L.O.S.T Positioning

Line Of Sight Technologies Positioning of RoboRally Robots

NICLAS BASSILI

ERIK BLOMQVIST
L.O.S.T Positioning

Line Of Sight Technologies Positioning of RoboRally Robots

ERIK BLOMQVIST
NICLAS BASSILI

Bachelor’s Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2018:64
Abstract

RoboRally is a board game where the goal is to drive a small robot from a start position to a goal across a dynamic board with different types of obstacles. The robots in this game aren’t real robots, they are small plastic figures which are moved by hand during the game. Our vision is to research a solution for implementing the game with real robots. In order to do this, precise positioning is key. The purpose of this project was therefore to create a local positioning system that allows a robot to find its position in a defined space and change it on a user command with good precision.

The built positioning system is based on the time of flight technique using infrared light and ultrasound. The system uses two transmitters and one receiver to calculate two planar distances which can be converted to a coordinate within the board.

To assess the system performance in a RoboRally application a small robot was built and multiple tests conducted. The first result was that a one-dimensional distance calculation using the time of flight technique achieved a mean error of 3.2 mm for distances shorter than 1 m. The next result was that the positioning system achieved a 90 x 70 cm rectangle field of view with an accuracy between 1 and 6 cm within this area. The last result was that the robot was able to move a specified distance with a precision of $\pm 1$ mm and then verify its position successfully at a rate of 97 %.

The final conclusion of the project is that the positioning system is accurate enough for a RoboRally application but needs improvements on its field of view.

Keywords: Mechatronics, Ultrasound, Positioning System, Time of Flight, Arduino.
Referat

Positionering av RoboRally robotar


Positioneringssystemet som tagits fram är baserat på time of flight teknik med hjälp av infrarött ljus och ultraljud. Systemet använder två sändare och en mottagare för att räkna ut två avstånd som sedan kan omvandlas till en koordinat på spelplanen.

För att bedöma systemets prestanda i RoboRally sammanhang så har en liten robot byggts och flera tester utförts. Det första resultatet är att en endimensionell avståndsberäkning med time of flight tekniken kunde uppnå ett medelfel på 3,2 mm för avstånd kortare än 1 m. Nästa resultat var att positionssystemet uppnådde ett 90 x 70 cm rektangulärt synfält med en noggrannhet mellan 1 och 6 cm inom detta område. Det sista resultatet var att roboten kan förflytta sig en bestämd sträcka med en noggrannhet på ± 1 mm och sedan verifiera sin position korrekt 97 % av försöken.

Projektets slutsats är att positionssystemet har tillräckligt bra noggrannhet för att kunna användas för RoboRally men att det behöver förbättrat synfält.

Nyckelord: Mekatronik, Ultraljud, Positioneringssystem, Avståndsmätning, Arduino.
Acknowledgements

We would like to thank multiple people for their help during the project. Workshop assistant Staffan Qvarnström for his practical knowledge and supplying hardware components. Mechatronics master student, Gustav Sten, and electronics professor, Hans Bengt Johansson, for their valuable feedback when designing the transmitter drive circuit. Course assistants; Johan Ehrenfors, Martin Gylling, Stanislav Minko and Ruben Svensson, for their invaluable help with various software, mechatronics discussions and other tips and tricks on how to make it through the bachelor thesis course. Last, but not least, prototyping assistant, Stefan Ionescu, and our colleagues and friends, Joakim Hellberg and Axel Sundkvist for their help with making this thesis flawless.
## Contents

1 Introduction ................................................................. 1  
  1.1 Background ......................................................... 1  
  1.2 Purpose ............................................................ 2  
  1.3 Scope ............................................................... 2  
  1.4 Method ............................................................. 3  

2 Theory ................................................................. 5  
  2.1 Microcontroller .................................................... 5  
  2.2 Ultrasound .......................................................... 5  
    2.2.1 Ultrasound module US-015 ................................ 5  
  2.3 Infrared Light ..................................................... 7  
  2.4 Time of Flight ..................................................... 7  
  2.5 Signal Detection with Interrupts ............................... 8  
  2.6 Coordinate Calculation ......................................... 8  
  2.7 Encoders ............................................................. 10  

3 Demonstrator ......................................................... 11  
  3.1 Hardware .......................................................... 12  
    3.1.1 Robot Electronics ........................................... 12  
    3.1.2 Transmitter and Receiver circuits ....................... 13  
  3.2 Software .......................................................... 14  

4 Testing and Results .................................................. 17  
  4.1 Test 1 - Calibration .............................................. 17  
  4.2 Test 2 - FOV for ultrasound transducers ....................... 18  
  4.3 Test 3 and 4 - Angle dependency of US-015 module TX/RX components 19  
  4.4 Test 5 - Two-dimensional positioning error ................... 22  
  4.5 Test 6 - Movement and position verification .................. 23  

5 Discussion and conclusion ........................................... 25  
  5.1 Discussion ......................................................... 25  
  5.2 Conclusion ....................................................... 26  

6 Recommendations and future work .................................. 27
List of Figures

1.1 The robot platform and the two ultrasound towers with multiple transmitters. .................................................. 3

2.1 Eight pulses at 40 kHz measured on the transmitter output of the US-015 and the echo signal rising edge. .................. 6

2.2 Received ultrasound waves measured on the receiver output on the US-015 and echo signal falling edge. ...................... 6


2.4 Top view of game board and the different transmitters. Created in paint.net ...................................................... 9

2.5 Top view of game board. Created in paint.net .............................................................................................. 9

2.6 Basic example of an encoder disc with holes.[2] ............................................................................................ 10

3.1 Robot platform with stacked modules ............................................. 11

3.2 Transmitter driver Arduino shield. .................................................. 13

3.3 Top receiver module. .......................................................................................................................... 14

3.4 Flowchart of transmitter and receiver logic. Made with Gliffy. ............................................................................. 15

3.5 Flowchart of robot main logic and the function get_position(). Made with Gliffy. .................................................. 16

4.1 Comparison of calculated distances and correct values after calibration, created with Google Sheets. ............................ 18

4.2 Test 2; definite Field Of View. Created in paint.net. .................................................................................. 19

4.3 Test 3; Increase in measured distance affected by angle to receiver center line. Created in paint.net. .......................... 20

4.4 Result of Test 3; Error in increased calculated distance due to rotation, $\alpha$, of the receiver. Created with Google Sheets. .................................................................................................................. 20

4.5 Test 4; Increase in measured distance affected by angle to transmitter center line. Created in paint.net. .................... 21

4.6 Error in calculated distance due to transmitter not facing receiver, created with Google Sheets. .............................. 21

4.7 Calculated absolute positioning error, created with MATLAB. ........................................................................... 22

4.8 Test 6; Successful verification of correct goal position. .................................................................................. 23

