation and often refers to a book that is seven years old at the time of this writing [15]; it is not obvious how to get started when using WxWidgets. Thereby, GTK+ and Qt have an advantage over WxWidgets in the area of documentation.

It should also be noted that both GTK+ and Qt provide a graphical GUI designer where components easily can be dragged and dropped [13, 14] to create the GUI, whereas WxWidgets doesn't provide any open-source GUI designer.

At the end, it was decided to use the Qt toolkit to write the GUI and the network interface as it seems to have the most advantages for this task.

[3] see http://qt.nokia.com/products/developer-tools.

3

Brief System Overview

In this chapter a brief description will be given about each one of the subsystems on the Gulliver vehicles in order to provide a picture about the whole system, to make it easier to follow later when the subsystems are described in detail. An overview of the system on the vehicles can be seen in the block diagram in Figure 3.2. Each vehicle consists of a network of nodes connected over the same, or independent, communication buses. Some of the nodes are connected to actuators and others are connected to sensors. Where everything is located on the physical vehicles can be seen in Figure 3.1. The system of several vehicles and anchors along with the distances measured by ranging can be seen in Figure 3.3.

The work in this project involves every system described in this section to some extent; in particular the main controller, the motor controller and the mini-ITX computer. A lot of effort was also spent into writing a program using the Qt framework to communicate with and control the Gulliver vehicles.

3.1 The Motor Controller

This node is responsible for controlling the main motor and the steering servo. The main motor has built-in sensors used to detect the position of the motor for commutation, which additionally are used as encoders for sensing the travelled distance. The steering servo has an internal control system for the positioning of the front wheels. The motor controller only communicates with the servo in one direction when setting the desired steering angle and the rest of the system on the vehicle will assume that the servo is at the requested position at all times. Because of the importance of the motor controller it has it's own communication bus.

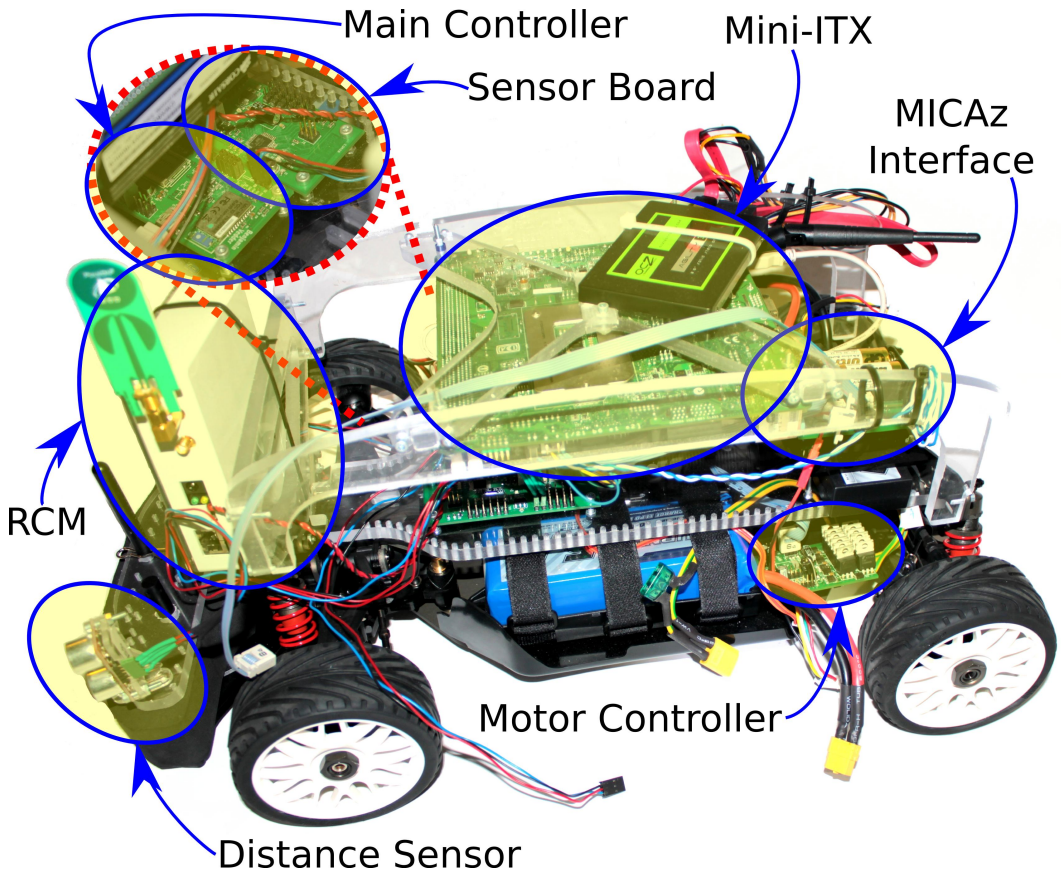


Figure 3.1: Location of the different components on the Gulliver vehicle. The 9-Degrees of Freedom sensor is not attached in this picture. It is located on a stick attached to the vehicle, so it can easily be seen on the real vehicle.

3.2 The Main Controller

The main controller communicates with the other nodes on the vehicle and runs the algorithms for localization, dead reckoning and sensor data fusion. Further, the algorithms for autonomous driving (or navigation) of the vehicle also run on this one.

3.3 The Sensor Board and the 9-Degrees of Freedom Board

The sensor board and the 9-Degrees of Freedom (DOF) board are responsible for collecting data from external sensors on the vehicle and applying some rough scaling and filtering to it. These boards are connected to the same multi-Universal Asynchronous Receiver/Transmitter (UART) bus.

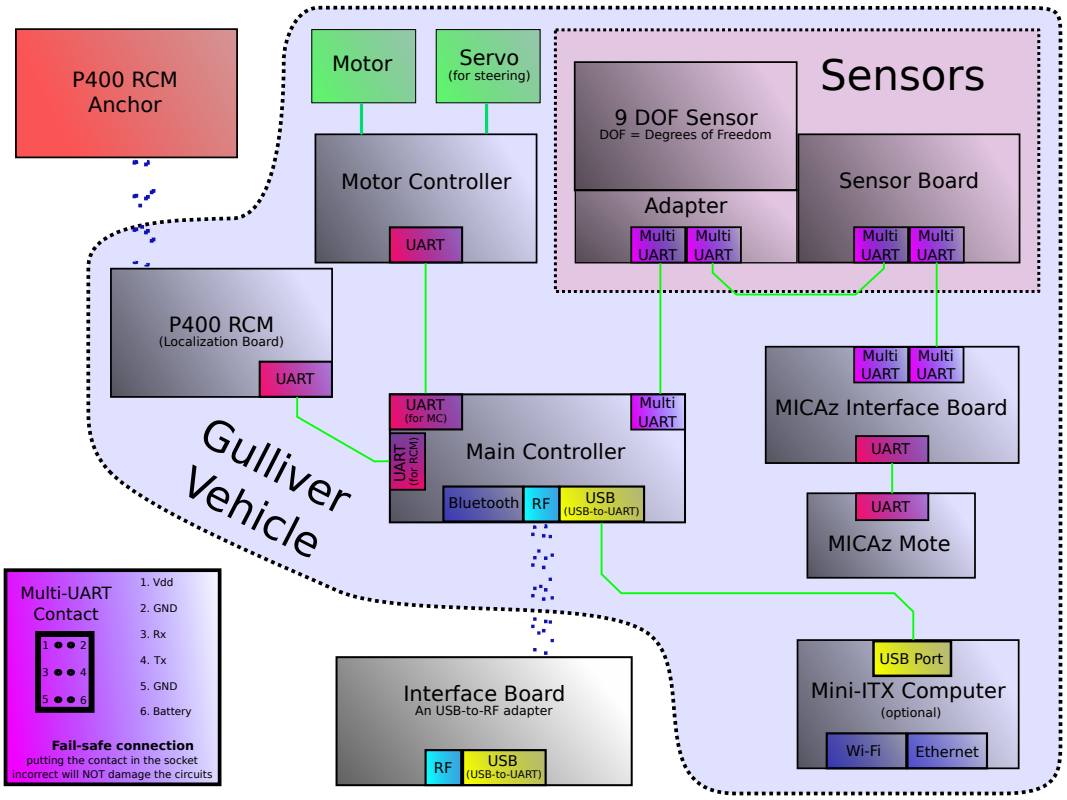


Figure 3.2: Overview of the control system on the vehicles

3.4 The Ranging and Communication Module

The RCM [16] is connected directly to the main controller via UART and is used to measure the distance to fixed anchors on the area where the vehicle navigates, thereby providing an absolute position reference. The RCMs use Ultra Wideband (UWB) radios to measure the time-of-flight of the signal to calculate the distance between them.

3.5 The MICAz Interface

This is an interface board used for communicating between the Gulliver Vehicle and MICAz motes. Many of the people who are involved with this project are familiar with the MICAz motes, so therefore it is important that they can be used with the Gulliver vehicles.

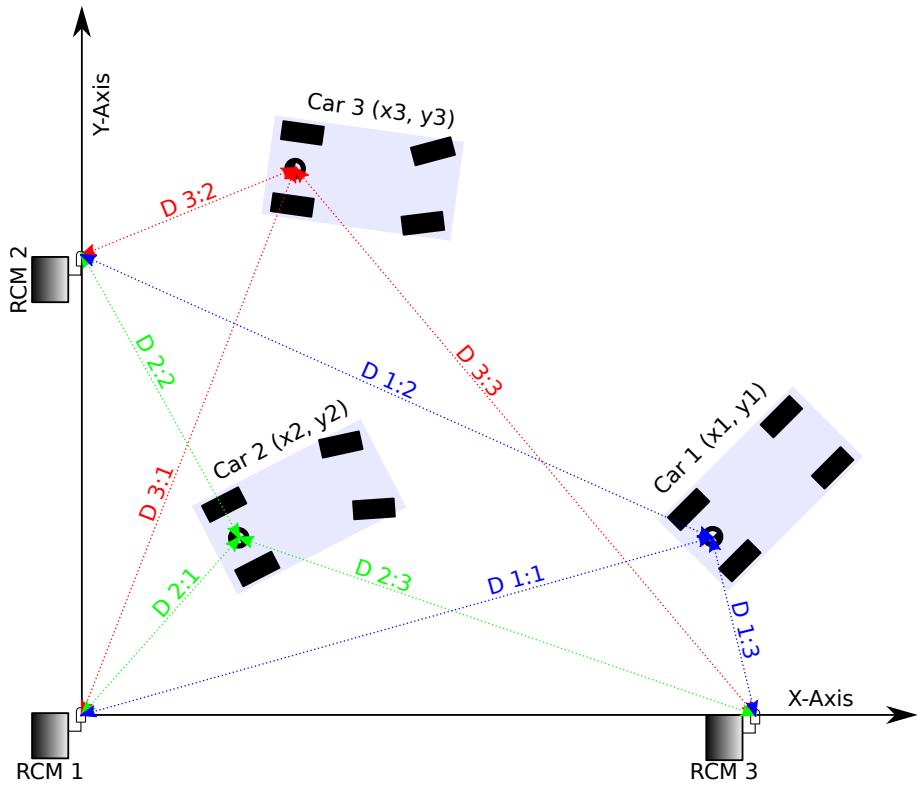


Figure 3.3: The whole set-up with the Gulliver vehicles, the anchor RCMs and the measured distances.

