Fróðskaparsetur Føroya

5025.16 KT-verkætlan (2018)

Arduino Project

Radar

Author
Helgi Poulsen

Author
Anna Thomsen

Supervisor
Benadikt Joensen

January 2, 2019

Contents

Abstract

Kunningartøkni, programmering og tól/dimsar er blivi lættari og lættari atkomuligt fyri forbrúkaran, also øll eiga onkran lítlan dims. Við alnótuni og microteldum, sum Arduino, kunnu øll gera teirra egnu dimsar.

Hetta projektið snýr seg um at nýta ein Arduino Uno og fleiri smærri komponentar, til at gera ein lítlan einklan radara. Hetta fyri at skilja tey grundleggjandi tingini ísmb. við programmering og at nýta tað til at gera okkurt funktionelt ting, td. ein dims.

Foreword

In the course 5025.16 KT-verkætlan (2018) we are required to make a project of our choosing, to learn how to create a project and write a report about it. This is done to prepare us to the bachelor project.

In this project we have included everything from building the radar, writing and understanding the code and showing how tests can be made from a ultrasonic sensor.

Doing this project helped us to understand and enhance our knowledge in starting a project, finding information about it and in the end write a report about it in LaTeX. Through this report we come to know about importance of team work and role of devotion towards the work.

Acknowledgement

In any project it is very important to have guidance. In this project we had Mr.Benadikt Joensen as our supervisor. We are very grateful for the help he provided us through this project from start to finish.

We also want to give all our classmates that helped us with various problems that we encountered.

We also want to give a special thanks to our classmates Fróði H. Joensen and Bergur I. Johansen for providing smart solutions when it came to the coding.

Chapter 1

Introduction

In this project we are going to work with an Arduino Uno microcomputer and a variety of components to create a small-scale radar. This to get a small understanding of the Arduino and the possibilities it comes with.

The Arduino programming comes with some number of pre-set libraries, but they haven't always got the most satisfying codes for the project. Therefore, one has sometimes to write new codes, which we will also see in this project.

From this point on *on* and off are going to be used instead of *start* and *stop*. This to indicate that the radar is being turned on and off, but the Arduino Uno itself is not being turned on and off.

Chapter 2

Problem description and statement

The object of this project is to get a small, but basic understanding of Arduino hardware and software, this to be able to create a small working piece of machinery. After some thinking and discussing this machinery is going to be a small radar, which the measurements will be shown on a small screen and also using LED's to indicate an object nearby. Throughout this project there is a learning curve incorporated, where the beginning is a rock bottom, we need to learn Arduino programming and how to utilize it. Thereafter it is possible to start to build the actual project or said in other words, we start with turning on a small LED and expand it into a small-scale radar. Even making some thoughts about how it can be upgraded in the future and possible improvements.

The radar is built in such a way, that it sends and receives signal in all directions, 360 degrees, anything else would be useless. However, when creating a small school project, it's a highly simplified version and it has its limitations. The radar has a view range of about 180 degrees and can only check one degree at the time and is only accurate on static objects.

Chapter 3

Radar

A radar, well what does it do and why we use it?

The challenge of seeing objects and estimating a distance to these objects, has been vastly minimized by the invention of the radar. Not only does the radar tell you that there is an object, but it also tells you in which direction and at what distance the object is. This has shown to be very use full onboard ships and aeroplanes, but also on land, at stations monitoring ship and air traffic, airports etc. While placed on land, the main purpose of the radar is to detect ship traffic and air traffic. But when a radar is placed on for example a ship, its main purpose is to detect other ships and/or objects within the sailing path, so that the ship doesn't collide, it prevents accidents.

The radar is the eyes and ears of the navigator. It has a 360-degree view and can detect moving objects as well as still standing objects. It gives you the direction, distance and speed of the object, no matter how big or small the object is, but of course depending on the type of radar and the use.

A radar is based on a radio transmitter and a receiver, therefore the name radar (Radio Detection And Ranging). It uses a very high frequency, much higher than any television or radio signal, this to prevent distortion from other radio signals. When in function it sends out small electromagnetic pulses, through an antenna, and if this pulse hits anything, the echo gets sent back for the receiver to pick up, very much like how a bat finds its way in the dark. There do exist many types of radar, one for each specific use. The angle of which the transmitter sends out the pulse, decides the use of the radar, for example does an airport radar point upwards, since it detects air traffic and a ship radar is used to detect objects at the same level as the ship.

3.1 Hardware components

- LED
- RGB LED
- Push button
- LCD Display
- Temperature sensor
- Servo Motor
- Ultrasonic sensor
- Resistors

3.2 Hardware components purpose and what are they being used for

3.2.1 LED

Figure 3.1: LEDS

A LED (light-emitting diode) is used like a small light bulb, but is in fact a semiconductor, since it only lets current flow in one direction, from the anode to the cathode. It is used for many purposes in today's society, especially for lighting, regular light bulbs and Christmas lights, but also for back light in televisions. When working with Arduino you can make this small LED to light up constantly or to flash, depending on what purpose it serves.

In this project the LED, well in fact there are three LED's (green, yellow and red), their purpose is to indicate whether the radar is on, off or loading, depending on the state of the push button.

3.2.2 RGB LED

Figure 3.2: RGB LED

A RGB LED is much like a regular LED, but the main difference being that the RGB LED can change colour, meaning that it consists of three LED's with the standard colours red, green and blue. The brightness of each LED decides the colour of the LED, much like mixing paint, but using light instead of paint.

The RGB LED used indicates at what distance the object detected by the Ultra Sonic is, so if the light of the RGB LED turns red the object is closer than 10 cm, if it turns green the object is between 10 cm to 100 cm and when it's blue the object is at a distance of a 100 cm or more.

3.2.3 Push button

Figure 3.3: Push button

A push button reassembles a light switch in our home. When the button is not pressed it does not let any current through it, known as state HIGH in the programming. When the button gets pressed, it lets current flow through it, much like when a light switch is on, the state is known as LOW.

The purpose of the button used is to turn the radar on and off, that means when the radar is off the red LED is lit and nothing moves. And when the button is pressed, the green LED is lit and the Radar is moving.

3.2.4 LCD display

Figure 3.4: LCD Display

The LCD display (liquid crystal display) is a very common thing, displays are used in everything from televisions to cell phones. The one for the Arduino is a very simple one, it doesn't come with colour, meaning the writing is white, and it is only capable of writing two lines, containing 16 signs in each line, it mostly resembles the old cell phones.

Regarding the project, when the radar is turned off the display reads "Radar project version 2.0". When turning the radar on the display clears and starts displaying the direction of the Servo Motor and Ultra Sonic. When the Ultra Sonic detects an object, the display shows at what distance the object is and at what angle from the radar. It also displays the temperature measured, by the temperature measurer, since the temperature has an effect on the speed of sound.

3.2.5 Temperature sensor

Figure 3.5: Temperature sensor

The temperature sensors job is to measure the temperature of the surroundings, since the speed of sound and temperature go hand in hand.

3.2.6 Servo motor

Figure 3.6: Servo motor

A servo motor is based on an Arduino library, which is included in the Arduino, the library is the used in the programming. On the hardware part, the servo motor has integrated gears and shaft, which allows it to be controlled precisely. The standard servo motor can be positioned at various angles between 0 to 180 degrees.

The purpose of the Servo Motor is to turn the Ultrasonic sensor in an arc of 180 degrees, from left to right and back. Its job is also to let the user and Arduino know at all times at what angle it's located. This information is used to tell at what angel an object is located.

