TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Henry Juhanson 178190

DEVELOPMENT OF HANDS-ON EXERCISES FOR EMBEDDED SYSTEMS

Master's thesis

Supervisor: Uljana Reinsalu
PhD

Co-supervisor: Mairo Leier
PhD

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Henry Juhanson

SARDSÜSTEEMIDE LABORITE VÄLJA TÖÖTAMINE

Magistritöö

Juhendaja: Uljana Reinsalu
PhD

Kaasjuhendaja: Mairo Leier
PhD

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Henry Juhanson

06.05.2019

Abstract

This thesis covers development of new hands-on exercises for an embedded systems course. The thesis covers why a new set of exercises were necessary to be developed, how similar courses are handled and what was done to develop the new tasks.

For new exercises the given idea – a robot would navigate in an area to specified locations, find NFC tag, read data at those locations and present it to a local server. This idea elaborated and expanded upon so that it could be divided more into suitable parts for students to develop – ranging from blinking lights to implementing an operating system. Chosen components and how they are put together to form a whole system to navigate, detect tags in the specified area, read them, communicate with a local server for further information is described.

This thesis is written in English and is 65 pages long, including 4 chapters, 64 figures and 3 tables.

Annotatsioon

SARDSÜSTEEMIDE LABORITE VÄLJA TÖÖTAMINE

Antud lõputöö on kirjutatud uute sardsüsteemide kursuse jaoks laborite välja töötamisest. Lõputöös on kirjeldatud miks oli tarvis uusi labored, kuidas muud sarnased laborid on struktureeritud ja mis on tehtud uute laborite välja arendamiseks.

Uute laborite etteantud algne idee oli luua robot mis suudaks navigeerida etteantud alal, liikuda spetsifitseeritud asukohta, lugeda seal andmed ja saata edasi kohalikule serverile. See idee sai edasi edasi täpsustatud ja edasi arendatud, et loodu oleks võimalik jagada sobivateks osadeks, mida tudengid saaksid täita. Ülesanded on tulede vilgutamisest kuni operatsiooni süsteemi implementeerimiseni. Valitud komponendid ja kuidas need on ühendatud, et luua süsteem mis oleks võimeline alal navigeerima, lugema infot ja suhtlema serveriga on kirjeldatud.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 65 leheküljel, 4 peatükki, 64 joonist, 3 tabelit.

List of abbreviations and terms

95HF	NFC reader/transceiver chip
ADC	analog-to-digital converter
CMSIS-RTOS	cortex microcontroller software interface standard
D/C	mode select
DAC	digital-to-analog converter
DC	direct current
DMA	direct memory access
FreeRTOS	open source RTOS
GitHub	a code hosting site with version control
GND	ground
GPIO	general purpose input-output
HAL	hardware abstraction layer
HC-SR04	ultrasonic sensor
I2C	inter-integrated circuit
IMU	inertial measurement unit
IoT	internet of things
IP	internet protocol
ISR	interrupt service routine
LCD	liquid-crystal display
LDO	low-dropout regulator
LED	light emitting diode
ms	millisecond
NDEF	NFC data exchange format
NFC	near-field communication
NVIC	nested vectored interrupt controller
PC	personal computer
PID	proportional-integral-derivative
PCD	proximity coupling device
PWM	pulse-width modulation
RTOS	real time
RX	receive
SCE	chip enable
SCL	serial clock line (for I2c)

SCLK	serial clock line (for LCD)
SDA	serial data line (for I2c)
SDIN	serial data line (for LCD)
SPI	serial peripheral interface
STM	STMicroelectronics
STM32CubeMX	initialization code generator for STM microcontrollers
TalTech	Tallinn University of Technology
TCB	task control block
TCP	transmission control protocol
TX	transfer
VCC	voltage common collector
WiFi	radio frequency technology used for wireless communication
µs	microsecond