3.6 The Sensors Board

This module is connected to a set of distance sensors that are used to detect obstacles close to the vehicle. A number of proximity sensors are connected to this module and their data is filtered and converted to meaningful units.

3.7 The External Interface Board

This module can be used to establish a simple wireless connection between the vehicles and any computer. It can be used if the optional mini-ITX computer is not mounted on the vehicles, however, it has many limitations which are described later.

3.8 The Mini-ITX Computer

The mini-ITX computer is optional and can be mounted in order to provide a convenient interface to monitor, control and re-configure the vehicles wirelessly. This computer is connected to a wireless network and to the main controller of the vehicle, running a TCP-server for remote access. This computer was not mounted previously [2] and the software to communicate with it's TCP-server was a major part of this project.

3.9 The Anchors

The anchors are RCMs that passively respond to ranging requests from the vehicles. The Gulliver vehicles know where the anchors are located and at least three anchors are required to determine the position of the vehicles. An illustration of this can be seen in Figure 3.3.

4

The Motor and its Controller

The motor controller is one of the central parts of the Gulliver vehicles and will be addressed in this chapter. The motor of the vehicle is a permanent magnet Brushless Direct Current (BLDC) motor with sensors to detect its rotor position.

4.1 Experiments with Sensorless Commutation

The first two vehicles from the previous project [2] were equipped with sensored BLDC motors, however, the construction of more vehicles was planned. Therefore it would be beneficial to make the motor controller work with sensorless motors (see section 2.2.1).

Another revision of the motor controller Printed Circuit Board (PCB) was made with the addition of Back Electro-Motive Force (BEMF) measurement with a resistor-capacitor voltage divider, shown in figure 4.1, as proposed by an application note from Atmel [17]. The outputs from those three combined voltage dividers/filters were connected to Analog-to-Digital Converter (ADC) inputs of the microcontroller on the motor controller PCB.

With the BEMF measurement in place, the Universal Asynchronous Receiver/Transmitter (UART) link was connected to MATLAB and the measured back-emf was plotted with a 3-sample mean value (figure 4.2) and a 3-sample median value (figure 4.3) filter.

The reason that the median filter performs much better is that, according to measurements with an oscilloscope, the noise is caused by the Pulse Width Modulation (PWM) switching noise. To avoid the switching noise and thus get a higher sampling rate with less computational power wasted on filtering, the ADC sampling could be synchronized

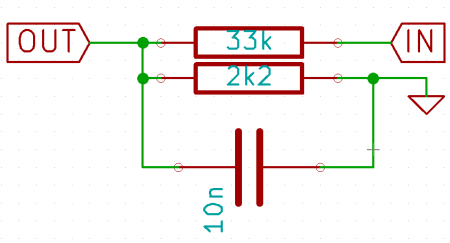


Figure 4.1: Resistor-Capacitor voltage divider/filter

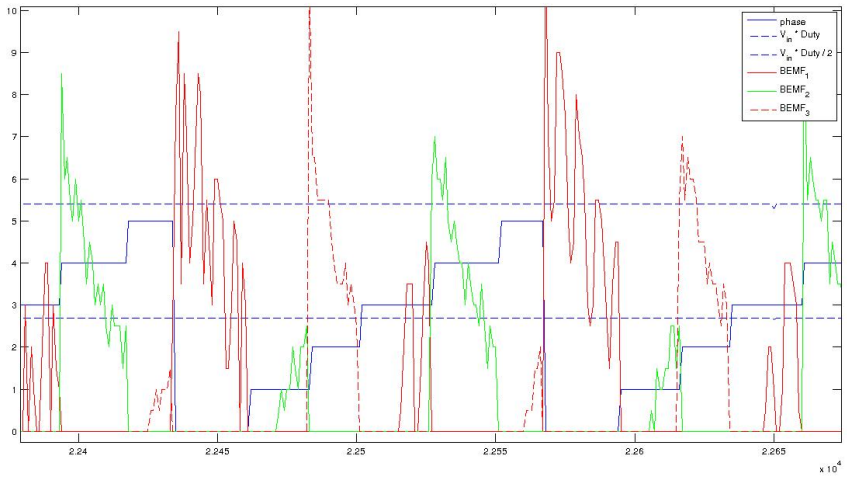


Figure 4.2: The BEMF filtered with a 3-sample mean value filter

to the timer responsible for generating the motor control PWM [18]. However, the important aspect of this experiment was to test the performance of the startup of the motor, so this was left for implementation in case it turns out that the startup works well.

The startup was implemented in the most common way, as proposed by others [8, 9], and did the job to start the motor. However, the start was not very smooth and the motor was not able to handle sufficient loads when running slowly, so the decision after this experiment was to continue with the use of sensored BLDC motors.

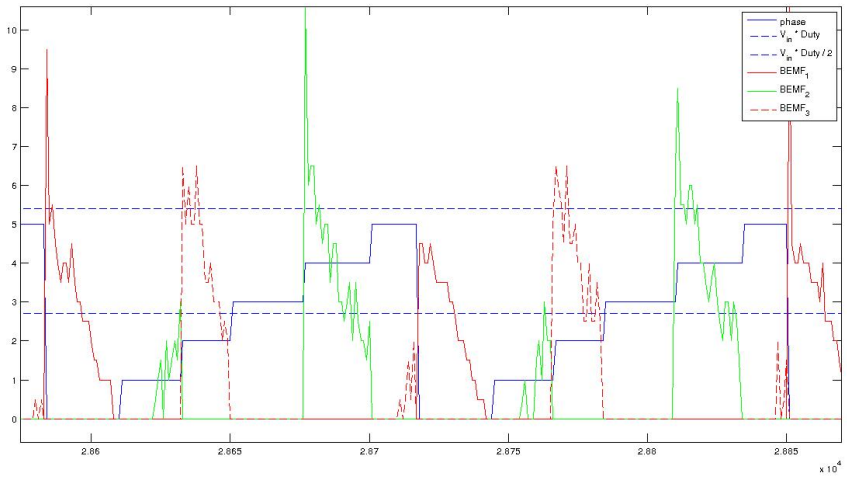


Figure 4.3: The BEMF filtered with a 3-sample median value filter. It can clearly be seen that there is less noise compared to the mean-value filter.

4.2 Speed Controller

The motor controller is not only responsible for the commutation of the motor, but also for controlling its speed. The main controller of the vehicle will tell the motor controller to run the motor at a certain speed and it is the task of the motor controller to adjust the PWM in such a way that the requested speed is maintained as good as possible, regardless of the load on the motor. For this purpose a Proportional–Integral–Derivative (PID) controller is used. The current speed of the motor is determined by measuring and filtering the time between commutations, after which it is fed to the PID controller, where the PWM duty cycle is adjusted.

There are many ways to implement PID controllers [19, 20], and which method to choose depends on the system to be controlled and the requirements of the implementation. In order to determine how difficult it is to control the motor, the step response was measured using the UART connection and plotted using MATLAB (see figure 4.4). As it turns out, the motor responds almost immediately and changes direction in less than 10 milliseconds. It was also noted that the load the vehicle causes has no major influence on the angular speed of the motor. Given this information, a very simple approach to design the PID controller, as described by [21], was used. The resulting speed control worked really well and was used for the motor controller.

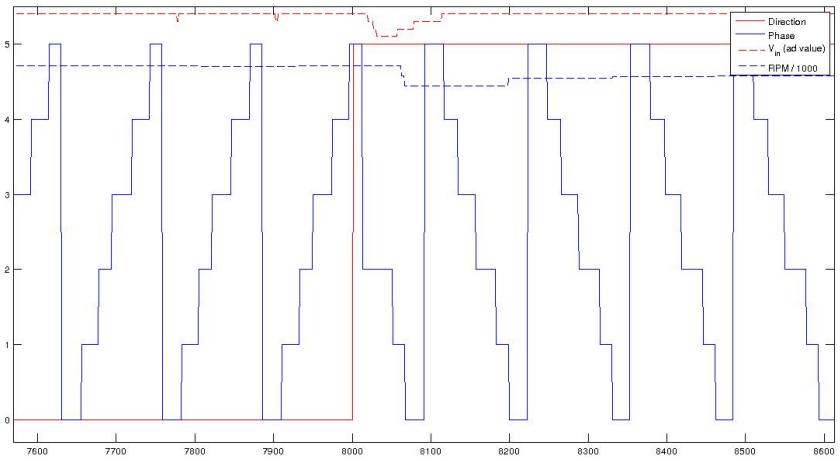


Figure 4.4: The commutations and angular speed of the motor when changing direction

4.3 Communications to the Main Controller

Communication with the motor controller is done by using a UART line. The initial program, written for the tests in [2], sent raw data over the line with no measures to check for errors. Also, the communication blocked the main controller, wasting clock cycles. Because of this a new way to communicate was introduced, which is based on packets and a separate packet handling layer.

The data is divided into packets with start and stop bytes, a byte specifying the length of the packet, an identifier byte, the parameters for the type of packet determined by the identifier and a 2-byte Cyclic Redundancy Check (CRC) checksum [22]. What this looks like can be seen in table 4.1. The same type of packet is used when data is sent from the main controller to the motor controller and also when data is sent back from the motor controller to the main controller. This was inspired by the way the P400 Ranging and Communications Modules (RCMs) handle their communication [5].

The receiver is implemented in such a way that each time a byte is received on the UART line, the state of a state machine is updated, and when the entire packet is received and the checksum is correct the packet handler is executed with a pointer to the packet data and its length. Therefore no time is wasted on waiting for data, thus utilizing the available computational power in a useful way. What also should be mentioned is that the state machine has a timeout and resets to its initial state in case no new data arrives for a longer time than expected. This is done in order to prevent the communication link from freezing if the state machine gets out of synchronization.

Bytes	Value	Description
1	0x02	Start byte
1	N	The length of the packet
1	X	The identifier. Used to determine how to interpret the payload.
N - 1	X	The payload of the packet
2	X	A CRC checksum.
1	0x03	Stop byte

Table 4.1: Packet format for motor controller

4.4 Safety Considerations

A lot of power can be delivered by the motor (about 1.6 kW) and it is therefore important that nothing goes wrong easily. The following list names a number of methods that have been implemented in order to minimize the probability for accidents:

- Each packet sent over the UART communication link has a CRC checksum and will only be used if the checksum corresponds to the payload. If data is altered due to noise, the probability that the checksum passes is very low.
- If no new commands that specify how fast to drive the motor have been received for more than 0.5 seconds, the motor will stop. So, if the vehicle is driving and someone disconnects the UART cable to the motor controller, the motor will stop after 0.5 seconds.
- The power output to the motor cannot be changed too rapidly. For instance, if the motor is running at full speed in one direction and the direction is changed to full speed in the reverse direction, the power will not be changed in one step; it will be ramped down to zero and then ramped to the desired power in the other direction. This is to avoid braking mechanical and electronic parts.
- There is a programmable limit to the maximum speed of the motor; if a higher speed is requested, it will be truncated to the highest allowed speed.