3.2.7 Ultrasonic sensor

Figure 3.7: Ultrasonic sensor

The Ultrasonic sensor is based on sonar technology and can determine the distance of an object, with a range of $2cm$ to $400cm$. It doesn't get affected by sunlight and comes as a complete module, including the transmitter and receiver.

In this case the ultrasonic sensor is placed on the moving servo motor, so it can detect objects in a 180-degree radius, thereby being able to see what is in front, it is the key component of the radar. By detecting an object and at what distance, it gives us an idea of where and how close in front of us an object is located.

3.2.8 Resistors

Figure 3.8: Resistors

The resistors help control the current flowing to different components, meaning the limit the amount of current flowing to the components. This becomes much needed when working with highly current sensitive components, like the LED's, which can only withstand $2V$. However, it is also used in front of the push button to direct the current, to detect if the button is pressed down or not.

3.3 Complete circuit

On the picture it looks like everything is connected to each other and in a way it is, since the board itself is what brings everything together. But we can simplify it into this electrical drawing:

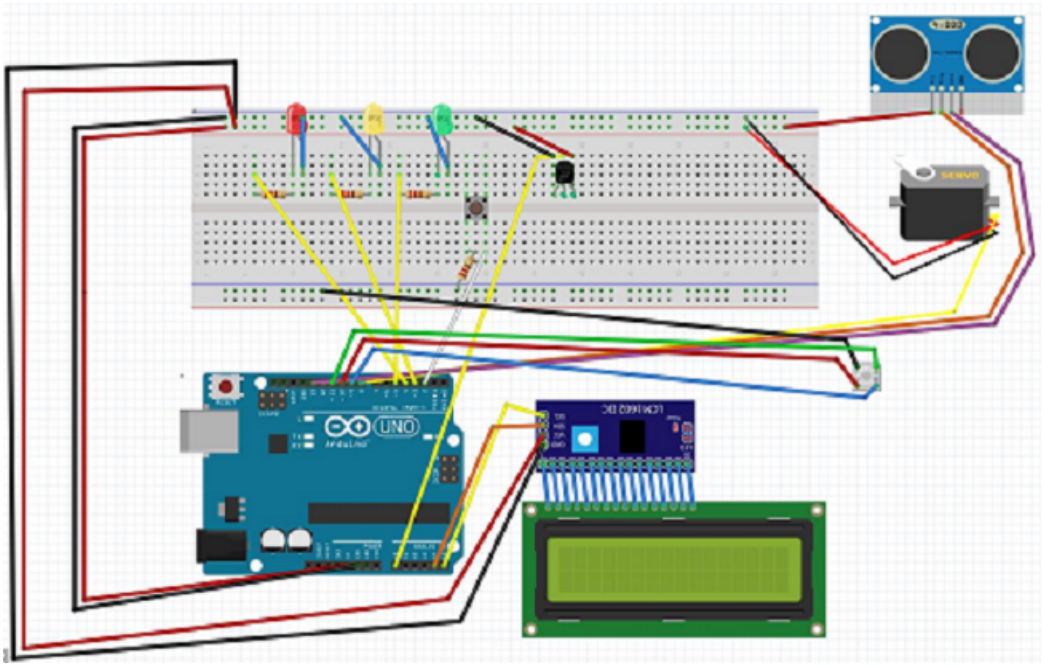


Figure 3.9: Arduino drawing

As we can see the circuit board has been left out, and the individual components are all connected directly to the Arduino.

OBS! The LCD display and the LCD controller are drawn as two components, but in this project they are one unit, so the non-important wires are grey, while the four wires connecting the LCD unit to the Arduino are the important ones.

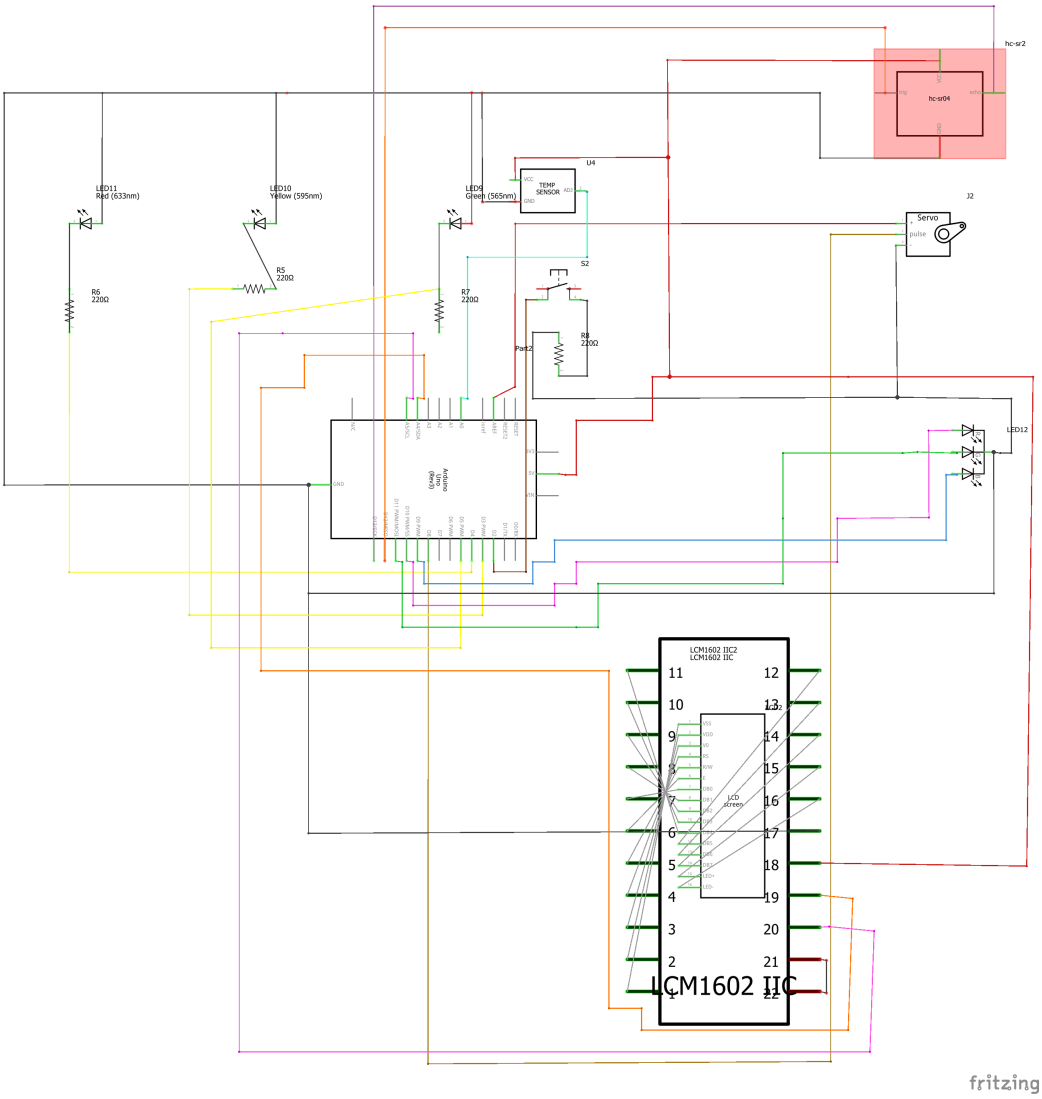


Figure 3.10: Circuit drawing

Chapter 4

How it's done

Describing the process from start to finish, and the challenges we met along.