4.9 Test 6; Successful verification of invalid goal position. .................................................................................. 23
B.1 Circuit schematics for the ultrasound receiver board, made with Eagle CAD. ........................................................................................................................................ 47
B.2 Simplified visual representation of how the robot is wired. Made in the free software Fritzing. ........................................................................................................................................ 48
B.3 Circuit schematics for the ultrasound transmitter driver, made with Eagle CAD. ........................................................................................................................................ 49
B.4 Example circuit of external IR-LEDs that can be connected to the driver shield. Made with Eagle CAD. ........................................................................................................................................ 50
List of Abbreviations

**CPU** Central Processing Unit.
**FOV** Field of View.
**FFF** Fused Filament Fabrication.
**GPIO** General Purpose Input/Output.
**IR** Infrared.
**I/O** Input/Output.
**LED** Light Emitting Diode.
**PLA** Polylactic acid.
**PCB** Printed circuit board.
**PWM** Pulse Width Modulation.
**TOF** Time of Flight.
**US** Ultrasound.
Chapter 1

Introduction

The thought of making the small robots in the board game RoboRally [4] come to life have probably crossed the minds of everyone who have played the game. This chapter introduces some of the basic principles of RoboRally, what the robots would be able to do, and some of the challenges presented trying to automate an analog board game with real robots.

1.1 Background

In the board game RoboRally the goal is to drive a small robot from a start position to a goal across a dynamic board with different type of obstacles, like conveyor belts, rotating platforms, and lasers. Normally the robots are represented by small plastic figures and the obstacles are printed graphics on the board. All movement and interaction with the obstacles are done by hand by the players. During the game the robots are “controlled” by programming cards that tell them how to move. By the nature of the game, these movements are often hard to anticipate as the robots will push/crash into each other and move with different board elements.

In order to implement the RoboRally game with real robots, precise positioning is key. For the game to work smoothly the robot should be able to autonomously move to new positions commanded by the programming cards and end up in a valid position, i.e. not half way between the tiles. Positioning can be achieved with many different solutions. Most solutions can be split into two groups defined by where the main intelligence is placed. You could either have a smart game board that knows the position of its robots and controls them according to the game or you could have smart robots that knows their own position and moves independently of the others. The latter is the approach focused on in this project.
CHAPTER 1. INTRODUCTION

1.2 Purpose

The purpose of this project is to create a local positioning system that allows a robot to find its position in a defined space and change it on a user command with good precision. Our research questions:

- What is the accuracy and maximum angle of sight for a one-dimensional distance calculation using "basic electronics" (see paragraph 1.3 Scope), a microcontroller and the time of flight concept?

- What is the field of view for the two-dimensional positioning system and what is its accuracy within the field of view?

- Is it possible to control a RoboRally robot using DC-motors and encoders while connected to the LOST positioning system constructed of "basic electronics"?
  - How accurately can it move a set distance using this setup?
  - How accurately can the robot verify its goal position using the LOST positioning system?

1.3 Scope

In this project we aimed to focus on the implementation and reliability of the positioning system. We did not intend to build the entire RoboRally game or an interactive game field with multiple robots. Our aim was to build one robot and a game field of no more than one square meter. Mainly due to short project time, resources, and limited space during the final presentation.

During the course of a real game the robots will crash into and push each other around to new positions. This was instead simulated by manually moving the robot away from its desired position. The robot would then use the positioning system to verify if it is standing in a valid position and signal the result. Movement for correcting the current position or compensation in the following moves was not developed at this point, but the system i prepared for adding the functionality. Further effects of robots colliding, such as latching onto each other or not having enough power to push the other, was not examined or solved.

The so called "basic electronics", was whatever components that was available to us in the lab at KTH or cheaper Arduino modules and other hobby grade electronics.

As the project was done during a short period of time and with limited resources, the solution is not intended to reach the performance limits of the concepts discussed in this report, but only to test the chosen technologies in combination for this application.
1.4 Method

The first step was to construct and implement the positioning system. This was done by first building one transmitter and a simple receiver consisting of only an Arduino and sensors to receive the signals. The signal processing and distance calculation was then implemented in the Arduino code.

The next step was the building of another transmitter and improvement of the code so the Arduino knows which transmitter the signal is coming from. The distance calculation was then updated to instead calculate a 2D position.

The last step was the construction of a robot platform for the receiver Arduino to control. Movement was achieved with the use of DC-motors with integrated encoders and multiple 3D printed parts. The final iteration of the transmitter towers and the robot can be seen in Figure 1.1).

Several tests were conducted in order to give a final answer to the research questions.

Figure 1.1: The robot platform and the two ultrasound towers with multiple transmitters.
Chapter 2

Theory

This chapter aims to explain the theory for the positioning system and the robot motion control.

2.1 Microcontroller

A microcontroller is a Central Processing Unit (CPU) with integrated peripherals such as memory and Input/Output (I/O) pins. Microcontrollers have a clock frequency between 30 kHz and 200 Mhz compared to CPUs in PCs that today reaches frequencies above 3 GHz. This frequency difference is due to their different purposes, a microcontroller is intended to be part of an embedded system, where energy resources are limited. Energy losses from the computations are reduced by using a slower processor, thus explaining the frequency difference [8].

This project used two Arduino Uno microcontrollers, one for transmitting ultrasonic and infrared signals and one for receiving the signals, calculating its position and robot control.

2.2 Ultrasound

Ultrasound (US) refers to sound waves with a frequency higher than what the human ear can hear, typically ranging from 20 kHz to multiple GHz depending on application. Sound waves in air, travel with a relatively constant and slow velocity [12]. This makes ultrasound fitting for use with the Time of Flight concept (explained below) and is widely used for measuring distances [7].

2.2.1 Ultrasound module US-015

The US-015 is a small module that can send and receive ultrasound signals. It is normally used for measuring distances by sending out a US-wave and measuring the time before the echo gets back to the receiver.
In normal operation the module is first triggered by a 11 µs pulse and after charging up a higher voltage it sends out eight square pulses at 40 kHz, as seen in Figure 2.1. It then brings the echo pin high and starts listening for sounds. When the received sound pressure gets above a certain threshold, the echo pin is driven low again (seen in Figure 2.2) and the falling edge of the signal can be detected by a microcontroller.

By removing one of the transducers it is possible to use the module for only receiving or transmitting signals. "As the module normally expects to receive echos from itself, it must be remotely triggered before it starts to listen for any incoming sounds and can function as a pure receiver."
2.3 Infrared Light

Infrared (IR) radiation transmitted by a Light Emitting Diode (LED), is referred to as infrared light and typically has a wavelength around 900 nm. IR-LEDs combined with IR sensors and detectors are widely used for wireless communication between electrical devices such as TVs and their remote controllers. An IR sensor picks up the level of incoming IR light and outputs a corresponding analog signal, while an IR detector reacts only to modulated IR signals from transmitters where the light is pulsed with a high frequency. The IR detector outputs a digital high signal, 1, when it detects IR light with the right pulse frequency [13][14].

