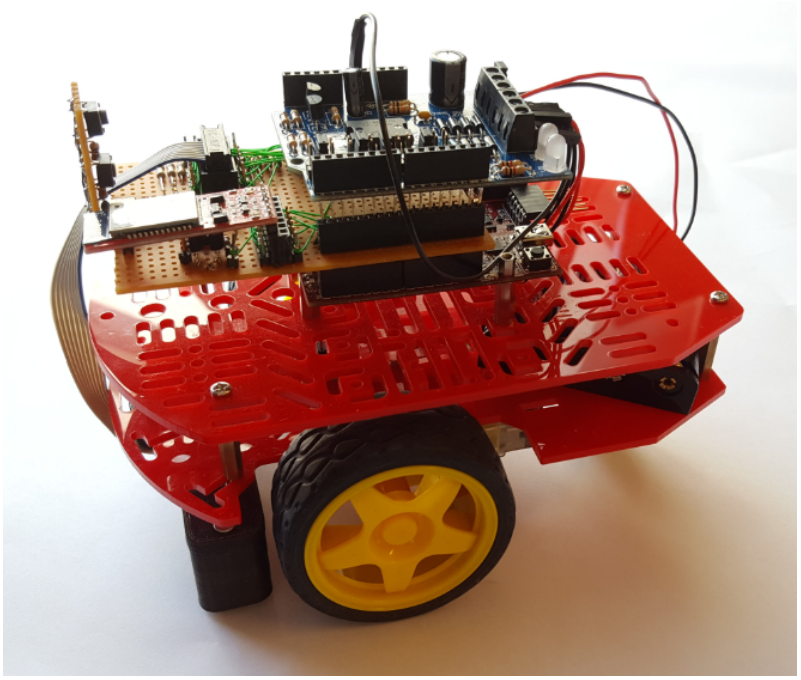
Fall Semester 2016

# Autonomous Object Avoidance Robot

Group 2

3. Semester IT-Technology



Group members: Benjamin Nielsen - Henrik Jensen - Martin Nonboe - Nikolaj Bilgrau

Supervisor: Jesper Kristensen - [1] Steffen Vutborg

FiXme Fatal: er han?

[1] FiXme Fatal: er han?

Uddannelse i virkeligheden

**IT-technology**
Sofiendalsvej 60
9200 Aalborg SW
*http://www.ucn.dk/*

Title:

Autonomous Object Avoidance Robot

Project Period:

3. Semester | Fall semester 2016

Projectgroup:

Group 2

Group participants:

Benjamin Nielsen
Henrik Jensen
Martin Nonboe
Nikolaj Bilgrau

Supervisors:

Jesper Kristensen
Steffen Vutborg

Pages:

Appendices:

Completed:

# Preamble

TBD fyld mere på? This project was written by group 2, for the 3rd semester on the IT-electronics education at university college Nordjylland, Sofiendalsvej 60. The project goal is to make an autonomous robot that can navigate a course utilizing object avoidance and localization.

| | |
|---|---|
| Benjamin Nielsen | Henrik Jensen |
| Martin Nonboe | Nikolaj Bilgrau |

# Table of Contents

# Glossary

3D print 3-Dimensional printing

ADC Analog-digital conversion

GUI Graphical User Interface

IDE Integrated Development Environment

MCU Microcontroller Unit

PCB Printed Circuit Board

PID Proportional-integral-derivative

PWM Pulse-width modulation

THT Through-hole-technology

UART Universal Asynchronous Receiver/Transmitter

# Introduction 1

In countries with high wages and where manual labour is expensive, the industrial production is often organized as an automated process. To make the whole industry smarter and more customizable, new automated robots and reliabilities are needed. The many new forms of robots rely on sensors to face the many different challenges. Automation of movement and avoidance enables even robotic space exploration, as seen in the many rovers visiting the different nearby planets.

There are different sensors in play when needing to avoid obstacles or collision. To mention a few, ultrasound, infra-red and laser sensors comes to mind, all of which are viable picks when building a robot with object avoidance.

In the project at hand, we will be focusing mainly on the ultrasound sensor for building an object avoiding robot. The objective of this project is to design and implement an automotive robot capable of autonomous object manoeuvring, specially a collision avoiding robot employing light detecting sensing and ultrasound sensing.

The project was handed to the group at the start of third semester and is to be handed in at the 9th of January, 2017.

# Analysis 2

This section will be focused on analysing any problems the group may face, how to approach them, and how they should be handled.

## 2.1 Problem statement

The problem presented to the group is how to make a robot move from point A to point B, with the help of different sensors, including ultrasound and infrared, and to make use of autonomous algorithms to avoid obstacles.

Problem statement:

- Bot should be able to move from A to B
- Should be able to stop at a predetermined point
- Manoeuvre around obstacles

## 2.2 Problem analysis

### 2.2.1 Mobility from A to B

The robot receives a coordinate to reach, and will use its own starting point to determine a direction to drive towards the given coordinate. The robot will need a way to control its movement and direct current to function optimal.
The robot needs a way to effectively regulate speed and also steer itself autonomously. To dictate how quickly the robot moves, the robot will need some system that allows it to move around on a flat surface, the robot needs to be able to move around from point A to point B. .

### 2.2.2 Predetermined end point

After starting, the robot needs to know when to stop. The pre-determined end-point consists of a series of circles which the robot needs to detect.

### 2.2.3 Obstacles avoidance

As part of its functionality, the robot needs to be able to see objects that are in front of it and avoid them.

## 2.3 Behaviour

FiXme Fatal: algorithm [1]

[1] FiXme Fatal: algorithm

# Requirements specification 3

FiXme Fatal: mere fyld

[1] This section specifies the requirements. The requirements have been found through the analysis.