The biggest challenge/uncertainty was to get the components in time. We had to order some components online, since the university did not have them in stock. We ordered the Ultra Sonic, LCD Display and RGB LED module form China.

So while waiting for these components, we decided to attach and program the other components, which we already had. Like the three LED's and belonging resistors, the servo motor and the button and to get them to function properly.

So to start with, we got the LED lighting up with the press of the button and to stay on until the button was pressed again. Adding a second LED, so when the button was pressed on, the green LED "lighted up" and when the button was off, the red LED "light" up. This button function was then applied to every component throughout the project, more on this later. We then added the Servo motor and got it running smoothly. It was then added to the button function, turning on and off with the LED's. By now the components started arriving from Chine, starting out with the LCD display. Getting this to work properly was a small challenge, especially when rewriting itself, it showed to be challenging to get it to clear itself at the right time. Shortly after the RGB LED and the Ultra Sonic where attached, here the most challenging problem being the Ultra Sonic. The challenge was to get the programming in place for the Ultra Sonic, so that the Servo motor, which it is attached to, still was running smoothly. The Servo motor kept waiting for the Ultra Sonic, since the Ultra Sonic did not receive a signal back and therefore stalled the entire radar and made it not run fluidly.

Chapter 5

Code

5.1 Arduino Enviroment

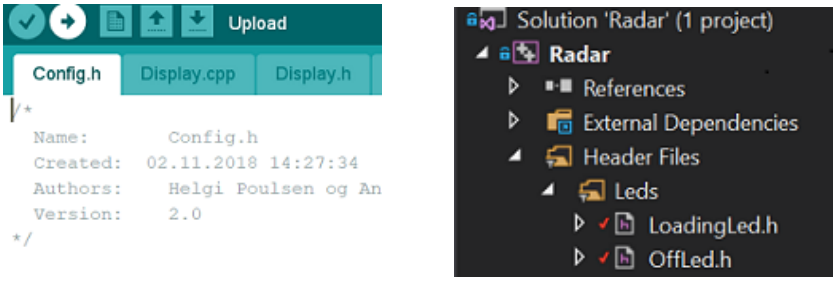
5.1.1 Arduino Coding Environment

To write code for Arduino devices, you need an IDE[1]. An IDE normally consists of a source code editor, build automation tools, and a debugger.

The most popular IDE is Arduino IDE, which is very beginner friendly, but it has it's weaknesses when it comes to writing code, for instance the lacking of *intelliSense*[2]. This is not an issue when making a small project, like making a LED blink. But when it comes to bigger projects, which have many components and 3rd party libraries, coding becomes very hard without intelliSense. Another disadvantage with Arduino IDE, is when working with many files. The more files the project has the more complicated it becomes to locate them, since it is all set up like a large document instead of small subsections. A small example of this 5.1

[1] Integrated development environment

[2] IntelliSense is a code-completion aid that includes a number of features: List Members, Parameter Info, Quick Info, and Complete Word.



(a) Arduino IDE (b) Visual Studio 2017

Figure 5.1: Difference in file structure

For these reasons and others, Visual Studio was used to create this project.

5.1.2 Arduino code

The Arduino language is merely a set of $C/C++$ functions that can be called from your code. All standard C and $C++$ constructs supported by $avr-g++$ should work in Arduino.[3]

Arduino uses sketches. A sketch is the name that Arduino uses for a program. It's the unit of code that is uploaded to and run on an Arduino board.

Every project must have one main sketch file, which must be an $.INO$ file. This is the file containing the *setup*() and *loop*() functions. The *setup*() runs only once, when the Arduino is powered up. After *setup*(), the *loop*() function runs over and over. This means that when } is reached, it starts from the top again.

$.CPP$ and $.H$ files can also be used and is recommended, since it gives you more control and overview.

[3] https://www.arduino.cc/en/Main/FAQ

5.2 Libraries

Libraries are files written in C or C++, which provide your sketches with extra functionality. More information about them can be found at https://www.arduino.cc/en/Main/Libraries, where even a template is provided so that the user can write his own library.

In this project we used a few libraries:

- Servo *Servo*
- NewPing *Ultrasonic*
- LiquidCrystal_I2C *Display*
- Wire *Display*

5.3 Radar.ino

This is the main file in the project. Here we have included all the header files and created objects that are needed, like the *servo* and *lcd* objects that come from the *servo* and *LiquidCrystal_I2C* libraries.

5.3.1 void setup()

```
1  void setup()
2  {
3    Serial.begin(9600);
4    //For printing on PLX-DAQ (Excel)
5    Serial.println("CLEARDATA");
6    Serial.println("LABEL,Time,Started Time,Test,Left/Right,Angle,
       Range,Temperature");
7    Serial.println("RESETTIMER");
8    //Servo------
9    servo.attach(DigitalPins::SERVO); //Attach servo with pin
10   servo.write(0); //Setting servo starting position
11   //Led--------
12   load.blink(); //Loading blinking lights
13   //Display----
14   lcd.init(); //Initialize display
15   lcd.begin(16, 2);   // iInit the LCD for 16 chars 2 lines
16   lcd.backlight(); //Turn on display
17   display.intro(lcd); //Showing intro on display
18 }
```

Serial.begin(9600)

Sets the data rate in *bitspersecond*(*baud*) for serial data transmission. For communicating with the computer. The rate set is 9600.

Serial.println()

The serial.println() functions are used for printing the data to the serial port. In this case it is used to get the data into a *Excel* spreadsheet via a software called $PLX - DAQ$[4] for the tests to measure the accuracy of the ultrasonic sensor. More information about $PLX - DAQ$ can be found on PLX-DAQ

Servo

We initialize the servo here by attaching it to a pin and setting it to position 0.

load.blink()

This is just making the yellow led blink fast multiple times. This just gives the user the sensation that everything is starting up.

lcd

here we initialize the lcd display and turn it on. *display.intro*(*lcd*) is the function that shows the intro screen.

5.3.2 void loop()

```
1 void loop()
2 {
3   powerOnOff.run(servo, lcd);
4 }
```

Here is where the magic happens. In our *loop*(), we only have one function, that controls everything.

run(*servo*, *lcd*) takes two arguments, the servo object and the lcd object.

5.4 Classes

We have created a class for each component(sensor,motor,led), since the main purpose of C++ programming is to add object orientation to the C programming language. This gives us much more flexibility with the project and makes it much easier to expand. Each class is also in its own header file. Our classes are:

- UltraSonicClass

[4]PLX-DAQ is a Parallax microcontroller data acquisition add-on tool for Microsoft Excel

- TemperatureClass
- ServoMotorClass
- SerialPrintClass
- PowerOnOffClass
- DisplayClass
- LedClass
  - LoadingLedClass
  - OnLedClass
  - OffLedClass
- RGBLedClass

We use hierarchical inheritance for the led classes. We wanted to add a factory method design pattern[5] to the led classes, but we encountered some strange object problems.

5.5 Config.h

The config header file contains the constants for the digital and analog pins. Here we use namespace[6]

5.6 Functions

This project contains lots of classes and functions. Here we will take a closer look at some of the more complicated and important ones.

5.6.1 void run(Servo s, LiquidCrystal_I2C lcd)B.2

As mentioned in 5.3.2, this is the main function.

voidrun(*Servos*, *LiquidCrystal_I2Clcd*) is the button. If turned on, turn the servo, measure temperature, measure distance and so on. If off, stop everything, turn the the red led on and display the into on the display.

[5] https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

[6] A namespace is a declarative region that provides a scope to the identifiers (names of the types, function, variables etc.) inside it.