![Modulated IR light](image)

Figure 2.3: Modulated IR light [11]

2.4 Time of Flight

Time of Flight (TOF) is a simple distance measuring technique that is based on knowing the propagation velocity of a signal. The method calculates a distance by first measuring the time between when a signal was sent and when it was received, this is the signal’s time of flight [1]. The distance can then be calculated with the simple time-speed-distance equation as

\[
\text{distance} = \text{propagation velocity} \cdot \text{time of flight.} \quad (2.1)
\]

In the LOST positioning system, the TOF is calculated by simultaneously sending an infrared and ultrasound signal, the speed of the infrared signal is many times higher than the sound and will reach the robot receiver almost instantly, this triggers the robot to save the current time and start listening for ultrasound. When the ultrasound signal reaches the receiver it will save the current time and calculate the TOF as the time difference between the two saved values. The infrared signal works as a wireless time synchronization between transmitter and receiver. Because the extremely high propagation velocity of light signals compared to sound signals, errors due to infrared time of flight are in this application approximated to zero.
2.5 Signal Detection with Interrupts

The time of flight measurement can be improved by using an interrupt. An interrupt is a call to the processor to get its attention. The processor will then do whatever the interrupt subroutine asks for until finished. In a real time system this could however cause problems if not used properly. If the interrupt call takes too long then other functions of the system such as feedback control might fail due to the long pause. The interrupt is therefore ideally a short command that can be executed quickly, so that the processor can return to wherever it left the loop [9]. Interrupts are particularly useful for measuring signals with a constant sampling frequency, calculating time between different events and pulsing a signal with a specific frequency [5].

The idea is that the infrared receiver will trigger an interrupt, saving the time and triggering the US module to listen for the incoming US signal. When the US signal reaches the robot it will trigger another interrupt that saves the time and returns the time difference.

2.6 Coordinate Calculation

The distance calculation, using the time of flight concept, can be expanded into a coordinate calculation. This is done by calculating the distances between the receiver and two transmitters and then using geometric relations. Two distances are sufficient for calculating the position of the robot. If only one distance $r$ is known then the robot can be anywhere on a circle with radius $r$ from the transmitter. The position of the robot is found at one of the intersections of two such circles, with centres at two the US-transmitters in opposite corners and the IR-transmitter in between as shown in Figure 2.4. As the transmitters are placed at known locations in relation to the game board and that the entire board is on one side of a line connecting the two transmitters, it is always known which one of the points, where the circles intersect, that is the correct one.
2.6. COORDINATE CALCULATION

Figure 2.4: Top view of game board and the different transmitters. Created in paint.net

Both circles possible coordinates \((x, y)\) are described by the circle equation \((x - a)^2 + (y - b)^2 = r^2\), where \((a, b)\) is the circle centre and \(r\) its radius. With the circle centers in the origin, \((0, 0)\), and in the down right corner, \((L_x, 0)\), (see Figure 2.5) the equations are

\[
\begin{align*}
  x^2 + y^2 &= r_1^2, \\ (x - L_x)^2 + y^2 &= r_2^2.
\end{align*}
\]

Figure 2.5: Top view of game board. Created in paint.net
CHAPTER 2. THEORY

The robot position is the coordinate \((X_r, Y_r)\) that solves both of these equations within the board. Subtracting equation 2.3 with equation 2.2 and solving for \(X_r\) yields

\[ X_r = \frac{r_1^2 + L_x^2 - r_2^2}{2L_x}. \]  

(2.4)

\(Y_r\) can then be calculated by inserting the expression for \(X_r\) in equation 2.2 and solving for \(Y_r\), this yields

\[ Y_r = +\sqrt{r_1^2 - X_r^2}. \]  

(2.5)

2.7 Encoders

When controlling DC-motors in various mechatronics applications, it is common to use rotary encoders for detecting the direction, speed, and acceleration of the motor shaft. With this information it is possible to get closed loop feedback control of the motors [10].

Rotary encoders often use a disc with evenly spaced markings that are picked up by a sensor each time they pass by. The sensor then produces an electronic pulse that can be picked up and analyzed by a microcontroller. The markings can for example be holes in the disc that makes light shine through and hit a light sensor, (example in Figure 2.6), that then produce a higher voltage or let current flow through it [6].

![Figure 2.6: Basic example of an encoder disc with holes.][2]

---

[2]: Link to image or figure.
Chapter 3

Demonstrator

A new custom robot platform, see Figure 3.1, was developed and built to test the positioning system with real robot movements and to get valuable insight to the plausibility of building several robots that could be in an actual RoboRally game.

Figure 3.1: Robot platform with stacked modules
3.1 Hardware

The spine of the robot consists of modular 3D-printed platforms that stack on top of each other. This design makes it easier to add or remove modules to the stack and have easy access to all components during the development process.

The robot is driven by two Faulhaber 1331T012SR19 motors with integrated IE2-400 encoders and planetary gearhead. The gearheads are model 14/1 by Faulhaber with a reduction of 14:1. The motors are attached directly with 3D-printed adapters to two diagonally opposite wheels. This setup allows the robot to turn around its own center axis by alternating the directions of the motors.

The 3D-printed parts where mainly printed on Ultimaker FFF [3] type printers in PLA and Ninjaflex plastic.

3.1.1 Robot Electronics

All calculations and logic for the robot is handled by an Arduino Uno mounted in the middle of the robot.

The speed and direction of the motors are controlled by a H-bridge, based on the L298N chip, that is connected to six General Purpose Input/Output (GPIO) Arduino digital pins, with two of them required to have Pulse Width Modulation (PWM) capabilities. The encoders have two output pins each, A and B. But only one is needed in this application, as the direction of rotation is not needed. The A outputs are connected to two GPIO digital pins on the Arduino and rely on software "Pin Change Interrupts" to handle the fast changes sent out by the encoders.

All the electronics are powered by two AA batteries mounted in battery holders integrated in the 3D-printed bottom plate, and can be accessed through a hatch on the underside of the robot. The voltage from the batteries are increased from circa 3 V to 5.1 V by an adjustable step-up module from Luxorparts. One switch is connected in series with the positive wire from the batteries as a main power switch. Another switch is connected in series with the positive wire going to the Arduino, to keep it from trying to power the robot from the USB-port when uploading new firmware or debugging on a computer.
3.1. HARDWARE

3.1.2 Transmitter and Receiver circuits

The transmitter towers each consist of three US-transducers wired in parallel to transmit ultrasound simultaneously with a wider spread. The towers and an IR-transmitter is connected to a driver circuit mounted on top of an Arduino Uno. This circuit turns a higher voltage of 20 V on and off for the transducers in sync with a 40 kHz signal, and controls which tower that is transmitting. The driver circuit is constructed as an Arduino "shield" and is also supplying 5 V to the Arduino from the higher supply voltage. It is made from a custom milled PCB with all the components soldered on to it.