Based on experiments with the vehicles, this covers all trivial error sources encountered. There are still other things that can go wrong, such as requesting that the vehicle drives into a wall, but they cannot easily be avoided from the perspective of the motor controller. How other things than can go wrong are handled on the Gulliver vehicles can be seen in chapter 5, section 5.2.2.

5

The Main Controller and Related Software

The main controller is where the algorithms for localization and navigation are carried out. Further, the communication between all modules is initiated from here. This chapter will describe how the localization works, how algorithms for driving the vehicle are carried out and how the communication between the vehicle and the Qt program to control the vehicle works. An overview about the connection between the different pieces of software can be seen in figure 5.1.

5.1 The Localization Algorithm

The localization is achieved by combining different sources of information to make an estimate of the current position of the vehicle. This is a form of sensor fusion, which is a common practice when dealing with mobile robots [23, 24]. The algorithm does not require all sensors to be connected at the same time and will handle removal and addition of sensors dynamically with the current design.

The position of the vehicle is defined as its x- and y-position and angle θ on a two-dimensional coordinate system. The center of the vehicle is defined as the point between the rear wheels. An illustration of this is shown in figure 5.2.

Part of the program running on the main controller is responsible for estimating the current position of the vehicle, and has a set of methods to update the current position based on measurements from different sensors. These methods can be called in any order and not all of them have to be used, which is why the program can handle different

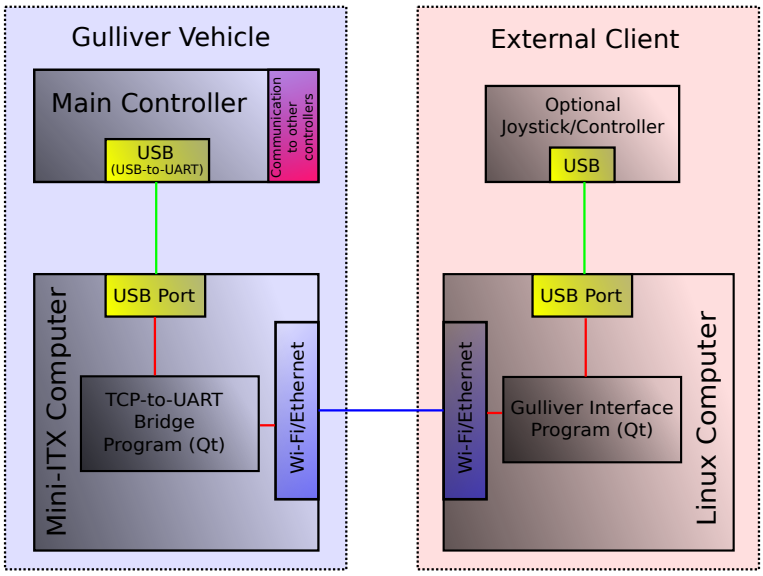


Figure 5.1: Overview of the external interface to the Gulliver vehicles

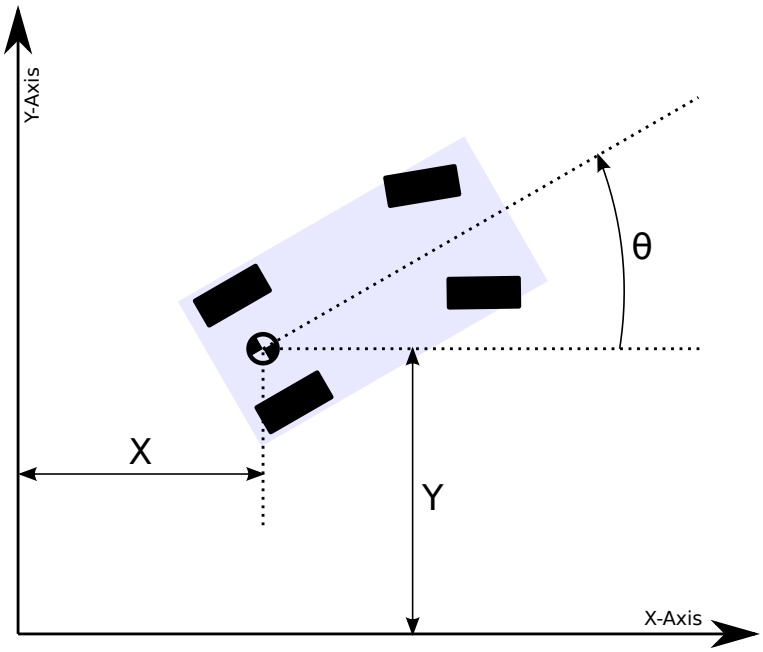


Figure 5.2: Definition of the position for the Gulliver vehicles

amounts of sensors dynamically. It is possible to extend this with more sensors if a more accurate position estimate is required. The following parts will describe how some of the

position update methods are implemented.

5.1.1 Position Updates Based on Tachometer and Steering Servo

The motor controller will continuously count the pulses the motor generates while moving and this value is read and reset from the main controller. This value is sent to the localization part of the program together with the last angle for the steering servo to update the position and angle of the vehicle. This assumes that the steering servo has been in a constant position during that movement and that the tires do not slip too much. This method to update the position has very fast update rate and low noise, however, the estimation of the position will drift over time. In the C programming language, this update looks like the following:

```
1  void dr_update_position_angle(double *pos_x, double *pos_y, double *angle) {
2      // Read the current steering angle
3      const double steering_angle = dr_get_steering_angle();
4      // Read the travelled distance from motor controller since last update
5      const double travel_distance = dr_get_travel_distance();
6
7      if (travel_distance == 0) {
8          // No movement since last update... abort
9          return;
10     }
11
12     if (fabs(steering_angle) < 0.001) {
13         // Avoid division by zero.. approximate small angles as going straight
14         *pos_x += cos(*angle) * travel_distance;
15         *pos_y += sin(*angle) * travel_distance;
16     } else {
17         const double turn_rad_rear = DR_AXIS_DISTANCE / tan(steering_angle);
18         double turn_rad_front = sqrt(
19                 DR_AXIS_DISTANCE * DR_AXIS_DISTANCE
20                         + turn_rad_rear * turn_rad_rear);
21
22         if (turn_rad_rear < 0) {
23             turn_rad_front = -turn_rad_front;
24         }
25         const double angle_diff = (travel_distance * 2.0) / (turn_rad_rear + turn_rad_front);
26
27         *pos_x += turn_rad_rear * (sin(*angle + angle_diff) - sin(*angle));
28         *pos_y += turn_rad_rear * (cos(*angle - angle_diff) - cos(*angle));
29         *angle += angle_diff;
30
31         // Make sure that the angle is within range (0 - 2PI)
32         while (*angle > 2.0 * M_PI) {
33             *angle -= 2.0 * M_PI;
34         }
35         while (*angle < 0) {
36             *angle += 2.0 * M_PI;
37         }
38     }
39 }
```

While this concept was developed and tested during the work of this project, the Localization Project [3] improved this function to also estimate a confidence interval for the

current position based on measurements made with the vehicle [3]. This was done by driving the vehicle on the floor and measuring how much this position estimate deviates from the actual position over distance.

Finally, it should be noted that this method to update the position of the vehicle will not work alone for a long time as there is no absolute reference, thus the deviation will just grow over time and has no upper bound.

5.1.2 Position Updates Based on the Ranging and Communications Module

Every time the Ranging and Communications Module (RCM) ranges with one of the anchors the measured distance, the location of that anchor and the measured standard deviation is sent to one specific position update method. This information is used to correct the x- and y-position of the vehicle. A Kalman-filter [6] is then used to update the estimated position of the vehicle and the current standard deviation. The algorithm for this was part of the localization project [3], so the details will not be covered here. It should be noted that the angle of the vehicle (see angle θ in figure 5.2) is not corrected from the RCMs at the time of this writing, so additional means are required to correct the angle of the vehicle.

5.1.3 Position Updates Based on the 9-Degrees of Freedom Sensor

The 9-Degrees of Freedom (DOF) sensor contains one three-axis accelerometer, one three-axis gyroscope and one three-axis magnetometer; which are the 9-DOF. In this application, only two out of three axes of the magnetometer are used to detect the magnetic field of the earth, effectively acting as a compass. This information is used to correct the angle of the Gulliver vehicles and is really important as the dead reckoning (see Glossary) depends on a correct angle to work properly.

5.2 How Navigation is achieved

The navigation, or the "driving algorithm", assumes that the current position of the vehicle is always known and uses a map, defined as a set of routes (see figure 5.3), to drive the vehicle along one of the routes. The algorithm is called approximately 100 times per second and in each iteration the motor controller is updated with a new speed and steering setpoint. The speed is calculated by weighting the desired speed of the previous and next point on the current route with the relative distance to the points. The calculation looks as the following:

$$k_{prev} = \frac{d_{next}}{d_{prev} + d_{next}} \tag{5.1}$$

$$v = v_{prev} * k_{prev} + v_{next} * (1 - k_{prev}) \tag{5.2}$$

where d_{next} is the distance to the next point on the route, d_{prev} is the distance to the previous point and v is the speed sent to the motor controller based on the set speed for the previous and next point (v_{prev} and v_{next}).