- The robot needs line following capabilities
- The robot needs object avoidance
- The robot should make use of an H-bridge
- The robot should make use of Motors
- The robot needs a way to implement motor control
- The robot should make use of a micro-controller unit
- The robot should make use of the Magician chassis

[1] FiXme Fatal: mere fyld

# Hardware section 4

## 4.1 Description of the hardware structure and functionality

[1] In this section the different components of the hardware will be listed, described and explained.

FiXme Fatal: mere fyld og billede

## 4.2 Hardware diagram

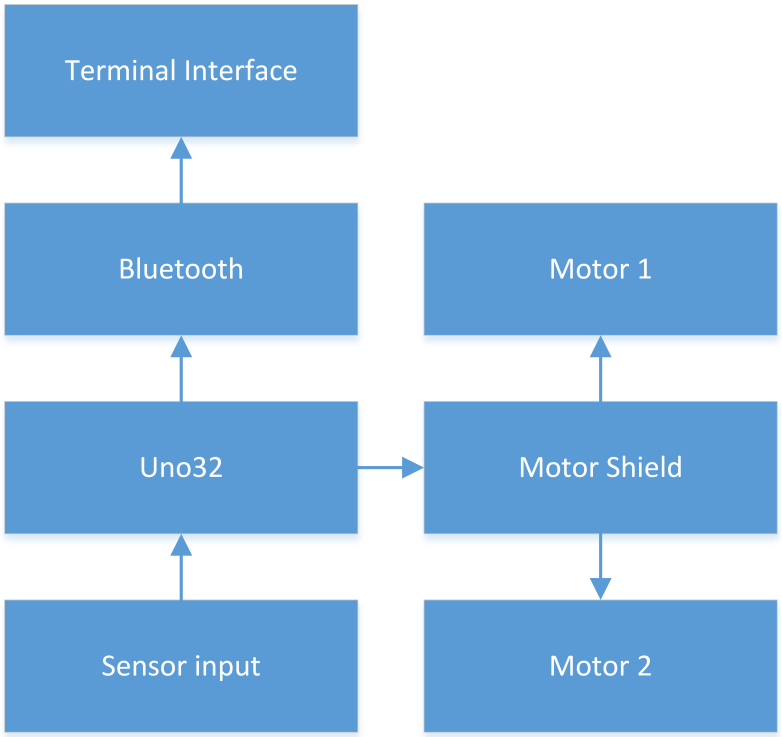



Figure 4.1: Hardware diagram with arrows

The micro controller is connected to the motor-shield. The motor-shield is then connected to the two motors, called motor 1 and motor 2 and is powering both motors. The micro controller is receiving data from the 3 sensor sets; the ultrasound sensors, infrared light-sensors and the tachometer sensor. The micro controller then receives the data from the sensors and sends it further to the Bluetooth transceiver and then the Bluetooth transceiver will send it to the interface.

[1] FiXme Fatal: mere fyld og billede

## 4.3 Sensors and sensor concept

The robot will utilize two sets on sensors - one set of QRE1113 sensors, which will be used for line-following capabilities, they are fastened towards the end of the robot, and will give the robot a way to detect what surface it is about to enter.
The second is a hybrid set of ultrasound and infrared sensors. These will be working together to make the robot able to navigate open spaces more precisely, since infrared and ultrasound sensors work the best under different circumstances. This will end up as a product which is more optimized for usage in situations that would not be ideal for one of the other, since the hybrid design will leverage shortcomings of a given sensor method.

### 4.3.1 Choice of sensors

FiXme Fatal: everything [2]

### 4.3.2 Ultrasound sensor - HC-SR04

When a robot should be able avoid obstacles it will need a device to inform the robot where it's position is compared to the obstacle. This is where an ultrasound sensor plays an important role. For this task the HC sr04 has been picked.


Figure 4.2: The HC-SR04 ultrasound sensor

FiXme Fatal: Mere om denne sensor [3]

The way the ultrasound sensor works is by emitting acoustic waves and then waits for the waves to reflect back to the sensor. The waves are often at about 40 kHz and humans are unable to detect the sounds because of the frequencies being above the human audible range.
What is causing the device to make ultrasonic sound is a piezoelectric crystal. The crystal is receiving a rapid oscillating electrical signal, this causes the crystal to

[2] FiXme Fatal: everything
[3] FiXme Fatal: Mere om denne sensor

expand and contract and thereby creating a sound wave. The sound waves will then after being reflected return to a piezoelectric receiver which can then convert the waves into voltage by using the same method as explained above.

There are several popular ways to process the information gathered from the ultrasound sensor.

- Time of flight
- Doppler shift
- Amplitude attenuation

In the scope of the project, the robot will be using "Time of flight" for sensing the distance between itself and the obstacle.
When working with the term time of flight, it means the ultrasound sensor only generates pulses of sound instead of an continuous streak of sound waves. to avoid confusion. In high speed situations this will mean there is waiting time limits.

The calculation for using the ultrasound sensor is:

t = time
r = distance travelled
c = speed of light

r= c*t

With this the robot can calculate the time of flight.

**Considerations:**

When using the ultrasound as a sensing tool, there are some factors that must be taken into consideration.
Temperature and humidity can affect the speed of sound, just as air currents have been known to be able to create invisible boundaries that can reflect ultrasonic waves.
Ultrasound sensors have something called a dead zone, this occours when an object is infront of the sensors and the receiver can't keep up.
Some materials are very absorbent, which will result in less reflected ultrasound to be detected by the receiver.

**Mounting**

TBD billede med lille tekst

## 4.4 The chipKIT Uno32 board

FiXme Fatal: section need readover

[4] The robot will utilize the chipKIT Uno 32 board to execute code. The board was chosen both due to past experiences, but also because the robot would need line-following properties, and we had already written a functional line-following robot previously, which also included some important features, including PID control and pulse-width modulation patterns.
This enabled a lot of recycled code, which was a strong point in the Uno 32's favor due to time constraints.