Table of contents

1 Introduction 14
1.1 Previous course 14
1.1.1 Problems to solve 15
1.2 Other materials 15
1.2.1 Non-courses 16
1.3 Teaching method 16
1.3.1 Sequential versus concurrent teaching 17
1.3.2 Project-based learning 18
2 New course 19
2.1 Concept 19
2.2 Components 22
2.2.1 STM microcontroller 24
2.2.2 Sensors 25
2.2.3 Peripherals 26
2.2.4 Other hardware 26
2.2.5 Software 27
2.3 Exercises' order 29
3 Development 32
3.1 Introductory 32
3.1.1 Configuring in STM32CubeMX 34
3.1.2 Program 35
3.2 Driving motor 35
3.2.1 Configuring in STM32CubeMX 37
3.2.2 Program 38
3.3 Custom delay 39
3.3.1 Configuring in STM32CubeMX 39
3.3.2 Program 40
3.4 Ultrasonic sensor 41
3.4.1 Configuring in STM32CubeMX 43

3.4.2 Program 45
3.5 IMU 48
3.5.1 Hardware connections 48
3.5.2 I2C 49
3.5.3 Configuring in STM32CubeMX 50
3.5.4 Program 50
3.6 LCD, speaker, encoder 51
3.6.1 Transmission 52
3.6.2 Speaker 52
3.6.3 Encoder 53
3.6.4 Hardware setup 54
3.6.5 Configuring in STM32CubeMX 56
3.6.6 LCD program 57
3.6.7 Speaker program 58
3.6.8 Encoder 59
3.7 NFC 59
3.7.1 Hardware connection 60
3.7.2 Configuring in STM32CubeMX 61
3.7.3 Program 62
3.8 RTOS 62
3.8.1 Threads 63
3.8.2 Semaphores 65
3.8.3 Queues 65
3.8.4 Configuring in STM32CubeMX 66
3.8.5 Delay code 68
3.8.6 Semaphore code 68
3.8.7 Queue code 69
3.9 WiFi 70
3.9.1 Hardware connection 70
3.9.2 Configuring in STM32CubeMX 71
3.9.3 Program 71
3.10 Workflow 72
3.11 Server 73
3.12 Result 76

3.12.1 Project 77
4 Summary 79
5 References 80
Appendix 1 – NFC tags 82

List of figures

Figure 1. Comparisons of average results for studies 17
Figure 2. Basic concept of robot interfacing 20
Figure 3. Basic communication UML 21
Figure 4. Physical concept 22
Figure 5. STM32CubeMX example 24
Figure 6. HAL example 24
Figure 7. Software setup 28
Figure 8. Server software setup 29
Figure 9. Lab order 30
Figure 10. Connection diagram 31
Figure 11. Board top layout 33
Figure 12. External LED connection 33
Figure 13. STM32 microcontroller configuration in CubeMX for LEDs 34
Figure 14. Step 1 execution code 35
Figure 15. DC motor principle 36
Figure 16. DRV8833 H-bridge 36
Figure 17. Motor connection 37
Figure 18. DC motor PWM timer 38
Figure 19. Setting PWM value 38
Figure 20. Starting PWM timer 38
Figure 21. Non-blocking vs blocking delay 39
Figure 22. Delay timer 39
Figure 23. Delay example 40
Figure 24. Move function with delay 41
Figure 25. General ultrasonic measurement concept 42
Figure 26. Take measurement block 42
Figure 27. Ultrasonic connection 43
Figure 28. STM32 microcontroller configuration in CubeMX for ultrasonic module 44
Figure 29. Ultrasonic timer settings 45
Figure 30. Ultrasonic measurement 46
Figure 31. Basic navigation diagram 47

Figure 32. IMU connection 49
Figure 33. I2C timing diagram 50
Figure 34. I2C HAL function call 50
Figure 35. Simple PID 51
Figure 36. Transmission of one byte 52
Figure 37. DAC sine wave example 53
Figure 38 LCD connection 54
Figure 39 speaker connection 55
Figure 40 Motor connection 55
Figure 41. STM32 microcontroller configuration in CubeMX for LCD, DAC, Encoder pins 56
Figure 42 Timer 6 parameters 57
Figure 43 DAC OUT1 settings 57
Figure 44. LCD write byte 58
Figure 45. Speaker code 58
Figure 46. Tag detection concept 60
Figure 47 NFC connection 61
Figure 48. SPI parameters 61
Figure 49. Non threaded example 64
Figure 50. Thread example 64
Figure 51. Binary semaphore with interrupt 65
Figure 52. Insert to queue 66
Figure 53. Take from queue 66
Figure 54. RTOS memory management 67
Figure 55. Thread with osDelay 68
Figure 56. Semaphore example 69
Figure 57. Semaphore and osDelay 69
Figure 58. Sending to LCD queue 70
Figure 59. WiFi connection 71
Figure 60.Wifi workflow 72
Figure 61. Robot workflow 73
Figure 62. Server function scheme 75
Figure 63. Server call and answer example 76
Figure 64. Lab robot 76