Since the code is heavily commented, to make it as easy to understand as possible, the need to explain each line is not necessary.

To get the button to work as intended was more complicated that it seemed.

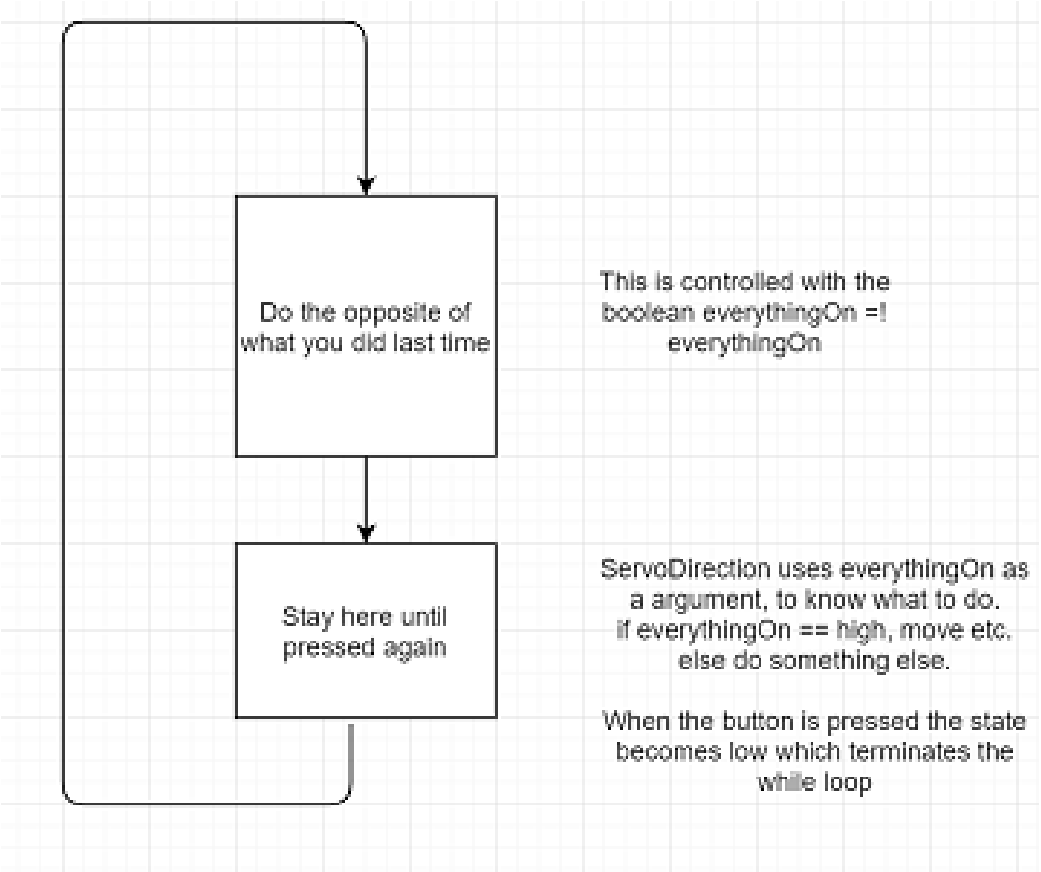


Figure 5.2: Button diagram

The basic principle of it5.2:

5.6.2 void servoDirection(Servo s, int state, LiquidCrystal_I2C lcd)B.2

In $voidservoDirection(Servos, intstate, LiquidCrystal\_I2Clcd)$. we check if the state is $HIGH$, then for every 10 counter degrees, we set the temperature and speed of sound. We do this since the temperature module is not stable enough. Then we set the distance in *centimeters* and then showing the distance with the rgb module and print the important information on the display.

We use a if statement with the argument *up*, that is a boolean, to decide which way the servo goes. If up is true then go from min to max, else go from max to min.

The *counter* is the servo position. So when *counter* reaches max, we set

up = $false$.

the string $leftRight$ and integer $test$ are only used for $serialPrint$B.9.

$sm.rotate$B.3 is the function that rotates the servo. It takes $servo$ and $counter$ as arguments.

5.6.3 void setSpeedOfSound(int temperature)B.4

To set the speed of sound with temperature, we need to use the speed of sound formula:

$$c = 331.3 \times \sqrt{\frac{\vartheta + 273.15}{273.15}} = 331.3 \times \sqrt{1 + \frac{\vartheta}{273.15}} \tag{5.1}$$

A simplified formula can be used for $35°C to -35°C$

Speed of sound in air in m/s:

$$c = 331.3 + 0.6 \times \vartheta \tag{5.2}$$

To make our calculations light weight as possible, we used the the simplified formula.

5.6.4 void setDistance(float speedOfSound)B.4

We found a very good library, $NewPing$, that can measure the distance for the ultrasonic sensor. We decided not to use it since it doesn't take speed of sound as argument and found it better for the project to write the code ourselves.

We show how to use it in the inline comments if the reader wants to use it instead of our code.

First we set the trigger pin to LOW and make a 4 microseconds delay to ensure a clean $HIGH$ pulse, then the sensor is triggered by a $HIGH$ pulse of 10 microseconds.

$duration$ is the time in microseconds from the sending of the ping to the reception of its echo off of an object.

Then we convert the time into cm

$$distance = \frac{duration \times \frac{speedOfSound}{1000}}{2} \tag{5.3}$$

5.6.5 void setTemperature()B.5

To turn the 10-bit analog reading into a temperature:

Formula that converts the number 0/1023 from Analog ADC input into $0 - 5000mV (= 5V)$:

First we read the analog pin, then:

$$voltage = reading \times 5.0 \tag{5.4}$$

then:

$$voltage = \frac{voltage}{1024} \tag{5.5}$$

Convert to Celsius:

$$temperatureCelsius = (voltage - 0.5) \times 100 \tag{5.6}$$

Chapter 6

Ultrasonic test results

6.1 Range test (No motion)

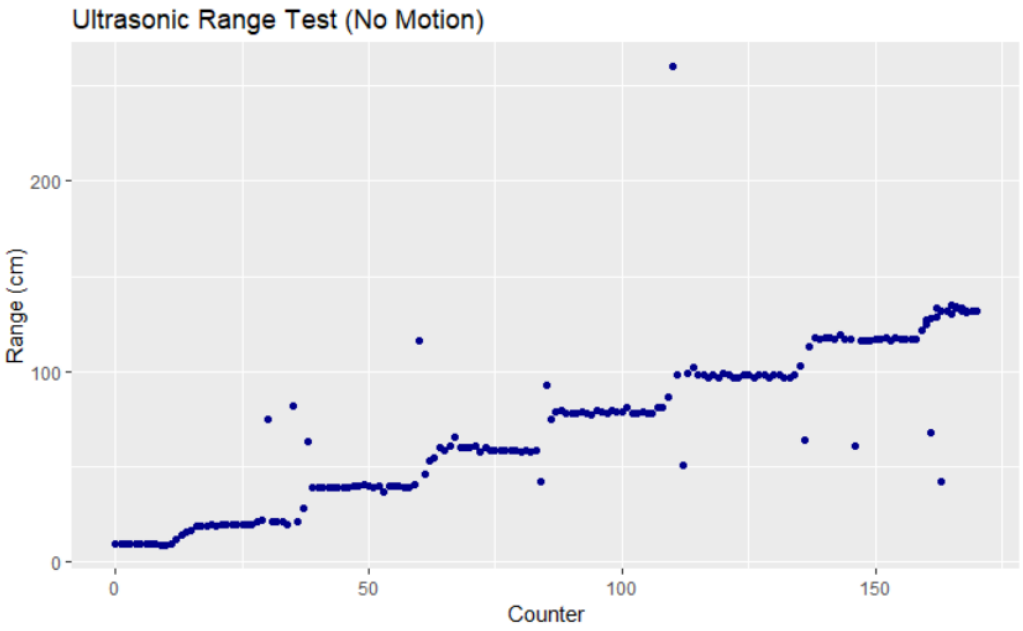


Figure 6.1: Ultrasonic range test (no motion)

We did a ultrasonic range test where we placed a object in front of the ultrasonic sensor and moved it back 6.1. The test result shows that we got some outliers, but overall the ultrasonic sensor was fairly accurate.