The board is also compatible with Arduino shields, and as such designing the H-bridge for it becomes more straightforward. It's fast enough to execute the code, and works well within the input power the robot will utilize.

## 4.5 The motor shield

The motor shield is the single add on board used in the project, and contains all the features needed for making the robot work. The features are:

- Dual H-bridges.
- Low side current sensor for each H-bridge.
- CPLD for reconfigurable H-bridge logic control.
- All connectors needed for sensors and other units needed:
  - Screw terminal for motor connection.
  - Screw terminal for input power.
  - Molex connector for ultrasound distance Sensor.
  - Molex connector for distance sensor.
  - Molex connector for motor encoder.
  - Molex connector for infrared light sensor.
  - Header for Bluetooth module.

FiXme Fatal: INDSÆT BILLEDE AF SHIELD [5]

### 4.5.1 The H bridge

The robot will make use of an H-bridge. An H-bridge is a circuit made for controlling the motor of the robot, by making sure the motor will never try to do forward and backward motion and cause errors or short circuits. The point of using an H-bridge is to ensure motor safety and functionality.

[4] FiXme Fatal: section need readover
[5] FiXme Fatal: INDSÆT BILLEDE AF SHIELD

### 4.5.2 Pololu 100:1 Micro Metal Gearmotor 6V High Power

This is a small motor, drawing 120mA when there is no load, and 1600mA when stalling. They run up to 320 RPM; because of this, the robot will be able to move very rapidly. The ones used for the robot have an extended motor shaft, which makes it possible to utilize the Pololu motor encoders.

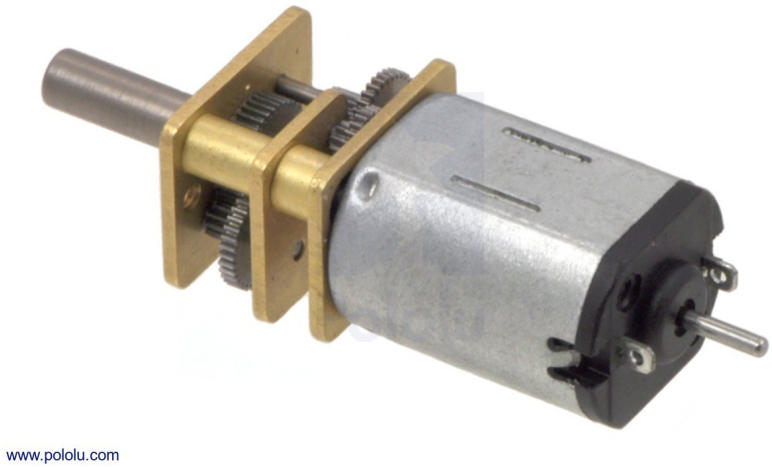



Figure 4.3: Pololu micro metal gear motors

#### Pololu Magnetic Encoder Pair Kit

These encoders allow the transmission of data based on motor movement. It works by having the encoder board count the revolutions of the magnetic disc mounted on the board. It does this twelve times per revolution.

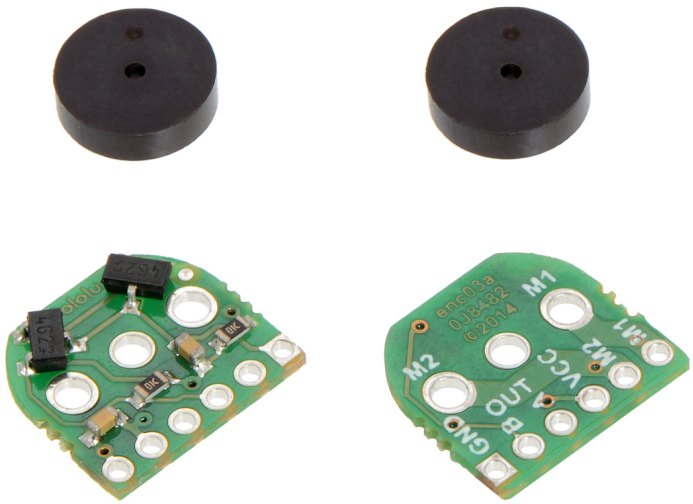



Figure 4.4: Pololu Magnetic Encoder Pair Kit

## 4.6 The Bluetooth tranceiver

The robot will utilize the BlueSmiRF Silver bluetooth tranceiver. The transceiver is made by Sparkfun, it is utilized to make use of an GUI, by sending data from the MCU to the computer (GUI) by the use of bluetooth.
The bluetooth tranceiver makes it possible to monitor both the inputs and the logic behind the steering. The baudrate is between 2400-115200 bps and the tranceiver can be powered from 3.3v up to 6v.

## 4.7 Part conclusion

After initial H-bridge problems, the rest of the process of building the robot went according to plan, and there were no future issues. The robot utillize a range of components which have been used for previous projects, which made the project much more simple to work with. This eliminated some of the learning curve that the previous robot presented, and made it possible to plan out and assemble the robot very rapidly, even though a lot of time was spent waiting for components for the motor shield, and the faulty components. This way, a lot more time could be used on programming and other software solutions.