List of tables

Table 1. Component list 23
Table 2. Database structure with example information 74
Table 3. Course lab timeline 78

1 Introduction

Embedded systems are all around in the modern world – electric scooters, smart watches, bus validation systems. Everything that runs on an integrated circuit is essentially an embedded system. Internet of things devices are embedded systems. It is highly beneficial to know how these systems work – how different components of a system communicate, how they are connected and what the sensors do. An embedded systems course introduces and explains how all these aspects work.

The problem to solve with this thesis was to create new hands-on exercises to be used by students in an embedded systems course. The previous exercises would be made completely deprecated with the use of the new exercises thus all new tasks had to be developed from scratch – this includes the full development cycle from an idea, to research, development and testing.

The base idea was presented by the department of computer systems of TalTech (Tallinn University of Technology). The presented idea was to create a system that would be able to be gradually developed by students during the course. The system was to be a robot that would be able to navigate in a specified area and communicate with a local access point for instructions to receive information where to go.

1.1 Previous course

The previous practical part of the course used to teach the students is a modified version of "Embedded Systems - Shape The World" by Jonathan Valvano and Ramesh Yerraballi. [1]. The course is built on using the TM4C123 Launchpad by Texas Instruments. Students have to be able to complete tasks starting from implementing switches and turning on LED (light emitting diode) lights, generating interrupts, using an ADC (analog-to-digital converter), DAC (digital-to-analog converter), displaying information on an LCD (liquid-crystal display), implementing an RTOS (real time operating system) and using WiFi (radio frequency technology used for wireless communication) to create a complete IoT (internet of things) device.

1.1.1 Problems to solve

The first problem - even though the previous exercises cover many essential areas to introduce students to embedded systems, the way they were presented to the students currently was not linear. The exercises were presented all as standalone problems to be solved. This meant that the even though some exercise share the same base the problems do no intertwine – when completing an exercise, the built solution is disassembled and starting a new exercise requires building it from scratch.

The second problem is that students don't want to do these exercises either because they are structured in such a way that no real result will be achieved at the end of the course. Based on this a third problem arises – if students want to create something more whole, grading them becomes more and more complicated. And if all students were to create their own embedded system too much workload would be put on the determining on how the student created it, what it composes of, did they do it by themselves.

A fourth problem would be that the used exercises are based on an online course, which has been around for years and many of the solutions can be easily found on the internet on public code hosting sites such as GitHub (a code hosting site with version control).

1.2 Other materials

There are embedded courses that are accessible online also such as embedded systems courses by ETH Zurich (Swiss Federal Institute of Technology in Zurich) and Berkeley university [2] [3]. The ETH Zurich course shows only that they cover some theoretical parts and how an RTOS works, and little is done on to interface different components together.

The Berkeley course covers more hardware but does it so by having students' program and communicate with different ready-made hardware pieces such as the WiiMote and a study robot called Cal Klimber [2], [3].

Another online example can be found from the Embedded Related website [4]. The tutorial covers basics like bit manipulation, using GPIO (general purpose input-output) pins, timers, interrupts, buttons, and displays as stand-alone exercises.

These courses are all built up in different ways – this shows that embedded systems can and are taught in various ways. None of these courses however are built up in a way that would eliminate all the problems of the currently used materials in TalTech for Embedded Systems course.

1.2.1 Non-courses

In addition to learning materials about embedded systems by following courses there are extensive material about embedded systems such as "Mastering STM32" by Carmine Noviello and "Discovering the STM32 Microcontroller" by Geoffrey Brown. Both books cover using STMicroelectronics' microcontroller. These books are both good reference materials for developing on these controllers but overall as they go into depth about how specifics work. They cover all the necessary points on how to handle developing an embedded system overall. The "Mastering STM32" provides also externally hosted examples to make learning easier [5], [6].

The books can be used as reference material for the developed course but not as the whole base of the course as they have some of the same problems as describe before – nothing whole is developed.

1.3 Teaching method

All courses and cited books have a hands-on approach for teaching. This is good as courses with hands on exercises have been proven to be highly beneficial. For example, in the article "Hands-on Engineering: Learning by Doing in the Integrated Teaching and Learning Program", where students were introduced to more of a hands-on approach to teaching was successful, as students became interested in taking more initiative and learning more by doing [7].