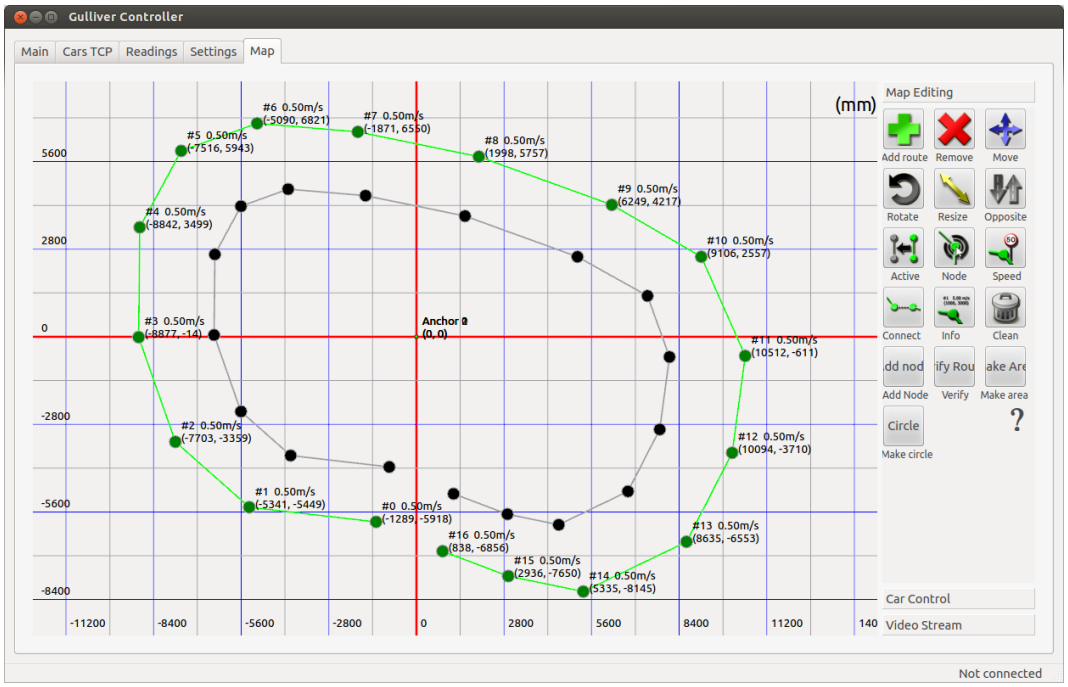


Figure 5.3: Map example in map editor with two routes (or lanes)

The steering angle is calculated such that the vehicle will follow an arc that goes through the next point on the current route. The steering angle also has a certain gain based on the distance to the next point to avoid driving a long detour when the next point is far away.

It should be noted that this navigation algorithm is very simple and that there are many considerably more advanced path planning algorithms [12, 25], but it works well for this case as the map is relatively well defined.

5.2.1 The Lane Changing Algorithm

Lane changing is the act of starting to follow another route from some point on the current route. Lane changing can be done in two ways:

1. One point on the current route together with one point on another route can be defined to change lanes. This means when the vehicle arrives at that point on the current route it will start driving towards the defined point on the other route and then continue on that route.
2. An automatic lane change can be done from any position by finding the closest point in another route and picking the second point after that one. The reason that two points are skipped is to avoid too sharp turns. The fact that exactly two points is a good number has been determined experimentally.

The most common and easiest way to change lanes is the automatic one, but the other one has been created to give more control over the lane change in case that is required.

5.2.2 Adaptive Cruise Control

Adaptive cruise control refers to the ability to adjust the speed of one vehicle to match the speed of the vehicle in front of it. This function is always active on the Gulliver vehicles and works in the following manner: The on-board sensors measure the distance to the object in front of the vehicle and if they are closer than a configurable distance, the speed is adjusted proportional to the distance to that object. This way, faster vehicles will slow down to match the speed of what is in front of them. There is also a configurable lower limit for the distance to the object in front of the vehicles, below which the motor will stop; thus this also acts as collision avoidance.

5.3 Local and External Communications

This section will cover how the communication between the nodes on the Gulliver vehicle is handled and also how the communication between the vehicle and the external Qt program is handled.

5.3.1 Local Communications

An overview about the communication links on the vehicles can be seen in figure 3.2. Most of the communications are initiated by the main controller and the other nodes respond to that. Every communication link has a timeout to make sure that the system does not freeze if that link fails. Also, the system has been designed in such a way that it does not require all communication links to function, as long as the important ones

are intact. For example, if the compass is disconnected the localization will still continue with the other sources of information, but the performance will not be as good. The individual communication links can be described as follows:

Motor Controller Communications

The communication between the main controller and the motor controller is carried out on a dedicated bus as it is important. The main controller will send packets to the motor controller (see chapter 4) with commands and for some of them the motor controller will respond. For example, the main controller may send "read and reset tachometer" and when the response is received the last tachometer value will be provided to the localization part of the program. If the communication link fails (for instance, if the cable is disconnected) the motor controller will stop the motor if no commands are received for more than 0.5 seconds, for safety.

RCM Communications

The communication to RCM is handled in a similar way to that of the motor controller. Range request packets are sent to the RCM from the timeslot manager [3] and the RCM will respond to them. When the response is received, it will be provided to the localization part of the program.

Sensors and 9-DOF Board Communications

The proximity sensors and the 9-DOF board are connected to the same bus. The main controller will address one of them at a time and ask for values and then wait for them with a very short timeout (they should respond immediately). If they respond, the response will be sent to the corresponding part of the program.

MICAz Communications

The MICAz will send packets to the MICAz interface board and they will be stored there. The interface board is also connected to the same bus as the sensors and at regular time intervals the main controller will address the interface board and read the packets received from the MICAz and sometimes send packets to the MICAz via the interface board. Note that in this case *interface board* does not refer to the external Universal Serial Bus (USB)-to-RF interface board, but the interface board between the MICAz and the multi-Universal Asynchronous Receiver/Transmitter (UART) bus.

5.3.2 External Communications

Communication to external clients can be done over USB, Bluetooth or a general-purpose Radio Frequency (RF) transceiver; however, in this project only the USB interface has been covered. The USB interface is a USB-to-UART converter and shows up as a serial port on the mini-ITX computer, where a simple Transmission Control Protocol (TCP) server application forwards data between that serial port and a TCP socket (see figure 5.1). Other clients can connect to that server using any interface that allows TCP connections, e.g. Wi-Fi.

The USB/TCP interface works with packets, very similar to the way the motor controller communication is implemented (see chapter 4.3). There are mainly two types of packets sent from the external client to the main controller: one that is sent when a response with data is expected, e.g., when asking for the current position of the vehicle; and another one that is sent when no response data is expected, e.g., when setting the speed and steering angle when the vehicle is driven manually from a joystick. What those packets look like can be seen in table 5.1.

Bytes	**Value**	**Description**
1	0x7E	Start byte
1	N	The length of the packet
1	X	The identifier. Used to determine what type of packet this is.
1	X	The sub-identifier. Used to determine how to interpret the payload of the packet.
N - 2	X	The payload of the packet
2	X	A Cyclic Redundancy Check (CRC) checksum.
1	0x7E	Stop byte

Table 5.1: Packet format for packet sent from an external client to the Gulliver Main Controller

The main controller will always respond to packets received. When no data is expected in the response an acknowledgement packet is sent back to confirm that the packet has been received and processed; when data is expected a packet with the requested data will be sent back. What the response packet looks like, sent from the main controller to the external client, can be seen in table 5.2.

Bytes	Value	Description
1	0x7E	Start byte
1	N	The length of the packet
1	X	The identifier. Used to determine what type of packet this is and how to interpret the payload of the packet.
N - 1	X	The payload of the packet
2	X	A CRC checksum.
1	0x7E	Stop byte

Table 5.2: Packet format for packet sent from the Gulliver Main Controller to an external client

5.4 The Gulliver Client Program

A program has been written, using the Qt toolkit, to provide a Graphical User Interface (GUI) for the Gulliver vehicles. It can be used to connect to any number of vehicles using the TCP interface described in section 5.3.2. From the beginning, the design has been made in such a way that communicating with the vehicles never blocks the GUI. What this interface looks like when two vehicles are connected can be seen in figure 5.4.

By using the *Cars TCP* tab, vehicles can be added dynamically and each vehicle will be plotted on the *Map* tab. Each vehicle will have its own communication and control object where data can be read from and written to the vehicle. In order to not block the GUI while communication is performed, threads and Qt's signals and slots [14] have been used. The communication procedure, using the high-level non-blocking methods, can be described as follows:

1. Call the method for the desired operation, for instance the one that reads the position of the vehicle. The returned result will be true or false depending on the status of the TCP socket, but no data from the connected vehicle is returned. Another thread will start the blocking communication and fetch the data.
2. When the result is received, a Qt signal will be emitted, containing the result. Every class that has a slot connected to that signal can process the data once it arrives. If the method called in step 1 only sent a command (not requesting data), another signal will be emitted, telling the listener that an acknowledgement (ACK) was received.
3. In case of a time out, another signal will be emitted to the listeners, telling that a timeout occurred and what caused it. This can be used in order to, for instance, show a notification about the timeout to the user.

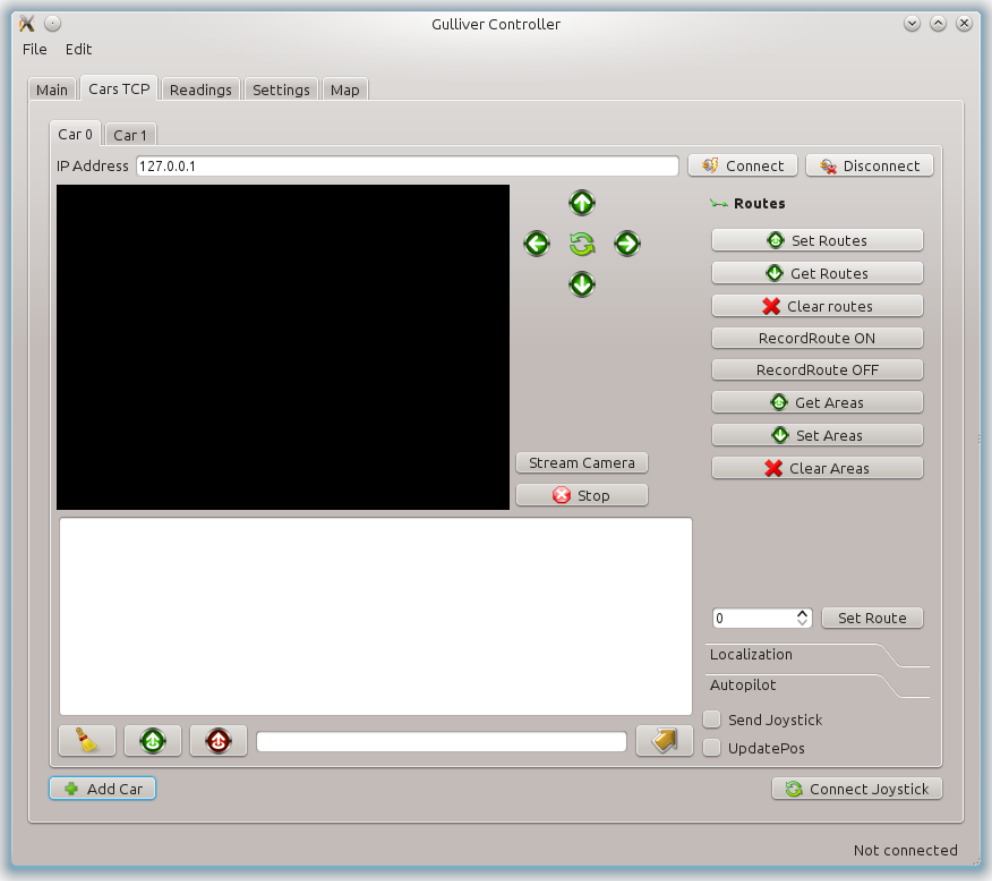


Figure 5.4: Two tabs with vehicle controls. More can be added with the **Add Car** button.