# Software section 5

The following section will introduce the most important parts of the software for the project. The design of the software will be shown as a flow chart and described. The section will focus on the feedback loop used, but will also go through some of the smaller parts, such as PWM and the code used for the CPLD.
Pulse Width Modulation is a very effective and straightforward way to control the speed of the robot rapidly. It works by limiting how long the of a given period the power is 'on' compared to 'off'. PWM is used by utilizing Output Compare on the chip, which lets the chip generate pulses based on a timer. This is initialized in the following way:

## 5.1 Software flowchart

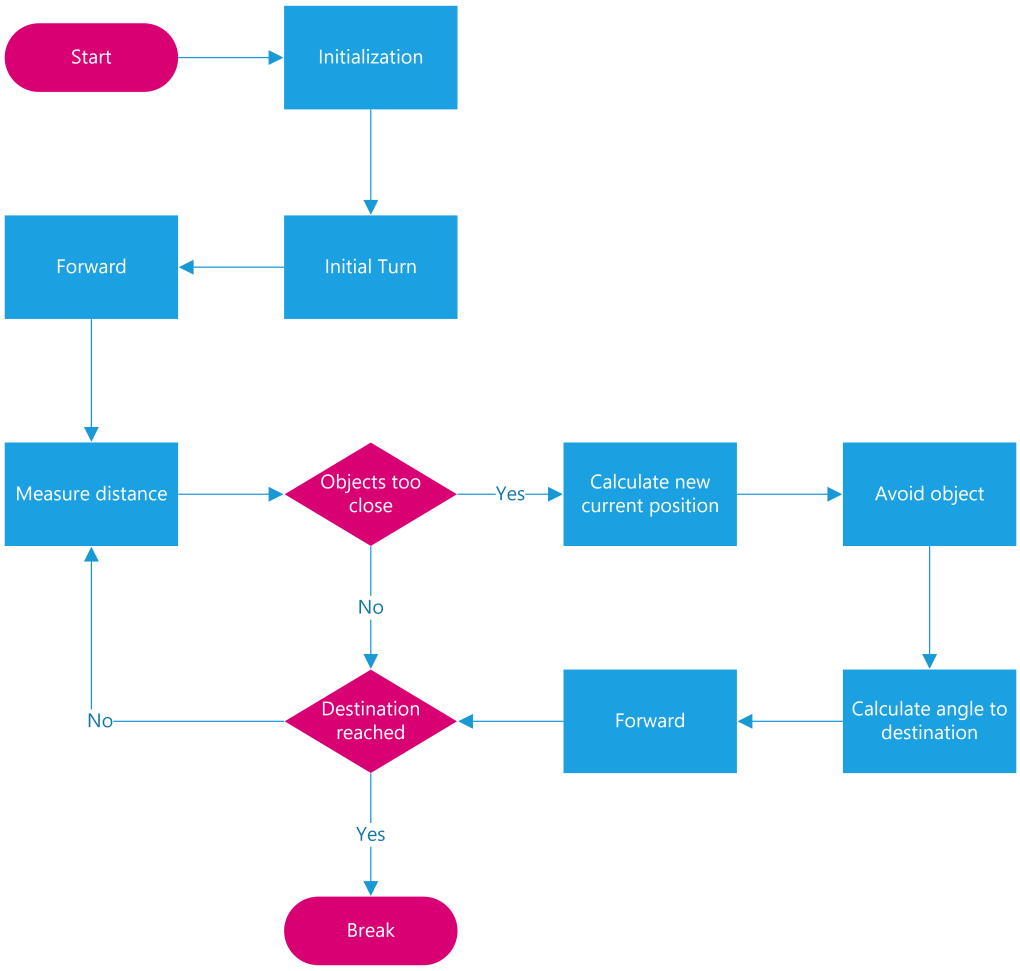



Figure 5.1: Software flowchart

## 5.2 Pulse-width modulation

Pulse Width Modulation is a very effective and straightforward way to control the speed of the robot rapidly. It works by limiting how long the of a given period the power is 'on' compared to 'off'.[1]

FiXme Fatal: virker lidt underligt? gentager intro direkte

```
1  void initPWM() {
2      int sysClk = 80000000; //FPB
3      int pwmFreq = 1000; //Desired frequence
4      int prescaleV = 1;
5      int dutyCycle = 0;
6
7      PMCONbits.ON = 0; //TBD bruger vi det?
8      PMAEN = 0;
9
10     OC4CON = 0x0000; //Turn off the Output Compare while setting
           up
11     OC4R = 0x00638000; //Config compare Register, rising edge
12     OC4RS = 0x00638000; //Secondary compare Register, falling
           edge
13     OC4CON = 0x0006; //Turn on Output Compare in PWM mode
14     OC4RS = (PR4 + 1)*((float) dutyCycle / 100); //Sets the duty
           cycle, RS = time until falling edge starts
15     OC4CONSET = 0x8020; //Enable peripheral,  bit 5: 0=16 bit
           compare mode, 1=32 bit
16
17     OC5CON = 0x0000; //Same as above
18     OC5R = 0x00638000;
19     OC5RS = 0x00638000;
20     OC5CON = 0x0006;
21     OC5RS = (PR2 + 1)*((float) dutyCycle / 100);
22     OC5CONSET = 0x8020;
23
24     T2CONSET = 0x0008; //Starts a 32-bit timer
25     T2CONSET = 0x8000; //Enables the timer
26
27     PR4 = (sysClk / (pwmFreq * 2) * prescaleV) - 1; //Calculate
           how often the timer should trigger
28     PR2 = (sysClk / (pwmFreq * 2) * prescaleV) - 1;
29 }
```

### 5.2.1 Why utilize pulse-width modulation

FiXme Fatal: section needs readover

[2]

Pulse-width modulation, or PWM, is a way to regulate power distribution within a system. It is a software solution that manages when a device receives power, and for how long at a time it does this. This is called a duty cycle. The robot utilizes

[1] FiXme Fatal: virker lidt underligt? gentager intro direkte

[2] FiXme Fatal: section needs readover

PWM for its motors, to regulate how quickly it moves. PWM can be compared to turning a switch on and off extremely quickly - much more quickly than what will affect the performance of the motors. Effectively, this means that the robot's programming will now be able to regulate speed autonomously [3].