The receiver is made as a top module for the robot platform where a custom milled PCB is attached to a 3D-printed plate. On the PCB there is an IR-receiver, US-receiver and US-module. The IR-receiver circuit is based on the TSOP 38283 IR receiver diode, the US-module used is US-015 from SparkFun (which is similar to HC-SR04) and the US-receiver is a omni directional piezoelectric film (PVDF) receiver from Measurement Specialties. On the US-module, the standard US-transducers have been removed and the module is only used to amplify, filter, and interpret the signal from the PVDF receiver.
The IR and US signal pins are connected to pin 2 and 3 on the Uno which are capable of external hardware interrupts. This makes it possible to get very fast response from the signals and more accurate time calculation. Complete circuit schematics can be found in appendix B.

### 3.2 Software

The transmitter is programmed to send the signals at different intervals. There is a short delay between tower 1 and tower 2 and then a longer delay after tower 2, before restarting the loop. This ensures that the robot, with the use of a specific algorithm, can know which tower a transmission originated from.

The algorithm for calculating a position based on the transmissions uses two main functions, `get_position()` and `get_distance()`, that both have different “timeouts” to ensure that the receiver gets “in sync” with the towers. These timeouts forces it to break an ongoing measurement if it takes longer than expected for that part of the loop and resets to begin a new measurement. Full Arduino code can be found in appendix A and flowcharts illustrating the full logic of these functions, the robot and the transmitter is seen in figures 3.4 and 3.5.
3.2. SOFTWARE

Figure 3.4: Flowchart of transmitter and receiver logic. Made with Gliffy.
Figure 3.5: Flowchart of robot main logic and the function `get_position()`. Made with Gliffy.
Chapter 4

Testing and Results

4.1 Test 1 - Calibration

The first test was to analyze the limitations and precision of the one-dimensional distance calculation. This was done by comparing calculated values with known measurements along a set of points on a straight line. Any constant or linear errors shown in these measurements was compensated for in the calculation algorithm by adding a constant and a multiplication factor and thus improving achievable accuracy. These calibration tests where performed multiple times throughout the project to keep optimal performance while alternating the hardware.

After multiple series of calibrating the one-dimensional distance measurement for the final system, using an omni directional receiver, the results in Figure 4.1 were achieved. Multiplication factor was set to 331, approximately representing the speed of sound, and an additional constant factor was set to 22.
It can be seen that the distance calculation has a slight error for the shortest and longest distances. This error has a maximum of 11 mm which was obtained at the shortest distance (135 mm). The mean absolute error was calculated to 3.2 mm.

### 4.2 Test 2 - FOV for ultrasound transducers

The second test was to measure the Field Of View (FOV) of the ultrasound transducers. This was conducted by placing the transmitter and receiver facing each other and then moving one of them along a straight line perpendicular to the center line, see Figure 4.2. When the measurements reported by the receiver no longer made any sense, i.e. suddenly increasing by one meter or so, the moved distance from the center line was measured. This step was repeated several times and an average distance was calculated. The definite FOV as degrees from the center line was then calculated by:

\[
FOV = \arctan\left(\frac{\text{distance from center line}}{\text{center line length}}\right).
\]
4.3 TEST 3 AND 4 - ANGLE DEPENDENCY OF US-015 MODULE TX/RX COMPONENTS

![Diagram](image)

Figure 4.2: Test 2; definite Field Of View. Created in paint.net.

The transmitter FOV was calculated to 37°. Distance measurements within this angle were reasonable, but some of them with large errors. Measurements outside of the 37° FOV were completely off. This test was performed using US receivers that are fitted on the US-015 module, not the omni directional receiver.

4.3 Test 3 and 4 - Angle dependency of US-015 module TX/RX components

Even though the receiver and transmitter where in the field of view, it was a noticeable difference in accuracy when the distance from the center line was increased. Therefore tests 3 and 4 where conducted to determine the actual loss of accuracy as a function of the angle from the center line. Both test are, in reality, testing the same kind of difference in angle between the center lines of the object as seen from one of them. But, for practical reasons, where conducted a little differently.

In Test 3, the angle dependency of the receiver was tested by placing the receiver at a known distance from the transmitter and rotating the receiver around its vertical center axis. The increase of the calculated distance as function of the angle was then recorded. The results are in Figure 4.4.
Figure 4.3: Test 3; Increase in measured distance affected by angle to receiver center line. Created in paint.net.

Figure 4.4: Result of Test 3; Error in increased calculated distance due to rotation, $\alpha$, of the receiver. Created with Google Sheets.

Test 4 was conducted by attaching a string between the receiver and transmitter, and then clamping the transmitter to one end of a table. This allowed the receiver to be moved along an equidistant arc around the transmitter, while also making sure it was always pointing towards the transmitter. The angle dependency of the transmitter was then recorded as an increase of the calculated distance as function
4.3. TEST 3 AND 4 - ANGLE DEPENDENCY OF US-015 MODULE TX/RX COMPONENTS

of the angle, $\alpha$ (Figure 4.5), between the center lines. The results are in Figure 4.6.

Figure 4.5: Test 4; Increase in measured distance affected by angle to transmitter center line. Created in paint.net.

Figure 4.6: Error in calculated distance due to transmitter not facing receiver, created with Google Sheets.

These two tests were performed using US transmitters and receivers that are fitted on the US-015 module, not the omni directional receiver that was used in the final version of the robot. Both tests show that increasing the angle between transmitter and receiver will increase the error up to a few centimeters instead of the previous result of a few millimeters when the angle was zero.
4.4 Test 5 - Two-dimensional positioning error

The fifth test was to gather data of the two-dimensional positioning system. This test was conducted by dividing the 1 m² area in front of the two transmitters into a 10 x 10 cm grid. The robot was then placed on these points and its position was calculated. The calculated position was then compared to the actual position and the absolute error in two-dimensions was calculated. If the robot failed to calculate its position due to IR or US signals not being in the line of sight of the receiver sensors, then the transmitting elements would be rotated slightly until a position could be calculated. If these adjustments to the otherwise static transmitting system failed, then the current position was noted as out of range for the positioning system.

Unfortunately, the middle transducer on one of the towers broke during this test and it was noted that the robot had considerable problems to detect both of the transmitters at the same time. For this reason, the transmitters were needed to be rotated often.

![Positioning error](image)

Figure 4.7: Calculated absolute positioning error, created with MATLAB.