6.2 Ultrasonic range test (motion)

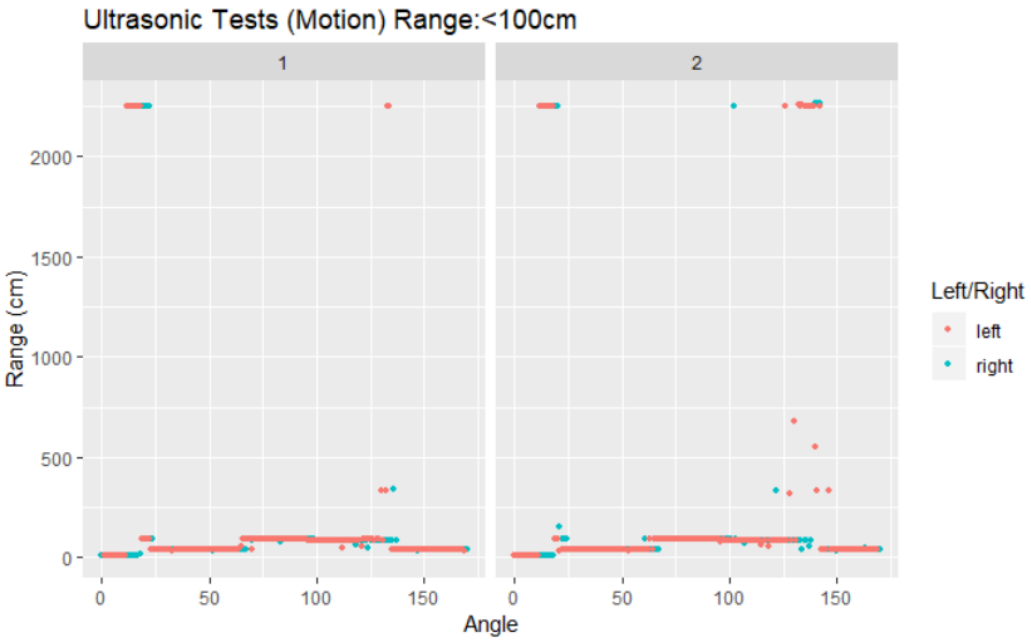


Figure 6.2: Ultrasonic range test (motion)

This test shows how accurate the sensor is when moving from 0 to 170 (right) then 170 to 0 (left) two times, $Test1$ and $Test2$. As seen on 6.2.

The test6.2 should show us again that the sensor is accurate enough for us to work with, even though we got some strange outliers, like in $Test2$ between Angle 130 and 150. The outliers that are $2500cm$ is when the sensor didn't hit an object.

6.3 Ultrasonic range test (motion) Range:<100 cm

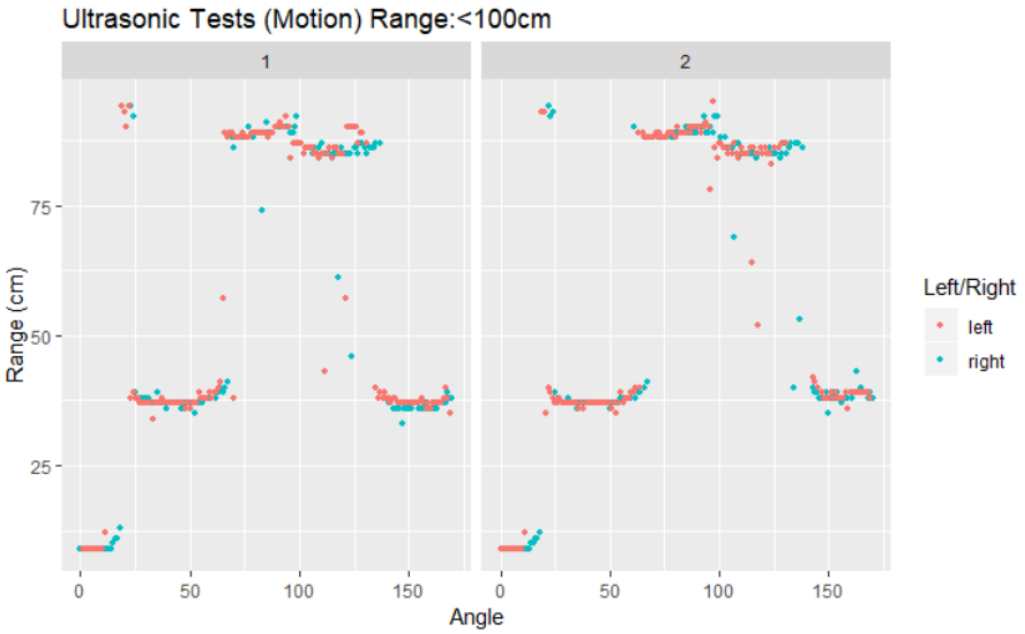
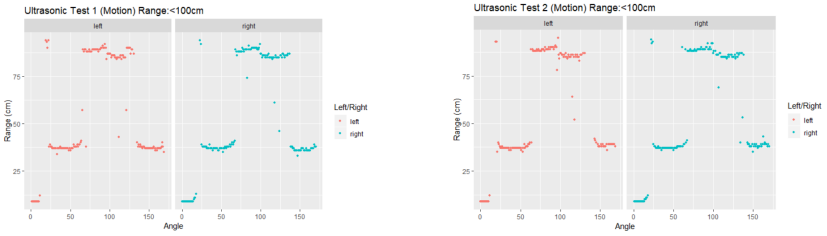


Figure 6.3: Ultrasonic range test (motion) Range:<100 cm

In 6.3 we only show everything that is below 100cm. This is to show a better image of the results.

6.4 Ultrasonic range tests separated (motion) Range:<100 cm



(a) Test 1 (b) Test 2

Figure 6.4: Ultrasonic range tests separated (motion) Range:<100 cm

Here everything is separated. If the ultrasonic sensor was 100% accurate, all the four images should look the same. As we can see to look quite similar, at least similar enough for us to be happy with the test results.

We can also point out that a reason for the test results not looking exactly the same where the temperature readings. Sometimes we also got strange temperature readings and since we took a reading every 10 degree, this could have a small effect on the results.

Chapter 7

The results

The complete radar is now serving its goal, but the one thing not working perfectly is the Ultrasonic sensor. There seems to be some kind of error in the code, so when the receiver doesn't receive an echo, the radar glitches for an instant, before continuing on its arc. We discovered that if we used the pre-set library, the glitch did not appear, but that library didn't take in account the temperature. Therefore, we decided to continue with our own code, instead of trying to change the library, also time was becoming an issue.

The test we made proved to be satisfying, even though there were some slight errors because of the fluctuations of the temperature sensor, these can be seen on the graphs shown earlier. Since the main principle of the radar is working and it's accuracy is precise enough for this project, the radar is found satisfying.