FiXme Fatal: autonomously?

### 5.2.2 Duty cycle

The duty cycle is used to describe how long the power is 'on' compared to 'off'. A higher duty cycle will yield more energy than a low one. The software uses a frequency of 1000Hz, which makes it straightforward to calculate to real time, if this is needed - it also provides enough precision to make the motors responsive quickly. The duty cycle can be changed by changing how long until the Output Compare sends a falling edge:

```
1  void adjustDuty(int channel, int duty) {//The function takes an
       argument based on which motors PWM should change, and the
       desired duty cycle
2      switch (channel) {
3          case 1:
4              OC5RS = (PR2 + 1)*((float) duty / 100); //Sets the
                   secondary register, to tell it how long until it
                   should send a falling edge
5              break;
6          case 2:
7              OC4RS = (PR4 + 1)*((float) duty / 100);
8              break;
9      }
10 }
```

## 5.3 Feedback loop

A feedback loop is a way of controlling how something, in this case a robot, behaves by receiving an output and adjusting the performance to match a desired output. The robot made in this project utilizes a feedback loop by firstly turning towards a set goal, and then avoiding obstacles along the way through sensors measuring the distance from the robot to the obstacle. If an obstacle gets too close, the robot will turn away from the obstacle and head back towards the goal.

### 5.3.1 Reading sensors

To know how far away an object is, some form of sensor feedback is needed. In this case, 3 ultrasound sensors have been implemented. The microprocessor sends a turn-on signal to the sensors, starts a timer and then waits until the sensors return with their own signal. The time between the start and finish signal can then be used to calculate the distance between the robot and the object.

[3] FiXme Fatal: autonomously?

```
long readUltrasonic(int channel){
    long timerFinish = 0;
    long timerOld = 0;
    int timeout = 3000; //Sets a limit for how long the MCU waits
        for data
    switch(channel){ //Switches between the 3 sensors
        case 1:
            Trigger1 = 1; //Pin RG6
            DelayUs(10); //Sends a short pulse
            Trigger1 = 0;
            while(Echo1 == 0){} //Waits until pin RF6 is set to
               high
            timerOld = micros(); //Sets the start timer to time
               since the program started, in microseconds
            while(Echo1 == 1){ //While the sensors signal is high
                timerFinish = micros(); //Sets the end timer to
                   check for timeout
                if(timerFinish - timerOld > timeout ||
                   timerFinish - timerOld < 0) //If the time to
                   return is too high or below 0, return a
                   default value
                    return 500;
            }
            break;
...
    }
    timerFinish = micros();
    long result = ((timerFinish-timerOld)*0.34)/2; //Calculate
       distance from speed of sound, divided by 2 since the sound
        wave also needs to return to the sensor.
    return result;
}
```

The short pulse sent in the beginning of the function has been set by the manufacturer: [4]

FiXme Fatal: reference til http://www.micropik.com/P

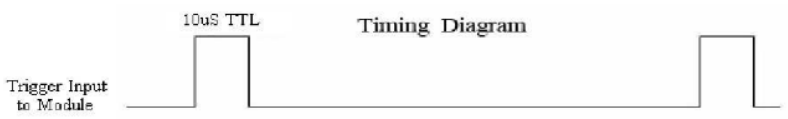



Figure 5.2: HC-SR04 timing diagram

To reduce false sensor data, which there seems to be a lot of with these sensors, a rolling average has been implemented. This calculates the average of the previous 5 readings instead of a single reading, allowing the robot to ignore spikes in data.

[4] FiXme Fatal: reference til http://www.micropik.com/PDF/HCSR04.pdf

### 5.3.2 Feedback

The controller function takes the data from the 3 sensors and uses these to control how the robot should behave. The function is by no means fully optimized, but allows the robot to avoid obstacles in it's way.

```
void controller(int midSensor, int rightSensor, int leftSensor){
    int midDistance = 100; //Sets the limit for the sensors, in
       millimetres
    int sideDistance = 50;
    if(midSensor < midDistance) //First check the middle sensor
    {
        brake(); //Stops the motors
        calculatePosition(); //Defunct, but should calculate
           where the robot is now in a coordinate system
        backwards(); //Starts backing away from the object
        int tachTarget = tach1-300; //Sets a tach target
        while(tach1 > tachTarget){} //Drives backwards until
           hitting the target
        brake(); //Brake again
        presetTurnLeft(); //Turn 30 degrees left
        brake();
        forwards(); //Start driving forwards again
    }
    else if(leftSensor < sideDistance)
    {
        int tachTarget = tach1-20; //Sets the target tach
        turnRight(); //Turn right
        while(tachTarget > tach1){} //Until target tach is hit
        brake(); //Brakes
    }
    else if(rightSensor < sideDistance) //Same as left, but
       reverse
    {
        int tachTarget = tach1+20;
        turnLeft();
        while(tachTarget < tach1){}
        brake();
    }
    else
        forwards(); //If no sensor is within the limit, just
           drive forwards
}
```

## 5.4 CPLD

Compared to a regular microprocessor like the PIC32MX320F128H that is programmed in C converted to machine code in a procedural manner:

```
1 if(this) {
2   then this
3 } else {
4   this
5 }
```

a CPLD is programmed with what is called Hardware description language or HDL for short. instead of writing condition statements like you do in C, you define logic blocks and describe what that logic block do with the inputs and outputs given. [3] This allows for programming small logic circuits that act like several 74-series logic chips put together, but a CPLD can also be configured to contain a small microcontroller what internally can interpret machine code.
This make a CPLD a very powerful device since it can be configured to anything that can be made with pure logic circuitry, given that the selected CPLD have the needed amount of internal configurable logic blocks for the implementation. [2]
To simplify the programming, the IDE used to program the CPLD used on the motor shield, allows for programming by drawing schematics, which the IDE then synthensize and translate into the format the can be uploaded to the CPLD.
The CPLD on the motor shield is configured to allow full individual control of each h-bridge by using 4 control lines from the PIC32 to the CPLD for each h-bridge:

- Enable
- Direction A
- Direction B
- PWM

Using these inputs without any logic control, allows for unwanted configurations of the h-bridge that will cause the power supply to be shortet to ground, this issues have been solved by programming the CPLD with a logic block that does not allow the configuration to occur, no matter the configuration of the input to the CPLD.
5.3 shows the logic configurations of one of the two logic blocks that have been programmed into the CPLD.