Even a little bit more active teaching method allow the result to better by a significant amount. An example Figure 1 by Carl E. Wieman shows improved test result indicating that even a small amount of activity in teaching can improve test results[8].

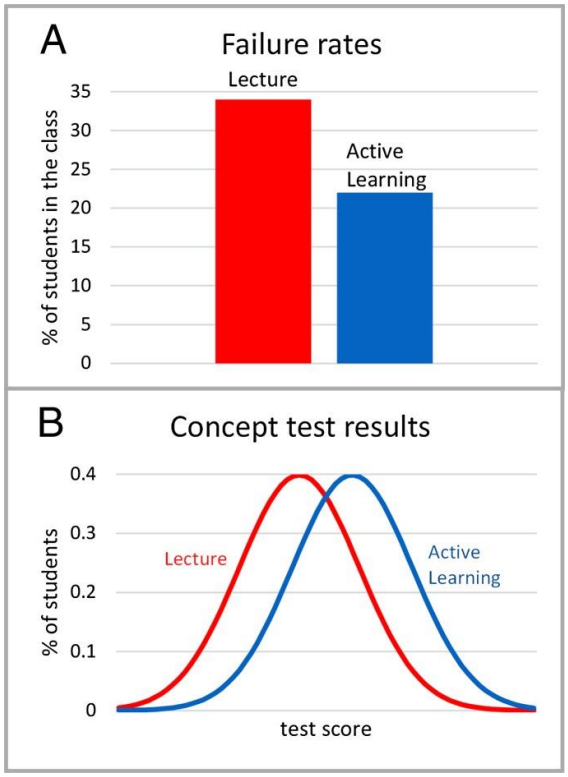

Figure 1. Comparisons of average results for studies [8]

1.3.1 Sequential versus concurrent teaching

Sequential teaching is the traditional way of teaching – the curriculum is divided in to strict blocks, which cover the material more in-depth. Content is divided in to blocks presented sequentially, following some logical structure. This way of teaching is used more for lecturers and non-active teaching methods, to introduce the student to the topics. Sequential teaching requires a lot from the teacher. The teacher has to present the material in such a way that the students understand and remember it. With sequential teaching all the material is presented when teaching and the students are tested only on the based materials [9].

Concurrent teaching – often seen as problem-based teaching, follows more of a learning by doing principle. The students are given a concrete engineering problem to solve, trying to mimic the workflow that of a real engineer. Students are expected to take the lead and the teacher is to act as an aid if necessary. In concurrent teaching the aspect of play can be adopted. Trying and testing how things work, finding an optimal solution. If all students try to figure out their own solutions the play aspect can be expanded up by the teacher, for example by holding a contest to see whose solution functions the best [9].

To further expand upon concurrent teaching, project-based learning can be used.

1.3.2 Project-based learning

Project-based learning is concurrent learning specified more in depth and over a longer time period. The play idea is focused on and students work in groups to achieve a desired goal.

Research in [10], shows a relation between students' motivational orientation and cognitive engagement in learning. It presents that a project requires two important components – a problem to be solved and drive activities. The problem allows all organization activities to revolved around it, and the drive activities result in products leading up to the final solution. The problem definition and drive can be defined by both the student and the teacher. The constraints however, when defining them especially by the teacher must be loose, so that the students have room to come up with and design their own solutions [10].

Reaching a solution should be designed to take a more long-term period. Getting students engaged in a whole semester long project can be difficult. Taking in to account also that students have all different knowledge levels and may know more or less about any given subject. To help alleviate this problem the students should be first given enough knowledge and some skills to explore further information on their own. For this reason, the course starts off with basics such as blinking LEDs and printing out messages.
To make sure that during the development of the system the students' knowledge gap is closed some instructional information and a steppingstone as mentioned before is provided [10].
Research has shown that students who participate in solving real life problems grade higher than those who don't. Students who took part in the research for [11] showed better results on content knowledge tests, better assessments of conceptual understanding and problem-solving abilities. It is also noted that project-based methods for teaching can help even those students that struggle in regular learning environments. Similarly, research for [12] shows that students retrain learned information better and have a good conceptual knowledge when taking part in project-based learning [11], [12].