Figure 4.7 shows that the positioning system is functioning in approximately a 90 x 70 cm rectangle in front of the transmitters. There is however, a blind spot on the left side for values of Y > 40 cm and a smaller blind spot at X = 80 cm and Y > 60 cm. The accuracy for the two-dimensional positioning ranges from 1 to 5 cm except for a few peaks with larger errors.
4.5 Test 6 - Movement and position verification

The sixth test was to make sure that the robot can find its way to a designated position, simulating a player command. This was tested by commanding the robot to drive 600 mm forward, stop, calculate its position and then verify whether it was within a specified box (14 x 15 cm). If it determined the resulting position a success, it would light a green LED, otherwise a red LED would be lit.

A series of 20 attempts were made while alternating letting the robot drive uninterrupted and pushing it off course. The correct LED was lit for approximately 97% of the attempts. The error in the driven distance was calculated to an average of ±1 mm from the desired 600 mm.

Figure 4.8: Test 6; Successful verification of correct goal position.

Figure 4.9: Test 6; Successful verification of invalid goal position.
Chapter 5

Discussion and conclusion

5.1 Discussion

To answer the first research question it is relevant to analyze the results from test 1 to 4. Test one shows that the best achieved accuracy was a mean error of 3.2 mm and test two that the maximum angle of sight is 37°. For angles between 0° and 37° there is however an increase in error from millimeters to centimeters described by results from test 3 and 4. The conclusion from these tests is that the standard receiver won’t work for a positioning system, incoming signals will always come at an angle. For this reason, an omni directional receiver is needed.

The answer to the second research question can be found by analyzing the results from test 5 and Figure 4.7. It is seen that the FOV for the positioning system is approximately a 90 x 70 cm rectangle in front of the transmitters except for some smaller blind spots. This result is however not a realistic depiction of the system’s FOV, it is more of an optimal FOV. The actual FOV, when not assisting the system by rotating transmitters, is smaller. This is not considered to be a big problem because both these issues are because of the transmitter that was slightly broken. The bigger problem is that the optimal FOV has a short range. There are a few possible reasons as to why the positioning system has such short range. The first is that we are using a piezoelectric receiver instead of the receiver that was originally fitted to the US module. The piezoelectric film is many times larger than the membrane used in the standard receiver, for this reason, the piezoelectric receiver probably needs stronger signals / more energy to give the same input to the US module circuit board. Another reason would be that we have three ultrasound transducers in parallel that are connected to our own designed drive circuit. This circuit allows us to deliver higher voltage to the transducers but most likely not as good as we expected. The circuit should be improved to allow more voltage and better signal stability but there is not enough time or knowledge to do it in a structured manner. When analyzing the accuracy of the positioning system it is seen that the accuracy is remarkably worse than what was achieved for the straight line distance measurement in the first test. The accuracy within the FOV ranges
from 1 to 6 cm except for the few peaks with larger error. The position calculation is more complicated with its two distances and non optimal angles but the difference in accuracy is still very large. This accuracy could be compared to the error that was achieved for the angle dependency in tests 3 and 4, a few centimeters. The positioning system might therefore be affected by the angle dependency of the transmitters. This problem could be reduced by using an omni directional transmitter instead of the three standard transmitters that was used, it would theoretically result in a more uniform distribution of the US signals. The use of Omni directional transmitters was tried during the project. However, we never managed to get them to work despite our efforts with different designs of the driver circuit.

The last research questions are answered by the results of test 6. The robot can move a distance of 600 mm with an average error of $\pm 1$ mm and it could verify its goal position correctly 97% of the attempts. The last result is very good when considering that the positioning system has an error of a few centimeters. It is worth noting that the specified goal box of 15 x 14 cm, which is just slightly larger than the 10 x 11 cm robot, is much larger than absolute error of the positioning system. The accuracy of the system is therefore sufficiently small for this application.

5.2 Conclusion

The concluded answers to the research questions for this project are as following:

- The accuracy achieved for the one-dimensional distance calculation using the time of flight technique was a mean error of 3.2 mm for distances $< 1$ m. The maximum angle of sight for the US-015 module components is 37°, the distance error is however increasing up to a few centimeters when the angle is between 0° and 37°.

- The positioning system has a FOV that is a 90 x 70 cm rectangle in front of the transmitters. The accuracy within this area is between 1 and 6 cm.

- The robot can move a specified distance with a precision of $\pm 1$ mm and then verify its position successfully at a rate of 97%.

The final conclusion of the project is that the positioning system is accurate enough for a RoboRally application with a robot of this size, but needs improvements on its field of view. The motion system of the robot is more than accurate enough for the movements tested here, but should also be improved to make it easier with further development of other functions and RoboRally specifics.
Chapter 6

Recommendations and future work

The first recommendation for other projects, is to consider other technologies than ultrasound for any and all applications. Ultrasound, sound, and all form of wave propagation that needs to be measured exactly is extremely complicated and have many different factors to consider.

Future work is to improve the transmitters in terms of range and FOV. The most feasible way for this, would be making the omni directional transmitters work. Improving the drive circuit for increased power (higher voltage) might work for this. Otherwise the drive circuit needs to be further developed, for example by comparing it more thoroughly to the schematics supplied by the manufacturer of the transmitters. It could be worth contacting the manufacturer who also might help with some insight with what the problem could be.

A few IR-LEDs should also be added for increased visibility, this has already been prepared for in the current drive circuit with a connector where they can be connected. A schematic with an example circuit of external IR-LEDs can be found in appendix B.4. The transmitter loop should also be improved in terms of speed and reliability, this would allow the robot to receive both better measurements and more often. Ultimately the robot should be able to calculate a position, within some margin, while moving.

The robot construction could also be improved by changing the wheel configuration. Omni-wheels and Mecanum wheels would be a better solution because they allow the robot to move in 2D without rotating the robot. But the different wheel options have their own advantages and disadvantages. The Mecanum wheels allow movement back and forth without slipping, and can therefore keep the accuracy achieved by the encoders for moving a certain distance. The Omni-wheels instead allows accurate rotation without slipping, but can’t make any linear moves without slipping. Our recommendation for a RoboRally application would therefore be the Mecanum wheels as accurate rotation possibly could be solved by adding a 9 degrees of freedom internal measurement unit sensor, which makes it possible to get a relative and absolute angle of rotation of the robot.

The formula for how the robot brakes after a move should be changed to a
algorithm that varies the braking distance and time according to the actual speed of the robot. Now it is only calibrated for the robot’s own weight and any additional load will cause the robot to stop ahead of its destination. By using the speed of the incoming pulses from the encoder, the current speed and acceleration of the robot could be calculated and taken into account for, in the braking algorithm.

One alternative is to use a worm gear instead of a planetary gear head on the motors. This would make the wheels lock completely when the motors stop and thus removing the need of putting the motors in reverse to get enough braking force, which might simplify the required braking algorithm. Worm gears would also make it possible to use even smaller motors, which in turn makes it possible to shrink the entire robot platform.