It should be noted that the map part of the program also has access to the interface objects connected to the vehicles and can do everything that can be done from the tabs (the tabs are shown in figure 5.4) with some simple modifications if the people working on the map editor decide to do so [4].

The connected Gulliver vehicles can also be maneuvered individually from a joystick from this program. Further, the interface can stream video from the vehicles in real time if a camera is mounted and a server for video streaming is running.

6

A simple Traffic Scenario Application

With everything connected, a test of the system has been done with the MICAz motes connected to two vehicles. A simple scenario with two individual connected routes and two intersections, as shown in figure 6.1, has been set up.

The MICAz motes did run a virtual traffic light with a certain schedule for each intersection, shown as red areas in figure 6.1. The design of that schedule was not a part of this project, so the details will not be discussed here.

When the vehicles entered the intersection there were three things they could do, based on the decision from the MICAz mote: continue on the same route, change lanes and continue on the other route or stop and wait for other vehicles to pass. This did work well for this scenario and the whole scenario could also be followed from the map editor/viewer in real time.

6.1 Suggested Improvements based on this Experiment

During this experiment, a few issues with the current platform were noted that could be improved in the future, namely:

- The stop and go commands were not very smooth. When the decision from the MICAz was stop in the intersection, the vehicle would brake at full power. It would give a better impression if a smooth slowdown was implemented.

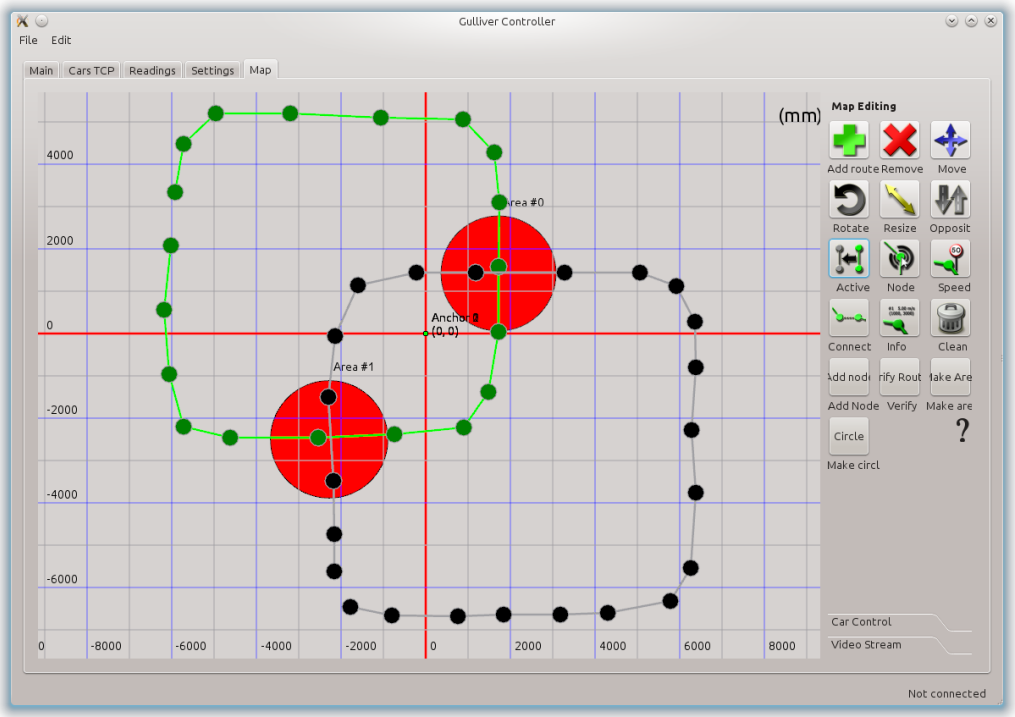


Figure 6.1: Intersection scenario with two routes and two intersection areas

- The direction of the collision sensors should be combined with the current steering angle of the vehicle. For instance, if the current steering angle is towards the left and some obstacle is getting closer on the right side, it might not be necessary to slow down.

In other aspects, this experiment went very well and can be considered as successful.

7

Conclusions

The aim of this project was to participate in the construction of a miniature vehicular platform with basic functionality to accept high level commands for navigation (see chapter 1, section 1.1 for more details). The Gulliver vehicles are now able to drive autonomously by carrying out simple high-level commands provided from another platform, such as the SUMO simulator [7].

In the end, everything in the goals and even some additional features, such as the camera stream, was implemented. It can be concluded that the Gulliver vehicles now are a bit closer to being useful in larger experiments as proposed in [1], however, there are many areas left that could and should be improved in future projects.

7.1 Suggested Future Work

For future reference, the following improvements are suggested, based on the work of this project:

- A new main controller with a more powerful microcontroller should be created. Currently, only small routes can be stored (about 400 points) and there is no hardware floating point unit, making all floating point operations considerably slower than fixed-point math.
- The algorithms for localization based on the Ranging and Communications Modules (RCMs) should be improved, such that the angle of the vehicle is considered. Currently, only the compass is used to correct the angle, and it is very sensitive to metal and other disturbances that come close to the vehicle.

- Better algorithms for navigation should be written to handle more complex traffic situations. For instance, currently there is no method to park the vehicle in a tight parking lot. A good reference for path-planning is [12], where a set of paths are generated and the one with the minimal cost based on a set of requirements is picked.
- It would be useful if every vehicle was aware of the other vehicles that are close by. Thereby, collision avoidance could be implemented based on localization in addition to the current implementation based on the on-board sensors.
- More work should be spent on integrating the Gulliver vehicles with other systems. The current on-board communication interface is specifically made for the MICAz motes and new commands are difficult to implement. Also, the client program (see section 5.4) should provide an external interface to connect to other programs, such as the SUMO simulator [7], directly, without involving the MICAz motes.

Bibliography

[1] M. Pahlavan, M. Papatriantafilou, E. M. Schiller, Gulliver: a test-bed for developing, demonstrating and prototyping vehicular systems, in: Proceedings of the 9th ACM international symposium on Mobility management and wireless access, MobiWac '11, ACM, New York, NY, USA, 2011, pp. 1–8.
URL http://doi.acm.org/10.1145/2069131.2069133

[2] P. Kaldén, E. Sternå, M. Tulldahl, M. Nilsson, N. Ilves, The gulliver car, Tech. rep., Chalmers University of Technology (2012).

[3] A. Altby, T. Boström, T. Sibgatullin, K. Stjärne, Robusta och noggranna positioneringssystem baserade på avståndsmätningar mellan mobila noder, Tech. rep., Chalmers University of Technology (2012).

[4] E. Dahlgren, J. Grundén, D. Gunnarson, N. Holtryd, A. Khazal, V. Swantesson, En plattform för testning, utveckling och demonstration med hjälp av miniatyrfordon, Tech. rep., Chalmers University of Technology (2012).

[5] Time Domain, Application Programming Interface (API) Specification, PulsON 400 RCM (2011).

[6] G. Welch, G. Bishop, An introduction to the kalman filter, Tech. rep., University of North Carolina at Chapel Hill (1997).

[7] M. Behrisch, L. Bieker, J. Erdmann, D. Krajzewicz, Sumo - simulation of urban mobility: An overview, in: SIMUL 2011, The Third International Conference on Advances in System Simulation, Barcelona, Spain, 2011, pp. 63–68.

[8] J. Shao, D. Nolan, M. Teissier, D. Swanson, A novel microcontroller-based sensorless brushless dc (bldc) motor drive for automotive fuel pumps, in: IEEE transactions on industry applications, Vol. 39, 2003.

[9] J. Shao, D. Nolan, T. Hopkins, A novel direct back emf detection for sensorless brushless dc (bldc) motor drives, in: STMicroelectronics Power Systems Applications Lab, 2002.

[10] T.-H. Kim, M. Ehsani, Sensorless control of the bldc motors from near-zero to high speeds, in: IEEE transactions on power electronics, Vol. 19, 2004.

[11] I. Cox, Blanche-an experiment in guidance and navigation of an autonomous robot vehicle, Robotics and Automation, IEEE Transactions on 7 (2) (1991) 193 –204.

[12] Z. Shiller, Y.-R. Gwo, Dynamic motion planning of autonomous vehicles, in: IEEE transactions on robotics and automation, Vol. 7, 1991.

[13] Q. Ni, W. Sun, X. Liang, Developing solaris gui application with gtk+, in: Information Science and Engineering (ICISE), 2009 1st International Conference on, 2009, pp. 3283 –3286.

[14] J. Blanchette, M. Summerfield, C++ gui programming with qt 4, 2nd Edition, Prentice Hall Press, Upper Saddle River, NJ, USA, 2008.

[15] J. Smart, K. Hock, S. Csomor, Cross-Platform GUI Programming with wxWidgets (Bruce Perens Open Source), Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[16] Time Domain, Ultra Wideband Ranging and Communications Module, pulsON 400 RCM (2011).

[17] Atmel Corporation, Sensorless control of 3-phase brushless DC motors, application note AVR444 (2005).

[18] Microchip Technology Inc., dsPIC33F/PIC24H Family Reference Manual, Section 14. Motor Control PWM (2010).

[19] R.-M. Jan, C.-S. Tseng, R.-J. Liu, Robust pid control design for permanent magnet synchronous motor: A genetic approach, Tech. rep., Ming Hsin University of Science and Technology (2007).

[20] O. Montiel, R. Sepúlveda, P. Melin, O. Castillo, M. Ángel Porta, I. M. Meza, Performance of a simple tuned fuzzy controller and a pid controller on a dc motor, in: Proceedings of the 2007 IEEE Symposium on Foundations of Computational Intelligence, 2007.

[21] T. Wescott, Pid without a phd, in: Embedded Systems Programming, 2000.

[22] P. Koopman, T. Chakravarty, Cyclic redundancy code (crc) polynomial selection for embedded networks, in: The International Conference on Dependable Systems and Networks, DSN-2004, 2004.

[23] M. Kam, X. Zhu, P. Kalata, Sensor fusion for mobile robot navigation, in: proceedings of the IEEE, Vol. 85, 1997.

[24] H. P. Moravec, Sensorfusion in certainty grids for mobile robot, in: AI Magazine, Vol. 9, 1988.

[25] Y. Kanayama, B. I. Hartman, Smooth local path planning for autonomous vehicles, Tech. rep., University of California, Department of Computer Science (1989).