## 5.5 Part conclusion

FiXme Fatal: Software problemer og løsninger

[5] [6]

[7]

FiXme Fatal: Hvorfor ikke RTOS?

FiXme Fatal: hvilke krav har vi droppet?

[5] FiXme Fatal: Software problemer og løsninger
[6] FiXme Fatal: Hvorfor ikke RTOS?
[7] FiXme Fatal: hvilke krav har vi droppet?

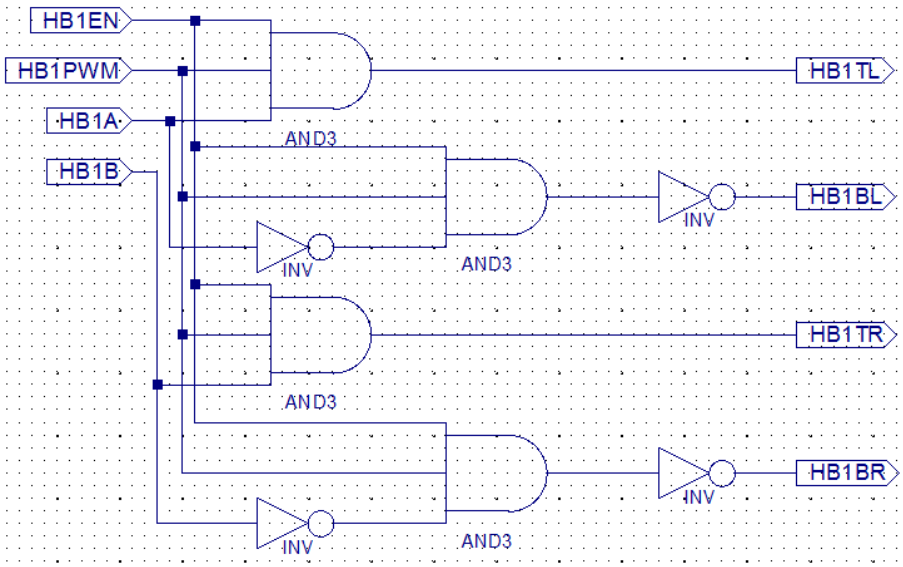



Figure 5.3: Single h-bridge logic block

# Test 6

In this section the different parts and signals will be tested, this is to make sure everything is up to par with our requirements specifications. The tests consists of measuring different factors, such as current, voltage and response time of each fundamental part. The testing will be done with the available measuring devices such as an oscilloscope, power supply or multimeter.

FiXme Fatal: ALT TEST SKAL TJEKKES IGENNEM [1]

## 6.1 Unit Testing

### 6.1.1 Ultrasound HC-SR04

#### Equipment

- Arduino UNO
- Agilent MSO-X 3024A Oscilloscope

#### Setup

A small program for the Arduino has been written which allows the MCU to trigger the sensor and waits a pre-specified amount of time before triggering again, to allow for the return of the ultrasound wave.

#### Results

**30mm:** 152 us
**50mm:** 285 us
**100mm:** 600 us
**200mm:** 1.23 ms
**500mm:** 2.32 ms

### 6.1.2 DC Motors

#### Equipment

- Hameg HM8040-2 Triple Power Supply
- Fluke 45 Multimeter

[1] FiXme Fatal: ALT TEST SKAL TJEKKES IGENNEM

**Setup**

**Results**

### 6.1.3 H-Bridge

**Equipment**

- Elcanic Power Supply
- Fluke 45 Multimeter

**Setup**

Power to the H-bridge is supplied by the power supply. Current and voltage will be measured by the multimeter. All of the Mosfet transistors were removed before doing the test. It was done to avoid powering the engines.

**Results**

There was a suspicion concerning the H-bridge, it had been measured that there was an error or a damaged component. The affected suspected part was the H-bridge for the left motor.

All of the transistors were tested on the part of the H-bridge in question. Since there was a 0 in value on the base, it is suspected that a resistor which sat on the board just before the transistor. This and the other resistor values were measured and found to be identical, based on the measures it could be concluded that it was most likely a failed transistor.

It was concluded that the damaged transistor was the: Q14 [1] [2].

FiXme Fatal: link til schematics

The results were as following:
Normal NPN transistors: base 0.854V, collector 0.073V
Faulty NPN transistor. base 0V, collector 0V.

[2] FiXme Fatal: link til schematics

### 6.1.4 PWM

**Equipment**

**Setup**

**Results**

## 6.2 Integration Testing

### 6.2.1 PWM motor control

**Equipment**

**Setup**

**Results**

### 6.2.2 Robot to Interface communication

**Equipment**

**Setup**

**Results**

## 6.3 System Testing

**Equipment**

**Setup**

**Results**

## 6.4 Acceptance Testing

**Equipment**

**Setup**

**Results**

# Conclusion 7

[1]

FiXme Fatal: Skriv en fucking Conclusion!!. FANDME!