Shrinking all other modules on the robot can be achieved by making one custom circuit board that holds all the individual components for power management, motor control and the receiver, which connects directly to an Arduino Nano instead of the Uno.
Bibliography


Appendix A

Code

A.1 Robot/receiver code

```c
/* Base code for using interrupts from https://www.allaboutcircuits.com/technical-articles/using-interrupts-on-arduino/
Accessed 2018-02-16
and then modified by Niclas Bassili and Erik Blomqvist *

// Hardware connections:
// IR and US:
#define IR_Pin 2 // Pin connected to the signal pin of the IR receiver
#define trigPin 4 // Pin connected to the trigger pin on the US-module
#define US_echo_Pin 3 // Pin connected to the echo pin on the US-module

// LEDs
#define green 16
#define yellow 15
#define red 14

// H-bridge:
#define motors_pin1 5
#define motors_pin2 6
#define motors_pin3 7
#define motors_pin4 8
#define motors_pinA 9
```

24 #define motors_pinB 10
25
26 // Encoders on motors:
27 #define encoder1PinA 12
28
29 #include <Motorcontrol.h> // Library for controlling two DC-motors connected to custom h-bridge module based on the L298n chip. Library written by Niclas Bassili.
30 Motorcontrol motors(motors_pin1, motors_pin2, motors_pin3, motors_pin4, motors_pinA, motors_pinB); // Creates object called motors that have methods for moving the robot
31
32 #define wheelDia 40.69 // Wheel diameter in mm for the wheels attached to the motors.
33
34 // Standard values for functions with optional arguments:
35 double get_distance(long timeout = 10000); // Standard timeout of 10s
36
37 // Variables that will change in an interrupt subroutine:
38 volatile long starttid = 0; // Variable for saving the starting time
39 volatile long stopptid = 0; // Variable for saving the stopping time
40 volatile int skrivut = 0; //
41 volatile int startad = 0; //
42 volatile int stopped = 0; //
43 volatile long encoder1Pos = 1; // Number of pulses detected from encoder
44 volatile int encoder1 = 0;
45
46 // Variables that need to match the Transmittercode
47 int burst = 40; // The number of (complete waves of) Ultrasound and IR signals transmitted
48 int burstAvg = 15; // The number of measurements averaged on each distance measurement
49 int burstTime = 420; // The maximum time to receive all signals from one burst, one burst is approximately 10 mesurements *(10+0.25)ms = 102.5ms
50 int tower_delay = 500; // Time between tower 1 and tower 2 transmissions
51
52 // Global times to sync everything properly for calculating the position
A.1. ROBOT/RECEIVER CODE

```c
53 int dissipateTime = 100; // (burst - burstAvg) * burstTime / burst;
54 long burstAndDelayTime = tower_delay - dissipateTime + burstTime; // time to wait for a signal from tower 2 before timeing out and assuming next signal is from tower 1
55
56 // Pin change interrupt libraries: ( arduino-pcimanager URL: https://github.com/prampec/arduino-pcimanager )
57 #include <PciManager.h>
58 #include <PciListenerImp.h>
59
60 // Create PCI listener:
61 void encoder1PinChange(byte changeKind);
62 PciListenerImp listener(encoder1PinA, encoder1PinChange);
63
64 void setup() {
65 // Initialize the serial communication for debugging purpose
66   Serial.begin (9600);
67
68 // initialize the pins as inputs and outputs:
69   pinMode(IR_Pin, INPUT);
70   pinMode(US_echo_Pin, INPUT);
71   pinMode(trigPin, OUTPUT);
72   pinMode(green, OUTPUT);
73   pinMode(yellow, OUTPUT);
74   pinMode(red, OUTPUT);
75   pinMode(motors_pin1, OUTPUT);
76   pinMode(motors_pin2, OUTPUT);
77   pinMode(motors_pin3, OUTPUT);
78   pinMode(motors_pin4, OUTPUT);
79   pinMode(motors_pinA, OUTPUT);
80   pinMode(motors_pinB, OUTPUT);
81
82 // Attach an interrupt to the ISR vector
83 attachInterrupt(digitalPinToInterrupt(IR_Pin), pin_ISR1, FALLING); // Modes according this documentation https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/
84 attachInterrupt(digitalPinToInterrupt(US_echo_Pin),
85   pin_ISR2, FALLING);
```

33
APPENDIX A. CODE

digitalWrite(trigPin, LOW);  // make sure trigpin is low

// Register PCI listener:
PCIManager.registerListener(&listener);

Serial.println("Ready.");
digitalWrite(green, HIGH);  // Turn on the green LED to indicate that the arudino has booted properly and will now start executing the program

void loop() {
  //Blink yellow light as warning before moving.
digitalWrite(yellow, HIGH);
delay(1000);
digitalWrite(yellow, LOW);
delay(1000);
digitalWrite(yellow, HIGH);
delay(1000);
digitalWrite(yellow, LOW);
delay(1000);
digitalWrite(yellow, HIGH);

double pos[2];
double goalPos[2];

  // Get starting position: ########### Edit/Insert functions here for continued development on what the robot can do. ###########
  // get_position(pos);
  //goalPos = ....  //Calculating goal position relatively where it is

  //Move forward 3 "steps"
digitalWrite(green, LOW);
driveForward(600, 100);  // 200mm/step = 600mm for 3 steps with motor power = 100/255

  // Get current position after movement
get_position(pos);
digitalWrite(yellow, LOW);
A.1. ROBOT/RECEIVER CODE

// Example goal position square center = (80,30)

// Edit/Insert functions here for continued
development on what the robot can do.

// Calculate absolute error in distance from the goal
position

double ErrorX = pos[0] - goalPos[0];
Serial.print("X_error = ");
Serial.println(ErrorX);
double ErrorY = pos[1] - goalPos[1];
Serial.print("Y_error = ");
Serial.println(ErrorY);
double AbsError = sqrt(pow(ErrorX, 2) + pow(ErrorY, 2));
Serial.print("ABS_error = ");
Serial.println(AbsError);

if (AbsError < 100) { // 10 cm
digitalWrite(green, HIGH);
}
else {
digitalWrite(red, HIGH);
}

while (1) { // Edit/Insert functions here for continued
development on what the robot can do.

// Endless loop to stop executing commands here
}

// Calculate numbers of turns made by one wheel

// Edit/Insert functions here for continued
development on what the robot can do.