List of Figures

List of Tables

Appendix A: The Gulliver Car

a part of the Gulliver Project

Students

Peter Kaldén	870705-5136	kalenp@student.chalmers.se
Erik Sternå	880924-6930	sterna@student.chalmers.se
Nigul Ilves	900908-5637	nigul.ilves@gmail.com
Mikael Tulldahl	901007-5977	mikael.tulldahl@live.se
Michael Nilsson	870918-5030	nilmic@student.chalmers.se
(*Benjamin Vedder*	*871026-7090*	*vedder@student.chalmers.se*)

Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden 2011

6 juni 2012

Summary

This report details the design and construction of a small scale test platform for simulation of traffic senarios. The platform is designed with a chassis from a remote controlled car as a base to approximate the behavior of a real car. On top of this base a system for controlling the platform has been constructed with purpose designed circuitboards. These circuitboards interface with the car's various subsystems to for example enable it track to it's own position and sense the distance to nearby objects. The car can also easily communicate with various host systems, such as a computer.

Sammanfattning

Den här rapporten behandlar designen och konstruktionen av en testplattform i liten skala för simulering av trafiksituationer. Plattformen är designad med en radiostyrd bil som bas for att uppvisa beteende som liknar en riktig bil. På denna bas har ett specialdesignat kontrollsystem byggts som kontrollerar bilens undersystem. Dessa undersystem möjliggör bilen att t.ex. spåra sin egen position i förhållande till omgivning och känna av avståndet till närliggande objekt.

Innehåll

Tack

Flera person har hjälpt oss under projektets gång och vi har varit glada för att fått arbeta tillsammans med så många kompetenta personer. Vi vill, utöver att tacka dessa, även speciellt tacka följande personer.

- Benjamin Vedder
- Elad Schiller
- Mitra Pahlavan
- Roger Johansson
- Henk Wymeers
- Gabriel Garcia
- Amir Tohidi
- Mohamed Mustafa

1 Inledning

Dagens bilar rör sig allt mer mot automatisering, med system som tex automatiskt bromsar eller väger undan om du håller på att krocka, system som upptäcker djur på vägen och system så ser håller dig mitt i filen och märker när då håller på att somna. Någon gång inom en inte allt för snar framtid så kommer det finnas helautomatiserade bilar. För att bana väg för det behöver det finnas effektiva sätt att kontrollera trafikflöden. Det går förstås att simulera i en dator, men då det finns så många faktorer så är det svårt att helt förstå alla situationer utan att verifiera simulationerna i verkligheten. Dock så är det dyrt och omständligt att verifiera i full skala och det är här Gulliver är tänkt att komma in, som en testplattform som enkelt kan använda för att verifiera simulationer utan att behöva göra en fullskalig verifiering.

2 Designmål

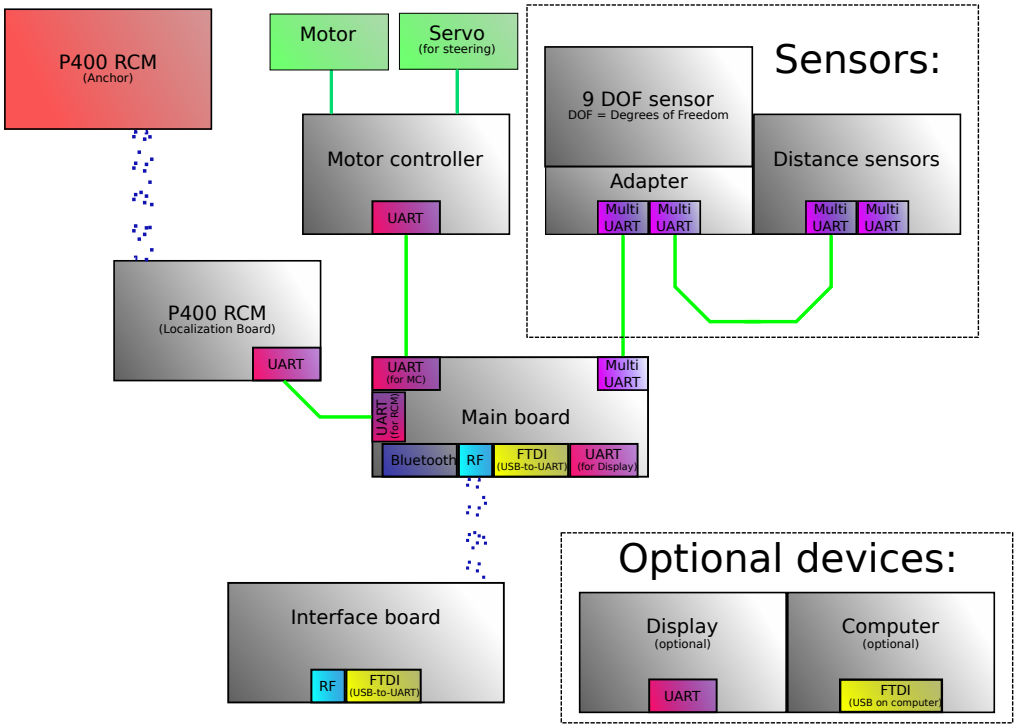
Då det är tänkt att Gulliver skall bete sig liknande en vanlig bil behövs en mekanisk struktur som möjliggör ett liknande beteende. Samtidigt krävs ett elektriskt system som kan kontrollera de mekaniska delarna på ett sätt som ger det beteende som eftersöks. Vidare krävs också återkoppling i form av sensorer så att bilen själv kan kontrolleras mer precist och ha en referens till de instruktioner som den får.

2.1 Mekaniska konsiderationer

Då en bils möjliga beteende är relativt begränsat så krävs det inte mycket mer än en motor för att åka framåt och bakåt samt en motor för att svänga. Dock är är det en fördel med dämpning och fjädring då dessa kan göra konstruktionen mer stabil och öka dess livslängd. Den mekaniska konstruktionen är dock inte avsedd att simulera de fysiska delarna av en bil utan snarare mer konceptet.

2.2 Elektriska konsiderationer

Det elektriska kontrollsystemet kräver mer ingående konsiderationer, då det måste kunna känna av omgivningen och dess egna absoluta position. Dessutom måste det även kunna kontrollera det mekaniska delarna att kunna följa de instruktioner som bilen får. *Figur 2.1* visar den föreslagna topologin för det elektriska systemet.

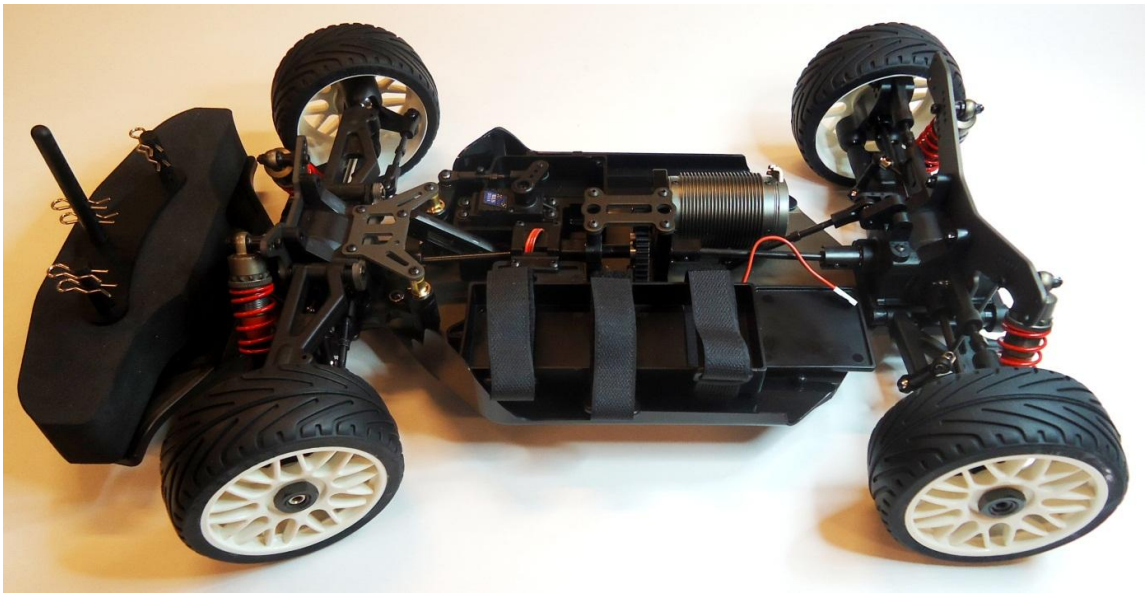


Figur 2.1: *Topologin*

De gröna linjerna symboliserar en fysisk kontakt mellan två moduler, de prickade blåa linjerna representerar trådlös kommunikation och lådorna på modulerna visar vilket protokoll som används.

3 Mekanisk Konstruktion

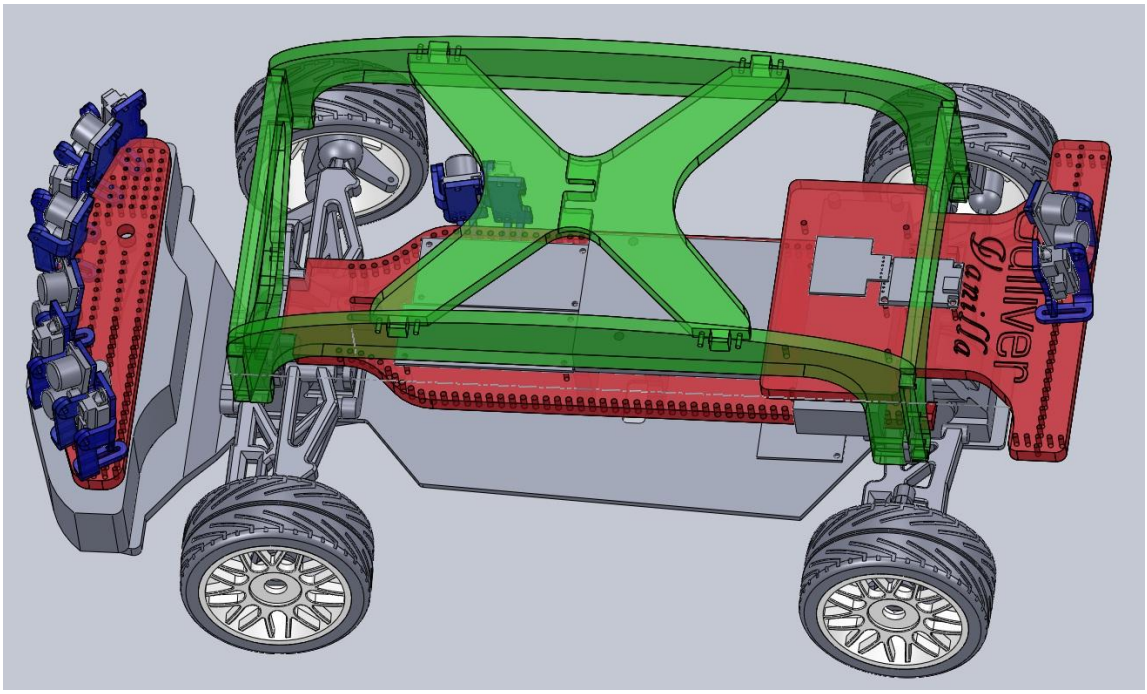
Att konstruera ett eget chassi ansågs vara allt för tidskrävande så därför valdes ett chassi till en radiostyrd bil ut till som chassi. Detta gav en bas som uppfyllde många av plattformens mekaniska krav. Chassit som valdes var av typ OFNA Dirt Electric. Det hade stötdämpare och fjädring på samtliga hjul, anpassad infästning för en elektrisk motor och ett styrservo och anlände 80 procent förkonstruerat. Figur 3.1 visar chassit när det är fullt hopsatt och styrservo och motor är monterat.