2 New course

As stated in the previous paragraph project-based learning has shown good result, which is why a project was chosen as the desired outcome for new embedded system labs. With a project the problems of the previously used exercises get alleviated. The new course is designed to try and cover as many different aspects to embedded systems as possible, ranging from hardware connections, including different sensors, multiple communication protocols, basics of writing software for embedded systems, and using an RTOS. The general idea what should be accomplished by the end is defined. The in-between however from start to finish is more of a steppingstone so that the students can get started on coming up with their own solutions.

2.1 Concept

The result of the hands-on exercises for students is to achieve a system that is capable of driving around in a 2x3m area and has 2 walls. Basic view of the system is depicted on Figure 2 – A central microcontroller will be connected to sensors and motors. Using sensors to get information from the environment and using motors to move around. Power is provided by an external power source.

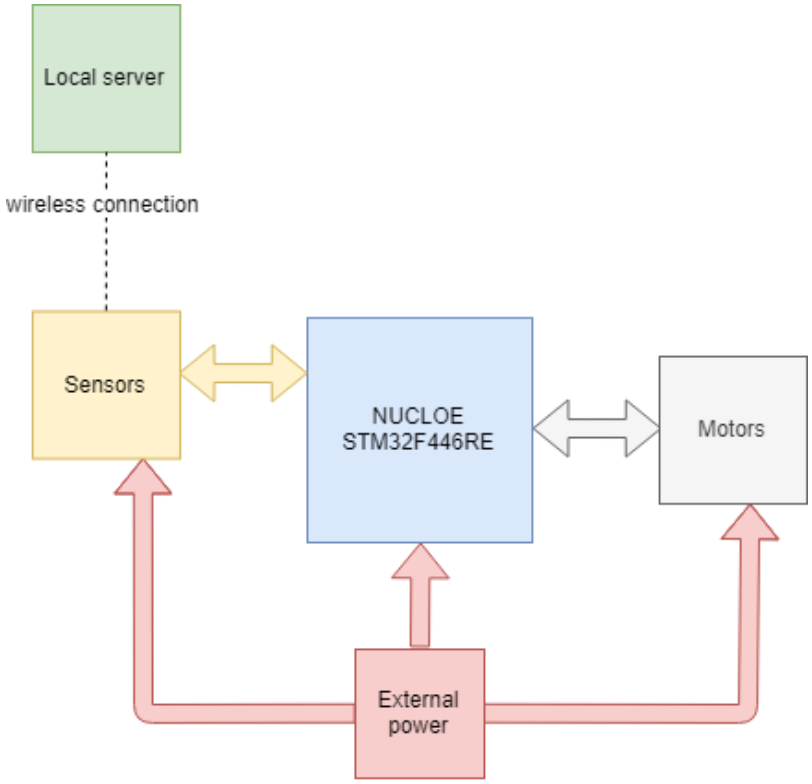

Figure 2. Basic concept of robot interfacing

The robot will start in a specified position – in a corner opposite to the walls. (Figure 4) The robot has to ask a local server for a tag location, to which it must then drive, read the tag information, report the tag information to the server and receive a new location if the read information was correct. Having several teams of students competition could be arranged: which robot registers all 5 tags within the minimum amount of time. The robot has to do this until all 5 tags have been registered as correct. Communication with the local server will be done using WiFi, UML diagram of communication is given at Figure 3. Physical concept scheme can be seen from Figure 4. The figure shows that the robot should communicate over WiFi with a local server to receive information about where to look for some tags on the field. The server will give the tag locations one by one and the robot has to find the tags, read them and report the read data to the server. Location of the next tag will be given only when the server receives correct information form the tag.