If we were going to continue working on the radar project, of course fixing the glitch would be our first priority, but adding an OLED display would be very beneficial. Sine the OLED display can give the user a graphical view of the measurements, meaning it looks more like the output you get from an actual radar. Also, it would be highly beneficial to design a surface or box, so that such things as wires, breadboard and Arduino Uno aren't exposed, hence it would look more like a small gadget that you could buy at a local store. It could potentially be a small toy for children, from the age of 10 or so.

Chapter 8

Conclusion

Looking back on the project we have been through a learning curve, a learning curve involving Arduino software and hardware, especially getting hardware and software working together has been a steep learning curve, since it has proven to be hard to get all the components working together and as intended, since the coding didn't always seem logical.

The majority of the hardware components are fairly simple, but when looking at the LCD display and the Ultrasonic sensor proved to be a challenge to understand their nature. Regarding the coding the challenge again showed itself regarding the same components as with the hardware, but here especially the programming of the push button proved to be a challenge, since it isn't logical or in other words it isn't a switch, but we want it to work as one.

The radar itself is now up and running, it doesn't work perfectly as a modern radar, since it's a school project and with the time given there is no chance to get it to the standards of a modern radar. But as a school project it is satisfying, and we are able to show the principle of a radar and we have gotten some insight in how Arduino can be used in the modern society

Appendices

Appendix A

Plots for tests

```
---
title: "Radar"
output:
  pdf_document: default
  html_document:
    df_print: paged
---

```{r setup, include=FALSE}
library(tidyverse)
library(readxl)
library(tinytex)
noMotion <- read_excel("test1noMotion.xlsx")
withMotion <- read_excel("Test2withMotion.xlsx")
```


Plots

```{r}

 ggplot(data = noMotion, mapping = aes(x = noMotion$Angle, y =
    noMotion$Range)) +
  geom_point(color ="darkblue")+
  labs(title = "Ultrasonic Range Test (No Motion)")+
  labs(x="Counter")+
  labs(y="Range (cm)")
```


```{r}

 ggplot(data = withMotion, mapping = aes(x = withMotion$Angle, y
```
```

```
    = withMotion$Range,
  color = withMotion$LeftRight)) +
  geom_point(size = 0.9)+
  facet_grid(cols = vars(withMotion$Test))+
  labs(title = "Ultrasonic Tests (Motion) Range:<100cm")+
  labs(x="Angle")+
  labs(y="Range (cm)")+
  labs(color="Left/Right")
```

```{r}
range100<-filter(withMotion, Range < 100)

ggplot(data = range100, mapping = aes(x = range100$Angle, y =
    range100$Range,
  color = range100$LeftRight)) +
  geom_point(size = 0.9)+
  facet_grid(cols = vars(range100$Test))+
  labs(title = "Ultrasonic Tests (Motion) Range:<100cm")+
  labs(x="Angle")+
  labs(y="Range (cm)")+
  labs(color="Left/Right")
```

```{r}
test1<-filter(withMotion, Range < 100 & Test == 1)


ggplot(data = test1, mapping = aes(x = test1$Angle, y = test1$
    Range,
  color = test1$LeftRight)) +
  geom_point(size = 0.9)+
  facet_grid(cols = vars(test1$LeftRight))+
  labs(title = "Ultrasonic Test 1 (Motion) Range:<100cm")+
  labs(x="Angle")+
  labs(y="Range (cm)")+
  labs(color="Left/Right")
```

```{r}
test2<-filter(withMotion, Range < 100 & Test == 2)

ggplot(data = test2, mapping = aes(x = test2$Angle, y = test2$
    Range,color = test2$LeftRight)) +
  geom_point(size = 0.9)+
```

```
  facet_grid(cols = vars(test2$LeftRight))+
  labs(title = "Ultrasonic Test 2 (Motion) Range:<100cm")+
  labs(x="Angle")+
  labs(y="Range (cm)")+
  labs(color="Left/Right")
```

Appendix B

Code

B.1 Radar.ino

```
/*
  Name:         Radar.ino
  Created:   02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

//Header files
//-----------------------------------
#include "SerialPrint.h"
#include <NewPing.h>
#include "Config.h"
#include "Display.h"
#include "PowerOnOff.h"
#include "LoadingLed.h"
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
//-----------------------------------

//Objects from libraries
//-----------------------------------
LiquidCrystal_I2C lcd(0x27, 2, 16);
Servo servo;
//-----------------------------------
LoadingLedClass load;
DisplayClass display;
PowerOnOffClass powerOnOff;
ServoMotorClass servoMotor;
//-----------------------------------

```

```
32 // The setup() function runs once each time the micro-controller
       starts
33 void setup()
34 {
35   Serial.begin(9600);
36   //For printing on PLX-DAQ (Excel)
37   Serial.println("CLEARDATA");
38   Serial.println("LABEL,Time,Started Time,Test,Left/Right,Angle,
       Range,Temperature");
39   Serial.println("RESETTIMER");
40   //Servo-----
41   servo.attach(DigitalPins::SERVO); //Attach servo with pin
42   servo.write(0); //Setting servo starting position
43   //Led-------
44   load.blink(); //Loading blinking lights
45   //Display---
46   lcd.init(); //Initialize display
47   lcd.begin(16, 2);    // iInit the LCD for 16 chars 2 lines
48   lcd.backlight(); //Turn on display
49   display.intro(lcd); //Showing intro on display
50 }
51
52 // Add the main program code into the continuous loop() function
53 void loop()
54 {
55   powerOnOff.run(servo, lcd);
56 }
```