Figur 3.1: *Bilen utan egna plastdetaljer*

3.1 Chassimodifikationer

Då chassit saknade bra yta för att montera elektroniken behövdes det byggas en sådan yta. För att denna skulle bli korrekt och för att kunna bedöma olika möjliga lösningar så gjordes en CAD-modell av chassit varefter en plattform i flera delar designades till chassit. Dessutom så designades en roll-bar för att skydda elektroniken och även fästen till sensorerna som lätt kunde riktas om och placeras på flera olika ställen på plattformen. Figur **??** visar den färdiga designen CAD-modellen, notera sensorfästena (blå på bilden), plattformen (röd på bilden) och rollbaren (grön på bilden). Dessa delar tillverkades sedan i en CNC-fräs alternativt laserskärare utifrån deras respektive CAD-modeller.



Figur 3.2: *Bilen i 3D, vinkel 2*

3.2 Motor

Motorn valdes efter kriterierna att den skulle vara en borstlös DC-motor med mycket effekt. Utöver detta var det viktigt för den motorstyrning som fanns tillgänglig att motorn hade sensorer inmonterade. Den motorn som valdes var en Team Orion Vortex MR8 vilken var kapabel att ge ut 1500 Watt och ge bilen en topphastighet på c:a 50 km/h.

3.3 Stryservo

Styrservo valdes efter kriterierna kvalitet och vridmoment då svänghastighet inte är någon större vikt. Kvaliteten ansågs viktig av den anledning att lågkvalitetsservon ofta är känsliga och går lätt sönder och att de kan dåliga egenskaper. Servot som valdes var av modell Turnigy HV-767 Digital HV-2S.

4 Sensorer

Det elektriska systemet designades efter de sensorer som valdes ut för att kunna fullfölja de krav som ställdes framförallt på positioneringen men även på möjlighet att känna omgivningen. De sensorer som valdes var följande.

4.1 Omgivningsensorer

För att upptäcka omgivande objekt hade en laserscanner varit ideal då denna kan scanna nästan 360 grader samtidigt med en modul. Denna fick dock väljas bort då den var utanför budgeten. Istället valdes en kombination av ultraljud och infraröda sensorer, detta då dessa kompletterar varandra bra och i tillräcklig mängd kan ge liknande funktion som en laserscanner skulle ge.

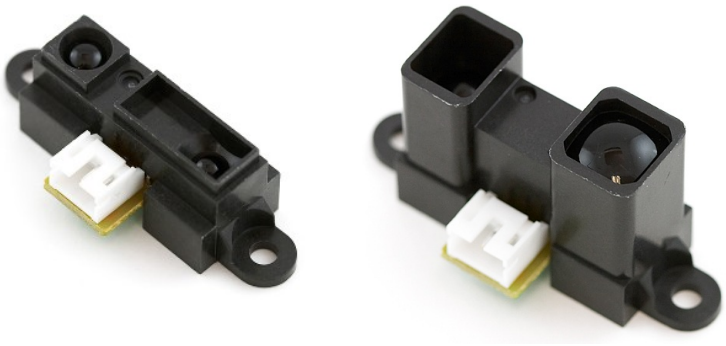
4.1.1 Val av ultraljudsensor

De ultraljudssensorer som valdes ut var av typen Ultrasonic Ranger SRF08 och de fungerar helt enkelt så att de skickar ut en ljudpuls och mäter tiden till att reflektionen kommer tillbaka till sensorn. Genom att mäta tiden tills ekot kommer tillbaka och baserat på ljudets hastighet så kan sensorn ta reda på avståndet till det objekt som reflekterade ljudet. Det finns ett par aspekter att tänka på då dessa sensorer används. Delvis kan ultraljudssensorerna regera på andra sensorers pulser ifall de sitter på ett sådant sätt och om de mäter för tätt, vilket skulle innebära att de kan rapportera felaktig mätdata. Dessutom har de en bred mätkon vilket lämpar sig mer för att upptäcka hinder snarare än att mer precist mäta avståndet till något välkänt objekt. Fördelen med ljudsensorer är även att de är färgoberoende, till skillnad från många ljusbaserad sensorer som kan ha svårt att se svarta objekt.

Figur 4.1: *Ultraljudssensor, SRF08*

4.1.2 Val av infraröd sensor

De infraröda (IR) sensorer som valdes ut är av tillverkad av SHARP och är analoga avståndssensorer som med hjälp av triangulering kan mäta avståndet till objekt. De skickar ut en infraröd ljusstråle som reflekteras av objekt och sedan mäter sensorn infallsvinkeln på ljuset och tar med hjälpa av triangulering fram avståndet till objektet. Två olika modeller av denna sensor valdes: GP2Y0D21YK, som mäter avstånd mellan 10 och 80 cm, samt GP2Y0A02YK, som mäter avstånd mellan 20 och 150 cm. Dessa sensorer är mer lämpade att mäta avstånd till större objekt då de har en mätstråle snarare än en mätkon vilket gör att de inte påverkas av andra närliggande objekt. Dessa sensorer har även en stor fördel gentemot många andra ljusbaserade sensorer som innebär att de är nästan färgoberoende (då de använder sig av triangulering).



(a) SHARP 10-80 cm (b) SHARP 20-150 cm

Figur 4.2: *Infrared distance sensors*

4.2 Odemetri

Odemetri innebär att med sensorer mäta hur någonting rör sig baserat på att veta någonting om sitt eget system och anta att det är deterministiskt (d.v.s. en styrsignal alltid ger samma resultat). Detta kallas ibland dödräkning. Med hjulens rotationshastighet kan fordonets hastighet beräknas och om hjulstorleken är känd kan också hur långt fordonet färdats beräknas. Om fordonets hjulvinkel finns tillgänglig kan baserat på fordonets axelavstånd och hjulavstånd en färdcirkelradie räknas ut vilket helt enkelt är fordonets nuvarande svängradie. Genom denna svängradie kan banan fordonets färdas i räknas ut. I en perfekt värld så vore detta tillräckligt för att spåra ett fordons position förutsatt att ursprungspositionen är känd. Dock slirar hjul, det finns mätfel o.s.v. vilket innebär att odemetri inte är tillräckligt då denna kommer att gradvis avvika allt mer från verkligheten ju längre tiden går om endast odemetri används.

4.2.1 Val av Odemetrisensorer

Motorn kommer med inbyggda sensorer vilka spårar motorns position i varvet. Dessa används av motorkontrollern för att styra motorn då denna sorts indikation är nödvändig för att driva en motor av typen borstlös DC-motor när varvtalet är lågt (och förenklar även avsevärt styrningen vid belastning). Genom att drivlinan är känd, d.v.s. hur motorrotation svarar mot hjulrotation så kan dessa sensorer också användas till odemetri. Motorkontrollern styr också servot som i sin tur styr hjulens vinkel (servon är internt reglerade positionsåterkopplade motorer). Detta innebär att när systemet är kalibrerat så kan den position som skickats ut till servot räknas som styrvinkel.

4.3 Positionering

För att odemetrin inte skall gradvis skall börja avvika från verkligheten så krävs att någonting kompenserar för denna avvikelse. Detta görs genom att uppdatera fordonets absoluta position i det koordinatsystem den befinner sig i från en annan källa. Om detta görs tillräckligt ofta jämfört med hur snabbt odemetrin avviker så hinner odemetrin inte avvika mellan uppdateringarna.

4.3.1 Val av positioneringssensor

För att ge en absolut positionsreferens används ett flertal moduler av typen PulsOn400 RCM från Time Domain. Dessa är bredbandsradiomoduler som kan mäta avståndet till varandra genom att mäta radions gångtid från modul till modul. Då minst tre av dessa moduler är utplacerade som änkareså kan fordonet med hjälp av en fjärde modul monterad på fordonet mäta avståndet till dessa ankare. Med avståndet till dessa ankare och deras positioner i ett fördefinierat koordinatsystem kan fordonet räkna fram sin egen position.

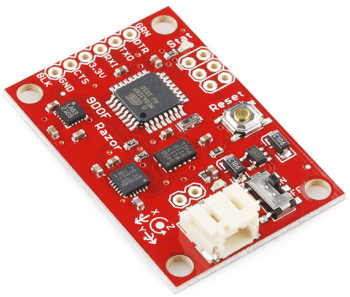
Figur 4.3: *PulsOn400 RCM*

4.4 Riktningssensor

För att snabbare kunna kompensera för gradvis avvikelse av odemetrin så krävs utöver positioneringskorrigering även riktningskorrigering eftersom ifall den uppskattade riktningen avviker från den faktiska riktningen kommer den faktiska positionen att snabbt avvika från den uppskattade. Det går att använda successiva positioner istället för en riktningssensor men då systemet är känsligare för riktningsavvikelser behövs högre uppdateringsfrekvens för riktningen.

4.4.1 Val av riktningssensor

För att mäta riktningen så valdes en "9 degrees of freedom-sensor som är ett kretskort med en 3-axlig accelerometer, 3-axligt gyro och en 3-axlig magnetsensor. För riktningsmätningsfunktionaliteten är magnetsensor den viktiga, då den känner av jordens magnetfält precis som en kompass vilket ger oss den riktning som behövs. De övriga sensorerna (accelerometern och gyrot) kan användas för att kompensera för rotationer och plötsliga accelerationer som annars kan ge tillfälliga fel.



Figur 4.4: *9-DOF-sensor*

5 Elektrisk konstruktion

Samtliga sensorer kräver stöd och kommunikationssystem och därför behövs det ett elektriskt system som ger dem de spänningar de behöver samt en kompatibel kommunikationskanal för att sända datan över. Dessutom behövs det en central enhet som kan samordna alla sensorer samt genomföra de instruktioner som krävs för att de trafikalgoritmer som skall testas. Nedan förklaras de kretskort som ingår i systemet. Deras relationer beskrivs i figur 2.1.

5.1 Motorkontroller

Motorkontrollern används för att styra motorn samt servot. Eftersom motorn är borstlös krävs det mer än att bara skicka en spänning till den så snurrar den. Motorn måste kommuteras manuellt och styras beroende på rotorns position. Motorkontrollern är en egenutvecklad motorkontroller som utvecklades utanför detta projekt. Motorontrollern skickar även ut en pulsbreddsmodulerad (PWM) signal som kontrollerar servot. För att kommunicera med motorkontrollern används en dedikerad UART-kanal.