/*
if (encoder1) {
float enclturns = (float) encoder1Pos / (400 * 14);
Serial.print("pci : ");
Serial.println(enclturns);
encoder1 = 0;
}
*/
// main loop end
// Interrupt subroutines called when hardware interrupts are triggered:
void pin_ISR1() {
  starttid = micros(); // Record current time since boot
  startad = 1; // Set flag that IR has been received
  digitalWrite(trigPin, HIGH); // turn on trigger for at least 10us
  delayMicroseconds(11);
  digitalWrite(trigPin, LOW);
}

void pin_ISR2() {
  stopptid = micros(); // Record current time since boot
  stopped = 1; // Set flag that US has been received
}

// Interrupt subroutines called when pin change interrupts are triggered:
void encoder1PinChange(byte changeKind) {
  // -- changeKind can be: CHANGEKIND_HIGH_TO_LOW, or CHANGEKIND_LOW_TO_HIGH and will have the value 1 or 0.
  if (changeKind) { // This subroutine is called whenever the pin changes its status, only count positions when the pin goes high.
    encoder1Pos++;
    encoder1 = 1; // Set flag that a change has occurred.
  }
}

// Basic function for calculating the average of several measurements
double average (double * measurments, int len) {
  double sum = 0;
  for (int i = 0 ; i < len ; i++) {
    sum += measurments[i];
  }
  return sum / len;
}
// Function for calculating the distance between US-transmitter and US receiver from the time difference

double get_distance(long timeout) {
    // ######################### WARNING; Enabling any Serial prints in this function will affect the performance of it and the ability to successfully sync with the transmitted signals. (Serial.print is really slow and makes HUGE difference for each loop time)
    #########################

    Serial.print("Timeout = ");
    Serial.println(timeout);

    int signals = 0;
    double measurements[burstAvg]; // Create an empty array with the correct size where the measurements can be saved
    long time_a = millis(); // Time since boot for when the function was entered
    long time_b = millis(); // Time since boot to be updated on each loop
    long time_c = millis(); // Time since boot for when the first measurement was received
    long time_d = millis(); // Time since boot to be updated on each loop after the first measurement has been received

    while (((time_b - time_a) < timeout) && ((time_d - time_c) < burstTime)) { // Break the loop if it took too long to receive any/all signals.
        if (signals < burstAvg) { // Receive a burst of signals from the receiver and then break the loop
            if (stopped && startad) { // wait for both interrupt signals
                if (signals == 0) {
                    time_c = millis(); // Record time since boot for when the first measurement is received
                }
            }
        }

        double duration = stopptid - starttid;
        double compensation = 22; // Compensation in mm to
account for various manufacturing differences in the US-modules.

```cpp
double distance = 331 * (duration) / 1000 + compensation; // Calculation of distance in mm with speed and compensation calibrated through testing.

if (distance > 0 && distance < 3400) {
    measurements[signals] = distance; // Save each measured distance
    Serial.print("Distance = ");
    Serial.println(measurements[signals]);
    signals += 1; // Count the number of successful measurements
    Serial.println(signals);
} else {
    Serial.print("Out of range, d = ");
    Serial.println(distance);
}

// Reset flags:
stopped = 0;
startad = 0;
delay(1); // Delay to let any weird spikes from the US-module fade away.
```

```cpp
time_b = millis(); // Update to current time elapsed after each loop.
if (signals > 0) { // if the first signal has been received
    time_d = millis(); // Update to current time elapsed after each loop.
}
else {
    break;
}
```

```cpp
if ((time_b - time_a) < timeout && (time_d - time_c) < burstTime) {
    double distance_av = average(measurements, signals);
    return distance_av;
}
```
A.1. ROBOT/RECEIVER CODE

```c
257   } else {
258       return 0;
259   }
260
261
262
263
264 // Function for calculating the position on the playing field
265 void get_position(double pos[]) {
266     // Size of the playing field:
267     double Lx = 1045; // [mm] // Must be the distance between the emitters in the two towers
268     double Ly = 1000; // [mm] // Not used in any calculations now ####### Edit/Insert functions here for continued development on what the robot can do. #######
269
270     double distance1 = 0;
271     double distance2 = 0;
272
273     while (1) {
274         distance1 = get_distance();
275         if (distance1 > 0) {
276             Serial.print("Distance to tower 1: ");
277             Serial.print(distance1);
278             Serial.println(" mm");
279             delay(dissipateTime); // allow all signals from tower 1 to disappear before listening for tower 2
280         }
281         distance2 = get_distance(burstAndDelayTime);
282         if (distance2 > 0) {
283             Serial.print("Distance to tower 2: ");
284             Serial.print(distance2);
285             Serial.println(" mm");
286             break;
287         }
288     }
289
290     // Syntax pow: pow(distance,2) == distance^2
291     double x = (pow(distance1, 2) + pow(Lx, 2) - pow(distance2, 2)) / (2 * Lx); // Calculate the X-coordinate
```
double y = sqrt(pow(distance1, 2) - pow(x, 2)); // Calculate the Y-coordinate
pos[0] = x;
pos[1] = y;
if (x > 1 and y > 1) {
    Serial.println("Position: ");
    Serial.print("x = ");
    Serial.println(pos[0]);
    Serial.print("y = ");
    Serial.println(pos[1]);
}
}

// Function for driving the robot a set distance forward:
void driveForward(int distance, int power) {
    int currentPos = encoder1Pos; // Store current number of
    // encoder pulses recorded before this movement
    float turnsRequired = distance / (wheelDia * PI); // Calculate the required number of turns of the wheel to
    // move the set distance
    int endPos = currentPos + turnsRequired * 400 * 14; // Calculate total number of encoder pulses to have been
    // recorded when the correct distance have been moved. (Gearhead with ratio 14:1 and encoder with 400 pulses/turn)
    for (int ramp_up = 1; ramp_up < power; ramp_up++) { // Gradually increase the power to the motors to get a soft
        motors.forward(ramp_up);
        delayMicroseconds(50);
    }
    while (encoder1Pos < (endPos - 600)) { // Edit/Insert functions here for continued development on
        // what the robot can do. 
        // wait to reach the desired moved distance minus a
        // small distance for breaking.
    }
    motors.brake(); // Active breaking by shorting out the
    delay(50); // Edit/Insert functions here
    // for continued development on what the robot can do.
A.1. ROBOT/RECEIVER CODE

```c
for (int ramp_up = 1; ramp_up < 255; ramp_up += 1) { // Increase retardation by gradually increase power to the motors to move in reverse while trying to prevent a hard jerk.
    motors.reverse(ramp_up);
    delayMicroseconds(50);
}

delay(30); // ########## Edit/Insert functions here for continued development on what the robot can do. ##########

for (int ramp_down = 255; ramp_down >= 0; ramp_down -= 5) { // Gradually decrease power to the motors to try to prevent a hard jerk.
    motors.reverse(ramp_down);
}

motors.brake(); // Active breaking to prevent the robot to bounce back after putting motors in reverse.

delay(100);
motors.stop(); // Turn off all motors to conserve power
```
A.2 Transmitter code