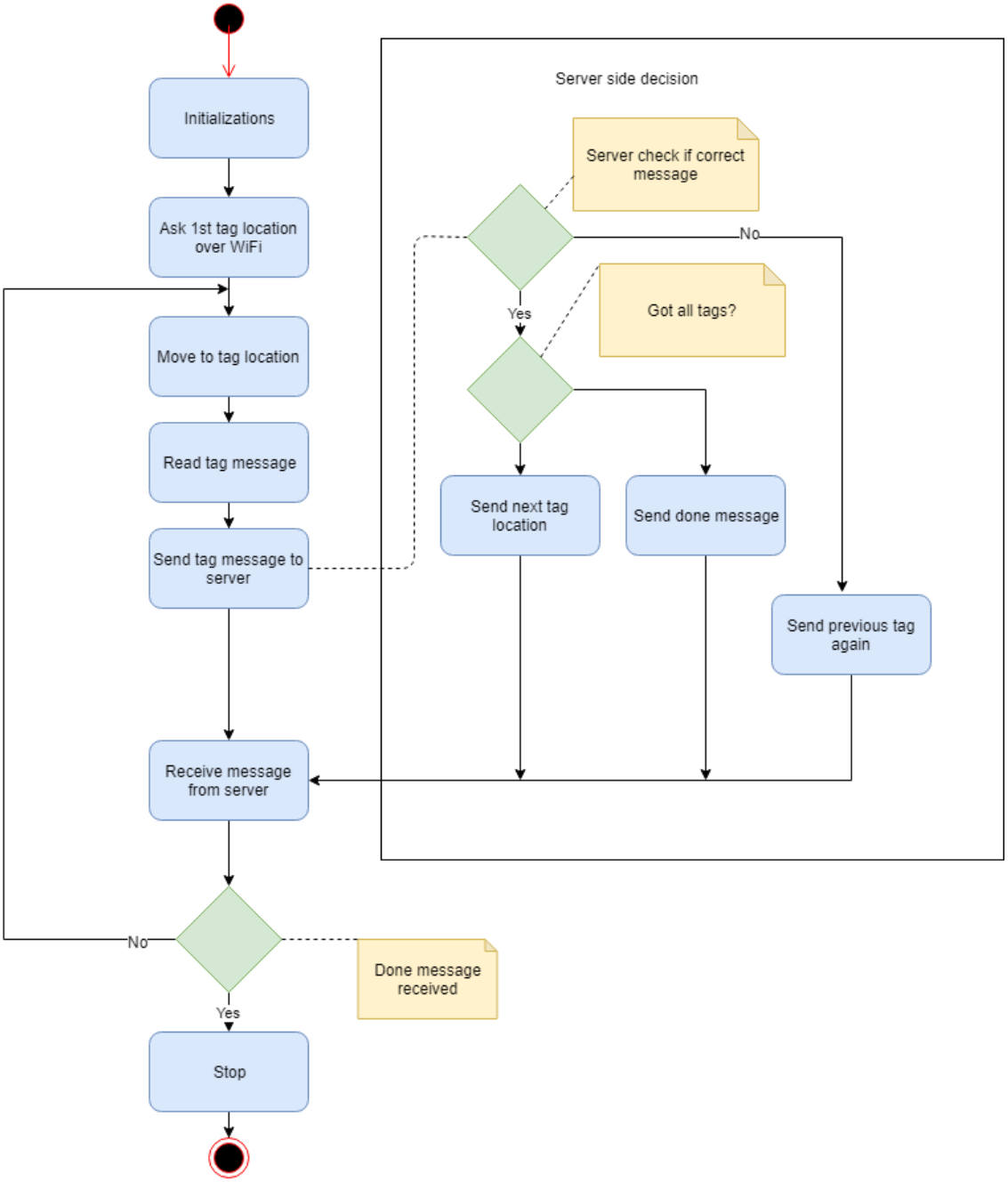

Figure 3. Basic communication UML

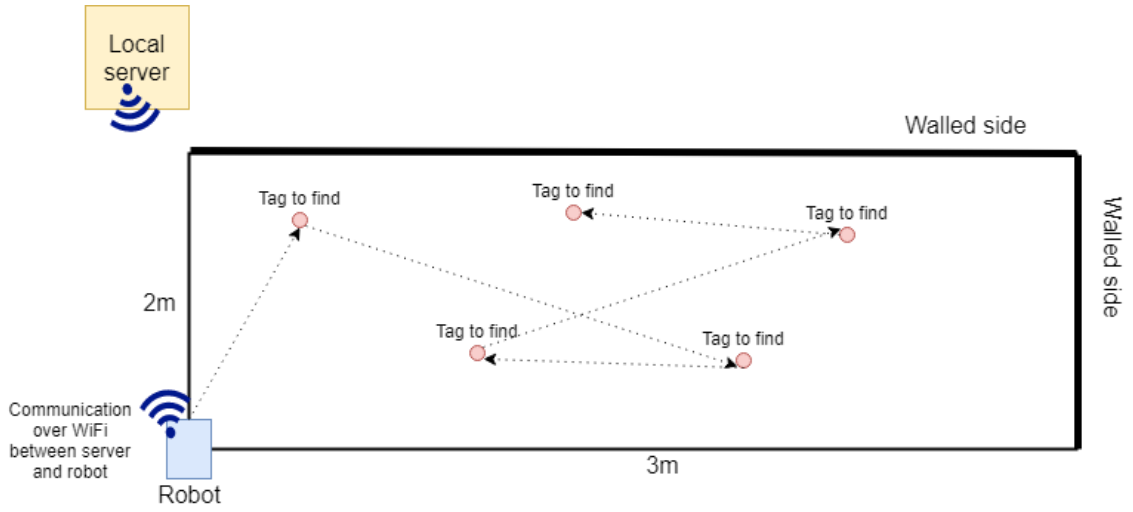

Figure 4. Physical concept

2.2 Components

One of the requirements for given system was it should run on an STM (STMicroelectronics) microcontroller. The requirement comes from the fact that the Tallinn University of Developments IoT Development center uses mostly these microcontrollers in their projects. In additional requirements an IMU (inertial measurement unit), an NFC (near-field communication) tag reader and a WiFi module to be added to the system. The rest of the components were chosen to diversify as much as possible the sensors and their communication protocols. The full list of components can be seen from Table 1.

Table 1. Component list

Component	Description
NUCLEO-F446RE	Microcontroller
X-NUCLEO-NFC03A1	NFC reader to read tags
Ultrasonic module HC-SR04 (ultrasonic sensor)	Ultrasonic distance measurement module
IMU BNO055	IMU with 9-axis of freedom
WiFi module ESP 8266	WiFi module for communication
Piezo Speaker MCABS-227-RC	Speaker for sound output using DAC
Nokia 5110/3310 monochrome LCD	LCD display
Pololu DRV8833 Dual Motor Controller	Motor controller driver, with H-bridge
Romi Chassis Encoder Pair Kit	Encoder for motor feedback
Romi Chassis Kit	Chassis and motor
Parallax 180 servo and mounting bracket	Servo and bracket for moving ultrasonic sensor left and right
Breadboard 400 holes	Breadboard to connect components with
LDO 5V	For bringing voltage level to 5V
LDO 3V3	For bringing voltage level to 3V3
Capacitors	Reducing noise where necessary
Resistor	Reducing current where necessary
Wires	For connecting components
Batteries	External power source

2.2.1 STM microcontroller

The STM microcontroller compared to for example the previously used TM4C123 is very similar both in price and regarding the functionality. As of 30.04.2019 NUCLEO-F446RE costs 12,71€ and TM4C123 12,97€ from Mouser.

Using a STM microcontroller however simplifies writing code. This is done by using the functionality of STM32CubeMX (initialization code generator for STM microcontrollers). STM32CubeMX allows to graphically initialize desired GPIO pins, communication protocols and even RTOS. As an example, from Figure 5. initialization of a timer can be seen, alongside with used GPIO pins on the right side. This way of initialization can help to speed up the process of development, by having the tool to generate all the necessary code based on set parameters. Manually initializing peripherals is still available through code if necessary.