[1] FiXme Fatal: Skriv en fucking Conclusion!!. FANDME!

# Appendices 8

FiXme Fatal: hele den her section [1]

## 8.1 Group collaboration agreement

### 8.1.1 Contact Information

Table 8.1: Contacts

| | | |
|---|---|---|
| Benjamin Nielsen | Tlf: 30427645 | @: yipiyuk5@gmail.com |
| Henrik Jensen | Tlf: 28568934 | @: henrik_kort@hotmail.com |
| Martin Nonboe | Tlf: 23827566 | @: nonsens_4@hotmail.com |
| Nikolaj Bilgrau | Tlf: 29802715 | @: nikolajbilgrau@gmail.com |

### 8.1.2 Workflow

### 8.1.3 Milestones and goals

#### Hardware

The hardware part of the project was supposed to be ready before the 22th

#### Software

Gerne en kalender der viser dage arbejdet!

[1] FiXme Fatal: hele den her section

# List of references 9

# List of Figures



**Page**

# List of Tables



**Page**

# Hardware appendix 10

## 10.1 Hardware Schematics

### 10.1.1 H-bridge 1

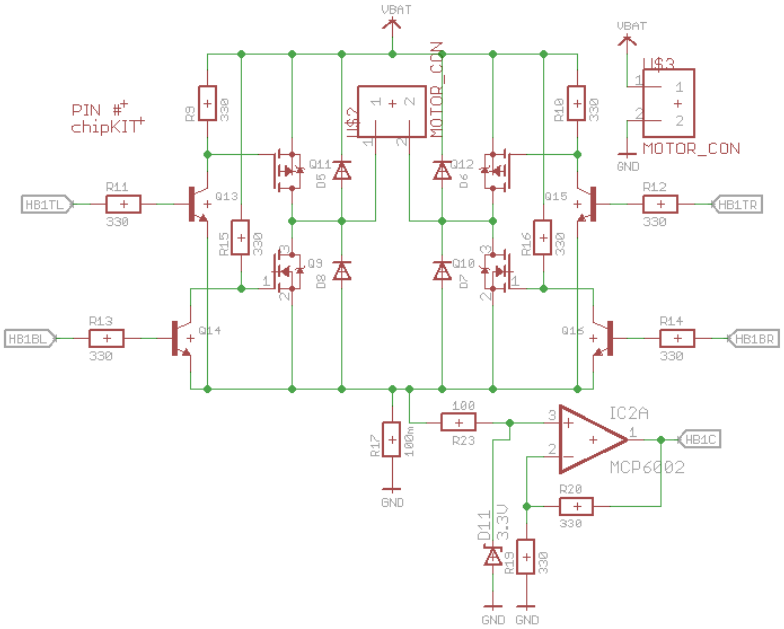