1 /*This code is run on a Arduino that controls the
2 transmitting towers for the "LOST Positioning" sytem
3 * Developed by Niclas Bassili and Erik Blomqvist as a
4 bachelor thesis project.
5 * Royal Institute of Technology, KTH, Stockholm 2018-05-16.
6 */
7
8 // Hardware connections:
9 #define tower1 10
10 #define tower2 11
11 // Pin 9 is used by the timers and registers edited by below
12 code and is also connected to the shield.
13
14 //Code supplied by : Coding Badly on December 6th , 2012 via
15 the Arduino
16 // forum website .
17 // https://forum.arduino.cc/index.php?topic=135847.0,
18 retrieved 2018/05/16
19 // This code modifies the internal clocks of the Arduino ,
20 thus allowing for
21 // timers to be changed and to output a square wave of
22 varying frequency.
23
24 void analogWriteSAHInit(void)
25 {
26 // Stop the timer while we muck with it
27 TCCR1B = (0 << ICNC1) | (0 << ICES1) | (0 << WGM13) | (0
28 << WGM12) |
29     (0 << CS12) | (0 << CS11) | (0 << CS10);
30 // Set the timer to mode 14...
31 // Mode WGM13 WGM12 WGM11 WGM10
32 // CTC1 PWM11 PWM10
33 // ---- ----- ----- ----- -----
34 // 14 1 1 1 0
35 //
36 // Timer/Counter Mode of Operation
37 // ----------------------------
38 // Fast PWM
39 //
40 //
41
A.2. TRANSMITTER CODE

34 // TOP  Update of OCR1x at TOV1
35 // ---- -----------------------
36 // ICR1  BOTTOM
37 //
38 // Flag Set on
39 // ---------
40 // TOP
41 //
42 // Set output on Channel A to...
43 //
44 // COM1A1 COM1A0  Description
45 // ------ ------
46 // 1 0  Clear OC1A/OC1B on Compare Match (Set output to low level).
47 //
48 TCCR1A =
49 (1 << COM1A1) | (0 << COM1A0) | // COM1A1, COM1A0 = 1 , 0
50 (0 << COM1B1) | (0 << COM1B0) |
51 (1 << WGM11) | (0 << WGM10) ; // WGM11, WGM10 = 1 , 0
52 //
53 // Set TOP to . . .
54 //
55 // f c l k I /O = 16000000
56 // N = 1
57 // TOP = 799
58 //
59 //
60 // fOCnxPWM = f c l k I /O / (N * (1 + TOP))
61 // fOCnxPWM = 16000000 / (1 * (1 + 799))
62 // fOCnxPWM = 16000000 / 800
63 // fOCnxPWM = 20000
64 //
65 ICR1 = 398; //modified to 398 from retrieved code (799) to make it to 40.1kHz
66 //
67 // Ensure the first slope is complete
68 TCNT1 = 0 ;
69 // Ensure Channel B is irrelevant
70 OCR1B = 0 ;
71 // Ensure Channel A starts at zero/off
72 OCR1A = 0 ;
73 // We do not need no stinkin interrupts
74 //
75 //
76 //
77 //
78 //
79 //
80 //
81 //
82 //
83 //
84 //
85 //
86 //
87 //
88 //
89 //
90 //
91 //
92 //
93 //
94 //
95 //
96 //
97 //
98 //
99 //
100 //
101 //
102 //
103 //
104 //
105 //
106 //
107 //
108 //
109 //
110 //
111 //
112 //
113 //
114 //
115 //
116 //
117 //
118 //
119 //
120 //
121 //
122 //
123 //
124 //
125 //
126 //
127 //
128 //
129 //
130 //
131 //
132 //
133 //
134 //
135 //
136 //
137 //
138 //
139 //
140 //
141 //
142 //
143 //
144 //
145 //
146 //
147 //
148 //
149 //
150 //
151 //
152 //
153 //
154 //
155 //
156 //
157 //
158 //
159 //
160 //
161 //
162 //
163 //
164 //
165 //
166 //
167 //
168 //
169 //
170 //
171 //
172 //
173 //
174 //
175 //
176 //
177 //
178 //
179 //
180 //
181 //
182 //
183 //
184 //
185 //
186 //
187 //
188 //
189 //
190 //
191 //
192 //
193 //
194 //
195 //
196 //
197 //
198 //
199 //
200 //
201 //
APPENDIX A. CODE

TIMSK1 = (0 << ICIE1) | (0 << OCIE1B) | (0 << OCIE1A) |
(0 << TOIE1);

// Ensure the Channel A pin is configured for output
DDRB |= (1 << DDB1);

// Start the timer . . .

//

// CS12 CS11 CS10 Description
// ---- ---- ----
// 0 0 1 c l k I/O/1 (No prescaling)

TCCR1B =
(0 << ICNC1) | (0 << ICES1) |
(1 << WGM13) | (1 << WGM12) // WGM13, WGM12 = 1, 1
(0 << CS12) | (0 << CS11) | (1 << CS10);

}

void setup() {
analogWriteSAHInit();

pinMode(tower1, OUTPUT);
digitalWrite(tower1, HIGH);

pinMode(tower2, OUTPUT);
digitalWrite(tower2, HIGH);
}

void loop() {

// Send from tower 1:
digitalWrite(tower1, LOW); // Activate tower 1 by
opening a pnp transistor
delay(1);

for (int varv = 1; varv <= 40; varv += 1) {
analogWriteSAH(200); // Start sending IR and
US
delayMicroseconds(250); // Send during 250
microseconds = 10 pulses
analogWriteSAH(0); // Stop sending IR and US
delay(10);
}

digitalWrite(tower1, HIGH); // Deactivate tower 1 by
closing the pnp transistor
delay(500); // Wait before sending
from the other tower

// Send from tower 2:
digitalWrite(tower2, LOW); // Activate tower 2 by
A.2. TRANSMITTER CODE

opening a pnp transistor

```
112  delay(1);
113  for (int varv = 1; varv <= 40; varv += 1) {
114      analogWriteSAH(200); // Start sending IR and US
115      delayMicroseconds(250); // Send during 250 microseconds = 10 pulses
116      analogWriteSAH(0); // Stop sending IR and US
117      delay(10);
118  }
119  digitalWrite(tower2, HIGH); // Deactivate tower 2 by closing the pnp transistor
120  delay(1000); // Wait longer before sending from both again
121 } // Loop end
122
123
124
125 void analogWriteSAH(uint16_t value)
126 {
127      if ((value >= 0) && (value < 400))
128          OCR1A = value ;
129  ```
Appendix B

Schematics

B.1 Receiver circuit schematics

Figure B.1: Circuit schematics for the ultrasound receiver board, made with Eagle CAD.
B.2 Robot hardware coupling

Figure B.2: Simplified visual representation of how the robot is wired. Made in the free software Fritzing.
B.3 Transmitter circuit schematics

Figure B.3: Circuit schematics for the ultrasound transmitter driver, made with Eagle CAD.
B.4 Example circuit external IR-LEDs

Figure B.4: Example circuit of external IR-LEDs that can be connected to the driver shield. Made with Eagle CAD.