B.2 PowerOnOff

```
/*
  Name:         PowerOnOff.h
  Created:   02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#ifndef _POWERONOFF_h
#define _POWERONOFF_h

#if defined(ARDUINO) && ARDUINO >= 100

#include "arduino.h"
#include "ServoMotor.h"
#include "Display.h"
#include "Temperature.h"
#include "UltraSonic.h"
#include "OffLed.h"
#include "OnLed.h"
#include "RGBLed.h"
#include "SerialPrint.h"

#else
#include "WProgram.h"
#endif

class PowerOnOffClass
{
private:
  //objects
  ServoMotorClass sm;
  DisplayClass display;
  TemperatureClass temperature;
  UltraSonicClass ultraSonic;
  OnLedClass onLed;
  OffLedClass offLed;
  RGBLedClass rgb;
  SerialPrintClass serialPrint;

  //For serialPrint
  //--------------------
  int test = 1;
  String leftRight = "right";
  //--------------------

  int state = LOW;
  bool everythingOn = true;
```

```
48   int counter = 0;
49
50   int tmp;
51   const int max = 170;
52   const int min = 0;
53   bool up = true;
54
55 public:
56   PowerOnOffClass();
57   void run(Servo s, LiquidCrystal_I2C lcd);
58   void servoDirection(Servo s, int state, LiquidCrystal_I2C lcd);
59 };
60 extern PowerOnOffClass PowerOnOff;
61
62 #endif
```

```
1  /*
2    Name:         PowerOnOff.cpp
3    Created:   02.11.2018 14:27:34
4    Authors:      Helgi Poulsen og Anna Thomsen
5    Version:      2.0
6  */
7
8  #include "PowerOnOff.h"
9
10 PowerOnOffClass::PowerOnOffClass()
11 {
12   pinMode(DigitalPins::POWERONOFF, INPUT_PULLUP); //Decleare the
       PowerOnOff button as a INPUT_PULLUP
13   tmp = max;
14 }
15
16 void PowerOnOffClass::run(Servo s, LiquidCrystal_I2C lcd)
17 {
18   state = digitalRead(DigitalPins::POWERONOFF);
19
20   if (state == HIGH)
21   {
22     everythingOn = !everythingOn; // setting bool to the opposite
       value
23     //Setting states on leds
24     onLed.setState(everythingOn);
25     offLed.setState(!everythingOn);
26
27     //Writing the states of the leds
28     digitalWrite(DigitalPins::ONLED, onLed.getState());
29     digitalWrite(DigitalPins::OFFLED, offLed.getState());
30
31     //Displaying intro screen
```

```
32      display.intro(lcd);
33
34      //loop running as long as state = HIGH
35      while (state)
36      {
37        servoDirection(s, everythingOn, lcd); // moving the servo
38        state = digitalRead(DigitalPins::POWERONOFF); // getting
      state  of button
39      }
40    }
41  }
42
43  void PowerOnOffClass::servoDirection(Servo s, int state,
      LiquidCrystal_I2C lcd)
44  {
45    if (state == HIGH)
46    {
47      delay(100); //For making the servo move a little bit slower
48
49      //Take temperature every 10 degrees and setting the speed of
      sound
50      if (counter % 10 == 0)
51      {
52        temperature.setTemperature();
53        ultraSonic.setSpeedOfSound(temperature.getTemperature());
54      }
55      //Setting distance
56      ultraSonic.setDistance(ultraSonic.getSpeedOfSound());
57      //displaying distance with rgb led
58      rgb.state(ultraSonic.getDistance());
59      //Showing info on display
60      display.degree(lcd, counter, ultraSonic.getDistance(),
      temperature.getTemperature());
61
62      //Getting data to excel
63      //--------------------------------
64      serialPrint.print(test, leftRight, counter, ultraSonic.
      getDistance(), temperature.getTemperature());
65      //--------------------------------
66
67      //Servo going up from 0 to 170 until reaching max
68      if (up)
69      {
70        leftRight = "right";
71        if (counter == max)
72        {
73          up = false;
74        }
75        sm.rotate(s, counter);
```

```
76      }
77      //Servo going down from 170 to 0 until reaching minimum
78      else
79      {
80        leftRight = "left";
81        if (counter == min)
82        {
83          up = true;
84          ++test;
85        }
86        sm.rotate(s, counter);
87      }
88      //Counter going up or down depending on boolean up
89      (up) ? ++counter : --counter;
90    }
91  }
92
93  PowerOnOffClass PowerOnOff;
```

B.3 ServoMotor

```
/*
  Name:        ServoMotor.h
  Created:  02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#ifndef _SERVOMOTOR_h
#define _SERVOMOTOR_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"

#include <Servo.h>

#else
#include "WProgram.h"
#endif

class ServoMotorClass
{
public:
  void rotate(Servo pServo, int counter);
};
extern ServoMotorClass ServoMotor;

#endif
```

```
/*
  Name:        ServoMotor.cpp
  Created:  02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#include "ServoMotor.h"


void ServoMotorClass::rotate(Servo pServo, int counter)
{
  //Rotating the servo
  pServo.write(counter);
}

ServoMotorClass ServoMotor;
```

B.4 UltraSonic

```
/*
  Name:         UltraSonic.h
  Created:   02.11.2018 14:27:34
  Author:       Helgi og Anna
  Version:      1.5
*/

#ifndef _ULTRASONIC_h
#define _ULTRASONIC_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#include "Config.h"
#include <NewPing.h>
#else
#include "WProgram.h"
#endif

class UltraSonicClass
{
private:
  unsigned long duration;
  int distance;
  const float speedofsound0C = 331.3;
  const float constant = 0.6;
  float speedOfSound;

public:
  UltraSonicClass();
  //Setters
  void setSpeedOfSound(int temperature);
  void setDistance(float speedOfSound);
  //Getters
  float getSpeedOfSound() { return speedOfSound; }
  int getDistance() { return distance; }
};
extern UltraSonicClass UltraSonic;

#endif
```

```
/*
  Name:        UltraSonic.cpp
  Created:  02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#include "UltraSonic.h"


UltraSonicClass::UltraSonicClass()
{
  pinMode(DigitalPins::TRIGGER, OUTPUT); // Sets the triggerPin
    as an Output
  pinMode(DigitalPins::ECHO, INPUT); // Sets the echoPin as an
    Input
}

void UltraSonicClass::setSpeedOfSound(int temperature)
{

  speedOfSound = speedofsound0C + (constant * temperature);
}

void UltraSonicClass::setDistance(float speedOfSound)
{
  //Library-----------------------------------
  //NewPing sonar(12, 13, 200);
  //distance = sonar.ping_cm();
  //------------------------------------------

  digitalWrite(DigitalPins::TRIGGER, LOW);
  delayMicroseconds(4);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(DigitalPins::TRIGGER, HIGH);
  delayMicroseconds(10);
  digitalWrite(DigitalPins::TRIGGER, LOW);
  // Reads the echoPin, returns the sound wave travel time in
    microseconds
  duration = pulseIn(DigitalPins::ECHO, HIGH);
  // Calculating the distance cm
  float tmp = speedOfSound / 10000;
  distance = duration * tmp / 2;
}

UltraSonicClass UltraSonic;
```

B.5 Temperature

```
/*
  Name:         Temperature.h
  Created:   02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#ifndef _TEMPERATURE_h
#define _TEMPERATURE_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#include "Config.h"
#else
#include "WProgram.h"
#endif

class TemperatureClass
{
private:
  float temperatureCelsius;
  float voltage;
  int reading;

public:
  int getTemperature() { return temperatureCelsius; }
  void setTemperature();
};
extern TemperatureClass Temperature;

#endif
```

```
/*
  Name:         Temperature.cpp
  Created:   02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#include "Temperature.h"


void TemperatureClass::setTemperature()
{
  reading = analogRead(AnalogPins::TEMPERATURE);
  voltage = reading * 5.0;
  voltage /= 1024.0;


  temperatureCelsius = (voltage - 0.5) * 100;  //converting from
    10 mv per degree wit 500 mV offset
  //to degrees ((voltage - 500mV) times 100)
}

TemperatureClass Temperature;
```

B.6 Display

```
/*
  Name:         Display.h
  Created:   02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#ifndef _DISPLAY_h
#define _DISPLAY_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#include "Temperature.h"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "string.h"
#else
#include "WProgram.h"
#endif
//LiquidCrystal_I2C lcd(0x27, 2, 16);

class DisplayClass
{
private:
  TemperatureClass temperature;

public:
  DisplayClass();
  void intro(LiquidCrystal_I2C lcd);
  void degree(LiquidCrystal_I2C lcd, int counter, int distance,
    int temperature);
};
extern DisplayClass Display;

#endif
```

```
/*
  Name:        Display.cpp
  Created:  02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#include "Display.h"

DisplayClass::DisplayClass()
{}

void DisplayClass::intro(LiquidCrystal_I2C lcd)
{
  lcd.setCursor(0, 0); //First line
  lcd.print("RADAR PROJECT   ");
  lcd.setCursor(0, 1); //Second line
  lcd.print("Version 2.0     ");
}

void DisplayClass::degree(LiquidCrystal_I2C lcd, int counter, int
    distance, int temperature)
{
  if (counter == 9 || counter == 99)
  {
    lcd.clear();
  }

  //Casting it to string
  String tmpRange = "Range:" + (String)distance + "cm" + "  ";

  if (distance > 200) {
    //out of range
    tmpRange = "Range:" + String((char)176) + "    ";
  }
  lcd.setCursor(0, 0); //First line
  lcd.print(String("Angle:") + String(counter) + String((char)
   223) + String("  "));
  lcd.setCursor(11, 0); //First line
  lcd.print(String("C:") + String(temperature) + String((char)
   223));
  lcd.setCursor(0, 1); //Second line
  lcd.print(tmpRange);
}

DisplayClass Display;
```