Figure 10.1: H-bridge 1 Schematics

### 10.1.2 H-bridge 2

### 10.1.3 CPLD

### 10.1.4 Power

### 10.1.5 Board schematics

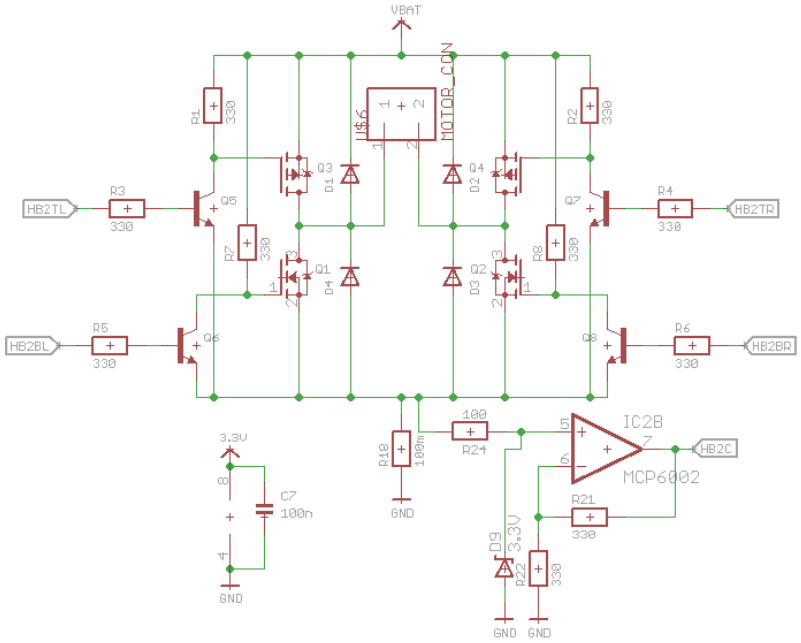



Figure 10.2: H-bridge 2 Schematics

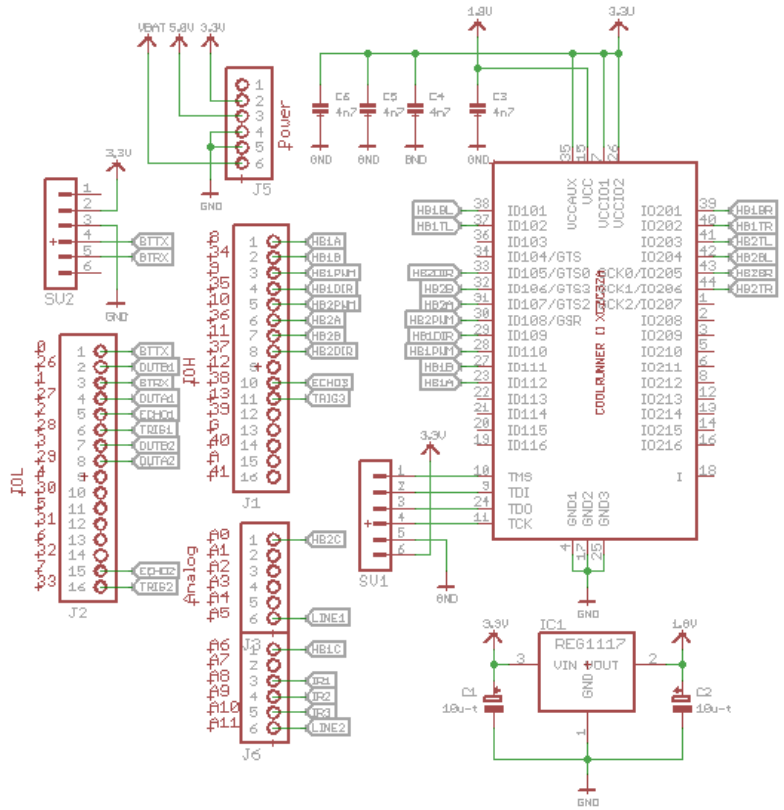



Figure 10.3: Schematics of the CPLD

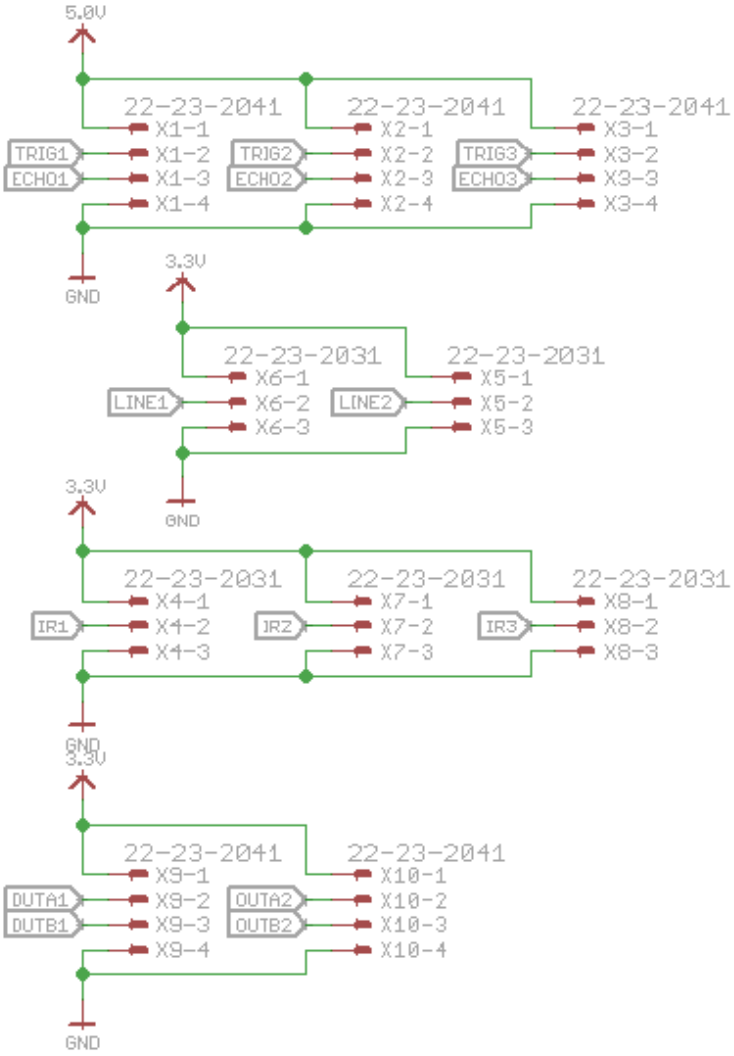



Figure 10.4: Power supply

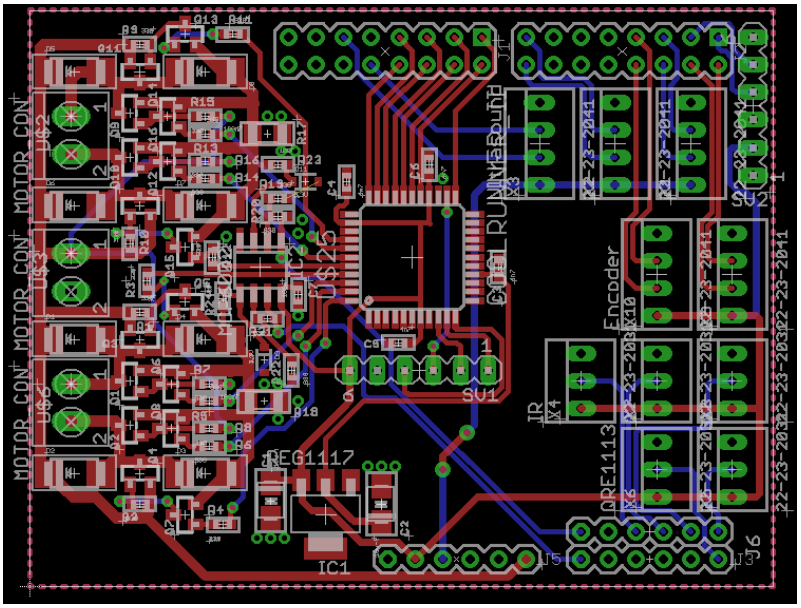



Figure 10.5: Full board Schematics

# Software appendix 11

## 11.1 C code

main.c:

ADC.c:

## 11.2 C# code - interface

# Bibliography

[1] UCN Group 2. "Third semester report". Chapter 10, board schematics. 2016.

[2] Wikipedia. *Complex programmable logic device*. 2016. URL: https://en.wikipedia.org/wiki/Complex_programmable_logic_device.

[3] Wikipedia. *Hardware description language*. 2016. URL: https://en.wikipedia.org/wiki/Hardware_description_language.

## Rettelser