5.2 9-DOF-sensor

9-DOF-sensorn är ett färdigt kort som köptes från sparkfun. Den innehåller en 3-axlig accelerometer, 3-axligt gyro och en 3-axlig magnetsensor samt en mikrokontroller (av typ ATmega328) som läser av sensorerna samt gör lite filtrering. För att kunna kommunicera med den via Multi-UART-bussen (MUART) tillverkades ett adapterkort.

5.3 Distanssensorkort

Distanskortet är ett egendesignat kort som hanterar avståndssensorer. Som beskrevs i sektion 4 används två typer av sensorer: ultraljud och IR-sensorer. På kortet sitter en mikrokontroller (även denna av typen ATmega328) som samlar in data från sensorerna och kommunicerar detta till huvudkortet över MUART-bussen. Detta kort är det enda som använder 5 V som logikspänning, eftersom IR-sensorerna kräver detta. Detta medför att en spänningsnivåomvandlare till MUART-bussen är nödvändig.

5.4 P400 RCM

P400 RCM (även kallad endast RCM) är en färdig modul från Time Domain. Den mäter avstånd till andra likadana moduler genom gångtidsmätning av radiovågor och används för att bilen ska kunna bestämma en absolutposition. Huvudkortet kommunicerar med denna modul med en dedikerad UART-kanal.

5.5 Interface-kort

Interface-kortet är ett adapterkort som används för att kunna kommunicera trådlöst med bilen via en generisk RF-modul. Den kopplas till datorn via en USB-kabel. På kortet sitter en krets som tolkar USB och omvandlar det till UART som mikrokontrollern (av typ ATxmega32A4U) tolkar för att sedan sända vidare till bilen via RF-modulen.

5.6 Huvudkort

Huvudkortet är den modul som samordnar alla enheter. På den sitter en mikrokontroller av typ ATxmega128A3 som tar in data från alla sensorer och skickar kommandon till utenheterna.

Den har de kommunikationskanaler som nämnts tidigare; dedikerad UART till RCM-modul och motorkontroller; MUART till distanskort och 9DOF-kort. Dessutom finns det en krets som omvandlar USB till UART därigenom direkt kommunikation med en dator är möjlig och en RF-modul som kan användas för att kommunicera med interface-kortet. Utöver detta finns plats för ett SD-kort för datalagring och en blåtandsmodul för ytterligare kommunikationskanaler. Slutligen finns även en dipswitch som användaren kan använda för att ställa in bilens adress med ifall det används flera bilar.

6 Mjukvara

För att kunna utnyttja bilens funktioner krävs en stor mängd mjukvara. Varje kort har sin egen mjukvara. I denna del behandlas endast den helt eller delvis egenproducerade mjukvaran.

6.1 Motorkontroller

Mjukvaran i motorkontrollern är utanför ramen för detta projekt, men i korthet så mäter den positionen på motorn och beroende på detta skickar signaler till motorn så att den roterar. Den innehåller även en PID-regulator så att den alltid försöker hålla samma hastighet oavsett last på motorn. Vidare kontrollerar den styrservot med PWM-styrning. Den mäter kontinuerligt motorns rotationshastighet, motorns temperatur, sin egen temperatur samt batterispänningen på både logikbatteriet och motorbatteriet. Den tar emot kommandon från huvudkortet via UART. Den tar emot kommandon såsom hastighet och riktning på motorn samt position på servot och kan dessutom rapportera tillbaka all sin data till huvudkortet.

6.2 9-DOF-sensor

9-DOF-sensorn samlar in data från accelerometern, gyrot och magnetometern. Mikrokontrollern kommunicerar med de tre sensorerna med hjälp av I2C. Den har även stöd för kalibrering som går till så att extremvärdena noteras och därefter räknas det slutliga värdet för varje sensor utifrån detta. Detta är nödvändigt då varje sensor är individuell från fabrik. Varje sensor har även inbyggd viss filtrering (lågpassfilter).

6.3 Distanssensorkort

Distanssensorkortet samlar in data från distanssensorerna. Ultraljudssensorerna kommuniceras till via I2C och för att öka samplingshastigheten utan att riskera att sensorerna stör varandra utförs mätningar på fysiskt motstående sensorer samtidigt. Ultraljudssensorerna rapporterar sitt avstånd direkt i cm. IR-sensorernas avstånd ges ut i en analog spännings och avläses med den inbyggda analog-till-digital-omvandlaren (ADC). Värdet som sensorerna ger ut omvandlas till avstånd med hjälp av en look-up-tabell för att avlasta processorn, då den division som krävs skulle innebära mycket jobb för den.

6.4 Interface-kort

Interface-kortet är ett adapterkort som används för att kunna kommunicera trådlöst med bilen via en generisk RF-modul. Den får via USB-UART-omvandlaren in kommandon från datorn seriellt via UART som den sedan vidarebefordrar via SPI till RF-modulen som sedan sänder ut detta. Den kan även ta emot data från bilen via RF-modulen och skicka tillbaka den till datorn.

6.5 Huvudkortet

Här implementerades funktioner för grundläggande kommunikation till alla enheterna samt grundläggande styrfunktioner såsom kör- och styrkommandon. Som tidigare nämnts kommunicerar den med alla de andra korten, främst via UART (antingen dedikerad eller via MUART-bussen), samt även med SD-kortet och bilens RF-modul via SPI. Mjukvaran i huvudkortet är mycket omfattande och kommer därför bara att beskrivas kort.

6.5.1 RCM-kommunikation

För att kommunicera med RCMen används en paketstruktur. Paketet innehåller en header som används för synkronisering, längd på meddelandet, kommando, meddelande-ID och slutligen en CRC (cyclic redundancy check, används för att kontrollera att meddelandet kommer fram korrekt). Allt detta byggs i förväg upp och skickas sedan till RCM-modulen. RCM-modulen svarar sedan med det man bad om, t.ex. avståndet till en annan RCM med en viss adress. Huvudprocessorn ber bilens RCM att mäta avstånd till var och en ankar-RCMerna och räknar baserat på detta ut bilens position givet ankarnas positioner i det fördefinierade koordinatsystemet.

6.5.2 Styrkommandon

Förutom de enklare "kör-, stanna- och sväng-kommando kan bilen även göra en del mer avancerade kommandon. En av de grundläggande byggstenarna i mer avancerade användarfunktioner är kommandot "kör till en punkt i koordinatsystemet". Detta kommando använder sig utav odemetrin som den väger samman med positionen från RCMen och riktningen från 9-DOFen. Baserat på sin egen position och riktning räknar den ut en cirkelbåge som den kan köra för att komma till den angivna punkten. Detta innebär att algoritmen producerar väldigt vida svängar när målet är långt borta och skarpa svängar nära målet. För att sträckan inte ska bli onödigt lång och indirekt vid mål långt borta, skalas styrvinkeln med avståndet till målet, vilket gör att den fort svänger åt rätt håll för att sedan åka ganska rakt mot målet. Genom att att sekventiellt koppla samman ett flertal "kör till en punkt-kommandon kan man få bilen att köra en bana. Eftersom det kan vara svårt att träffa en punkt exakt har bilen en del funktioner för att åtgärda detta. Den har en viss tolerans-zonrunt varje punkt som innebär att om bilen kommer innanför denna zon räknas den är framme vid punkten och kan börja gå mot nästa. Den kan även "övergepunkter som den tycker är för svåra att komma till och fortsätta direkt till nästa. Med hjälp av dessa grundstenar kan användaren bygga sin applikation för att få bilen att göra ännu mer avancerade saker och med hjälp av detta testa sina scenarion.

7 Avslutning

7.1 Resultat

Gruppen lyckades under projektet att ta fram en fungerade bil som kan navigera i ett rum och analysera sin omgivning. Bilen som togs fram är flexibel och robust med goda möjligheter till utbyggnad både vad gäller hårdvara och mjukvara. Det är enkelt att kommunicera med bilen från en dator, då det finns flera sätt att göra detta på, samt att bilen innehåller en del kommandon på lite högre nivå förutom de mest grundläggande funktionerna. Det är även enkelt att få ut information om bilens status, t.ex. hastighet, position och sensordata som kan användas för t.ex. visualisering av bilen och felsökning.

7.2 Diskussion och förbättringsmöjligheter

Generellt sett ser vi projektet som en lyckat projekt. Dock finns det en del utvecklingsmöjligheter för framtiden. Det är dock självklart upp till användaren att definiera vad denna behöver. Nedan följer exempel på saker som kan förbättras samt förslag för fortsatt utveckling:

- Mjukvaran skulle kunna vara mer strukturerad och enklare att förstå. Som det är nu fungerar den väldigt bra och är väl optimerad. Med detta följer som de flesta som jobbat med programmering förstår att den samtidigt är ganska svår att sätta sig in i för att göra några större ändringar.
- Bättre integrering och användning av avståndssensorer skulle vara bra. Detta är dock någonting som skulle bero på vilken applikation som bilen skulle användas till och skulle vara upp till användaren att designa. Som det är nu har bilen endast mycket enkla funktioner för att vara en sista utväg ifall den skulle vara på väg att krocka.
- Integration av fler av sensorerna från 9-DOFen är någonting som funderats mycket på under projektets gång. När bilen testats har den främst körts ganska långsamt. Accelerometern och gyrot är sensorer som lämpar sig bra för korrigering av plötsliga förändringar. Dock skulle integration av dessa sensorer kunna förbättra dödräkningen avsevärt.
- Mjukvara för datorn är någonting som inte är inom detta projekts ram, men någonting som skulle vara nödvändigt för fortsatt utveckling. Sådant är även nödvändigt vid koordinering av flera fordon.
- Hitta en billigare lösning för positioneringen är nödvändigt för att hålla priset nere. En RCM-modul kostar mer än resten av bilen. Detta är dock svårt att lösa till ett billigt pris med samma prestanda som RCMerna ger. Vid ett större system utomhus skulle vanlig GPS kunna användas, men för denna bil skulle detta innebära alldeles för dålig upplösning och uppdateringshastighet.
- En annan lösning för den trådlösa kommunikationen till bilen vore bra. Eftersom RF-modulerna som används inte är extremt robusta händer det att datorn ibland tappar kontakt med bilen. Bluetooth eller WLAN skulle kunna användas komplementärt till RF-modulen för att uppnå högre robusthet.

7.3 Reflektioner på gruppen

Vi har haft mycket nytta av den spridda expertis som fanns inom projektgruppen. Detta var bra på många sätt, och vi nyttjade de olika personernas förmågor väl, anser vi. Vissa av oss var mycket duktiga på elektronik, andra på CAD och några på organisation av projektet. Dock är det av högsta vikt att en teknisk grund och förståelse om elektronik, mekanik och datorteknik, något alla deltagare hade. Denna grundkunskap behövs för att alla deltagare skall kunna på ett bra sätt diskutera design och upplägg av bilen och dess system samt kunna förstå problem och vad problemen innebär vid diskussioner med personer i gruppen och utanför gruppen.