B.7 RGBLed

```
/*
  Name:         RGBLed.h
  Created:   02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#ifndef _RGBLED_h
#define _RGBLED_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#include "Config.h"
#else
#include "WProgram.h"
#endif

class RGBLedClass
{
public:
  RGBLedClass();
  void state(int distance);
};
extern RGBLedClass RGBLed;

#endif
```

```
/*
  Name:        RGBLed.cpp
  Created:  02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#include "RGBLed.h"

RGBLedClass::RGBLedClass()
{
  pinMode(DigitalPins::RGBGreen, OUTPUT);
  pinMode(DigitalPins::RGBRed, OUTPUT);
  pinMode(DigitalPins::RGBBlue, OUTPUT);
}

void RGBLedClass::state(int distance)
{
  //0-255 brightness

  //Red
  if (distance < 10) {
    //off
    analogWrite(DigitalPins::RGBGreen, 0);
    analogWrite(DigitalPins::RGBBlue, 0);
    //on
    analogWrite(DigitalPins::RGBRed, 255);
  }
  //Green
  if (distance > 10 && distance < 100) {
    //off
    analogWrite(DigitalPins::RGBRed, 0);
    analogWrite(DigitalPins::RGBGreen, 0);
    //on
    analogWrite(DigitalPins::RGBBlue, 255);
  }
  //Blue
  if (distance > 100) {
    //off
    analogWrite(DigitalPins::RGBBlue, 0);
    analogWrite(DigitalPins::RGBRed, 0);
    //on
    analogWrite(DigitalPins::RGBGreen, 255);
  }
}

RGBLedClass RGBLed;
```

B.8 Leds

```
/*
  Name:         Led.h
  Created:   02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#ifndef _LED_h
#define _LED_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#include "Config.h"

#else
#include "WProgram.h"
#endif

class LedClass
{

protected:
  int state;

public:
  LedClass() {}

  virtual void setState(int pState) = 0;
  virtual int getState() const { return 0; }
  virtual void blink() = 0;
};

#endif
```

```
/*
  Name:         Led.cpp
  Created:   02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#include "Led.h"
```

```
/*
  Name:        OnLed.h
  Created:  02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#ifndef _ONLED_h
#define _ONLED_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#include "Config.h"
#include "Led.h"
#else
#include "WProgram.h"
#endif

class OnLedClass : public LedClass
{
public:
  OnLedClass();
  void setState(int pState) override;
  int getState() const override;
  void blink() override;
};
extern OnLedClass OnLed;

#endif
```

```
/*
  Name:        OnLed.cpp
  Created:  02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#include "OnLed.h"


OnLedClass::OnLedClass()
{
  pinMode(DigitalPins::ONLED, OUTPUT);
}

void OnLedClass::setState(int pState)
{
  state = pState;
}
```

```
20
21 int OnLedClass::getState() const
22 {
23   return state;
24 }
25
26 void OnLedClass::blink()
27 {}
28
29 OnLedClass OnLed;
```

```
1  /*
2    Name:         OffLed.h
3    Created:   02.11.2018 14:27:34
4    Authors:      Helgi Poulsen og Anna Thomsen
5    Version:      2.0
6  */
7
8  #ifndef _OFFLED_h
9  #define _OFFLED_h
10
11 #if defined(ARDUINO) && ARDUINO >= 100
12 #include "arduino.h"
13 #include "Led.h"
14 #include "Config.h"
15 #else
16 #include "WProgram.h"
17 #endif
18
19 class OffLedClass : public LedClass
20 {
21 public:
22   OffLedClass();
23   void setState(int pState) override;
24   int getState() const override;
25   void blink() override;
26 };
27 extern OffLedClass OffLed;
28
29 #endif
```

```
/*
  Name:         OffLed.cpp
  Created:   02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#include "OffLed.h"


OffLedClass::OffLedClass()
{
  pinMode(DigitalPins::OFFLED, OUTPUT);
}

void OffLedClass::setState(int pState)
{
  state = pState;
}

int OffLedClass::getState() const
{
  return state;
}

void OffLedClass::blink()
{}

OffLedClass OffLed;
```

```
/*
  Name:         LoadingLed.h
  Created:   02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#ifndef _LOADINGLED_h
#define _LOADINGLED_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#include "Led.h"
#include "Config.h"
#else
#include "WProgram.h"
#endif

class LoadingLedClass : public LedClass
```

```
{
public:
  LoadingLedClass();
  void setState(int pState) override;
  int getState() const override;
  void blink() override;
};
extern LoadingLedClass LoadingLed;

#endif
```

```
/*
  Name:         LoadingLed.cpp
  Created:  02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#include "LoadingLed.h"

LoadingLedClass::LoadingLedClass()
{
  pinMode(DigitalPins::LOADINGLED, OUTPUT);
}

void LoadingLedClass::setState(int pState)
{}

int LoadingLedClass::getState() const
{
  return 0;
}

void LoadingLedClass::blink()
{
  for (int i = 0; i < 20; i++)
  {
    delay(30);
    digitalWrite(3, HIGH);
    delay(30);
    digitalWrite(3, LOW);
  }
}

LoadingLedClass LoadingLed;
```

B.9 SerialPrint

```
/*
  Name:         SerialPrint.h
  Created:   02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#ifndef _SERIALPRINT_h
#define _SERIALPRINT_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#else
#include "WProgram.h"
#endif

class SerialPrintClass
{
public:
  void print(int pTest, String pLeftRight, int pCounter, int
    pDistance, int pTemperature);
};
extern SerialPrintClass SerialPrint;

#endif
```

```
/*
  Name:         SerialPrint.cpp
  Created:   02.11.2018 14:27:34
  Authors:      Helgi Poulsen og Anna Thomsen
  Version:      2.0
*/

#include "SerialPrint.h"

void SerialPrintClass::print(int pTest, String pLeftRight, int
    pCounter, int pDistance, int pTemperature)
{
  Serial.print("DATA,TIME,TIMER,");
  Serial.print(pTest);
  Serial.print(",");
  Serial.print(pLeftRight);
  Serial.print(",");
  Serial.print(pCounter);
  Serial.print(",");
  Serial.print(pDistance);
  Serial.print(",");
  Serial.println(pTemperature);
}

SerialPrintClass SerialPrint;
```

B.10 Config

```
/*
  Name:         Config.h
  Created:   02.11.2018 14:27:34
  Authors:     Helgi Poulsen og Anna Thomsen
  Version:     2.0
*/

#ifndef _CONFIG_h
#define _CONFIG_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "arduino.h"
#else
#include "WProgram.h"
#endif

namespace DigitalPins
{
  //Main button
  const int POWERONOFF = 2;
  //on off loading leds
  const int LOADINGLED = 3;
  const int OFFLED = 4;
  const int ONLED = 5;
  //Servo engine
  const int SERVO = 8;
  //RGB Module
  const int RGBGreen = 9;
  const int RGBRed = 10;
  const int RGBBlue = 11;
  //Ultrasonic
  const int TRIGGER = 12;
  const int ECHO = 13;
}

namespace AnalogPins
{
  //Temperature sensor
  const int TEMPERATURE = 0;
}

#endif
```