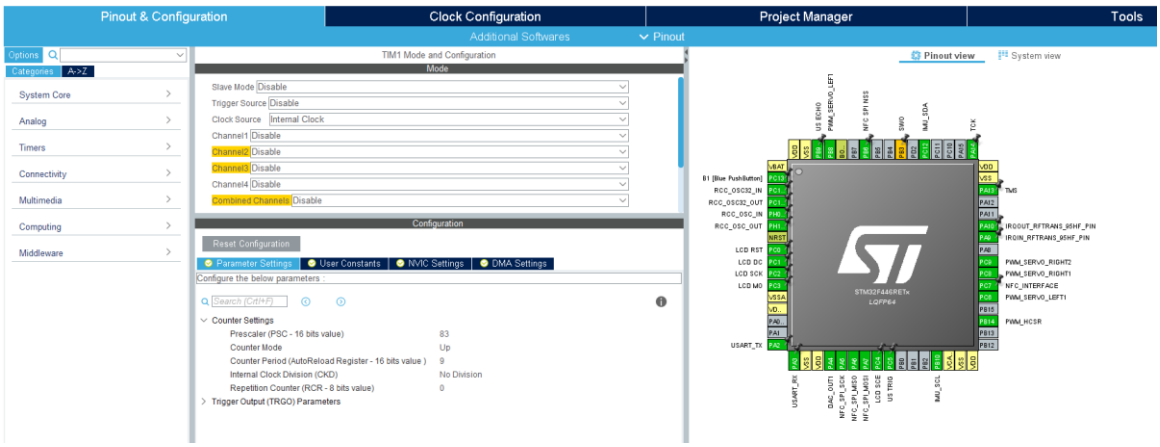

Figure 5. STM32CubeMX example

In addition to not initialize code manually, STM has a large library called HAL (hardware abstraction layer) library, which further simplifies the programming process. For example, starting the same timer as previously defined can be done with a single function call:

```
HAL_TIM_Base_Start_IT(&htim1);
```

Figure 6. HAL example

Moreover, an advantage when using STM microcontrollers is that regardless of the microcontroller, skills learned using one can be used on another STM or another ARM based microcontroller since the underlaying structure is similar.

The chosen STM32F4 family microcontroller is a midway point in its high-performance microcontrollers and was chosen so that as many different protocols and peripherals could be used.

2.2.2 Sensors

Required sensors for the system are NFC reader, ultrasonic sensor, IMU and encoder.

The NFC reader is a booster pack that can be placed on top of the development board and is specifically meant to be used with STM microcontrollers and comes with appropriate libraries to help integrating it.

The ultrasonic sensor main goal is to find obstacles in the direction the sensor is placed. It is easy to use and furthermore it is cheap. Compared to for example the PING module it is up to 10x cheaper and has enough range to measure the distances in the given case of a 2x3 field.

The IMU module was chosen based on a comparison by Adafruit, in which 6 units were compared. The BNO055 was chosen based on the result of that comparison [13].

Initially an encoder was not planned to be used, because with the IMU it is potentially possible to calculate speed based on acceleration. This proved however to be extremely difficult and that plan was abandoned. A Hall effect-based sensor was chosen to measure the rotations of the wheel and based on that calculate speed and distance. Before choosing the specific encoder 2 stand-alone Hall effect sensors were tested.

The tested sensors were US1881 and TLE4906LHALA1 magnet sensors. The first sensor is latch-based sensor, which would require 2 magnets per wheel to work effectively – one polarity to pull its output high and the other to pull it back to low. The other sensor is a switch-based sensor – meaning it will output a high signal only when a magnet is near. Both of the sensors work but when trying to use them near an operating DC (direct

current) motor the switches become unstable because of the magnetic field generated by the motors.

Because of the instability caused by the active motor a specific encoder that was designed to be attached to the motors was chosen instead.

2.2.3 Peripherals

The WiFi module, the LCD and the speaker were all chosen because of their ease of availability. The units were all available and had all been used also in the previous practical tasks for the Embedded Systems course, so they had been proven reliable.

2.2.4 Other hardware

Choosing a suitable chassis for the robot was a difficult task. The chosen chassis had to be big enough so that it would be able to hold a lot of sensors and peripherals, but not too big so that it would be difficult to use in an indoor environment. What made choosing a chassis also difficult was the fact that many of the chassis available on the market either have no connecting holes or do not specify them at all. Because system would be assembled and disassembled by students' multiple times, it had to have all the connection holes predrilled or need minimal work before it can be used.

The chosen Romi chassis was the one of the only, if not the only one that had a specific datasheet available and had plenty of connection holes. As a bonus the chassis came with wheels and a department to place batteries.

Rest of the used parts such as the LDO (low-dropout regulator) for both 5V and 3V3 were chosen so that enough current would be available when all sensors and peripherals are connected. The spike current specified in datasheets was used to add together the required current.

Capacitors and resistors were added to the system based on individual components' datasheets advising them to be included.

The batteries for the external power source were chosen to double the available current instead of necessary available. Other factor when choosing batteries was trying to balance capacity and price.

2.2.5 Software

Software tools used to develop the course consists of previously mentioned STM32CubeMX (version 5.0.1), also a serial reader, git bash, using python programming language, and STM development environment called Atollic TrueSTUDIO for STM32 (version 9.1.0). For debugging LogicPort Application with an Intronix Logicport logic analyzer was used. [14] [15] [16]. Software tools use order is seen from Figure 7.

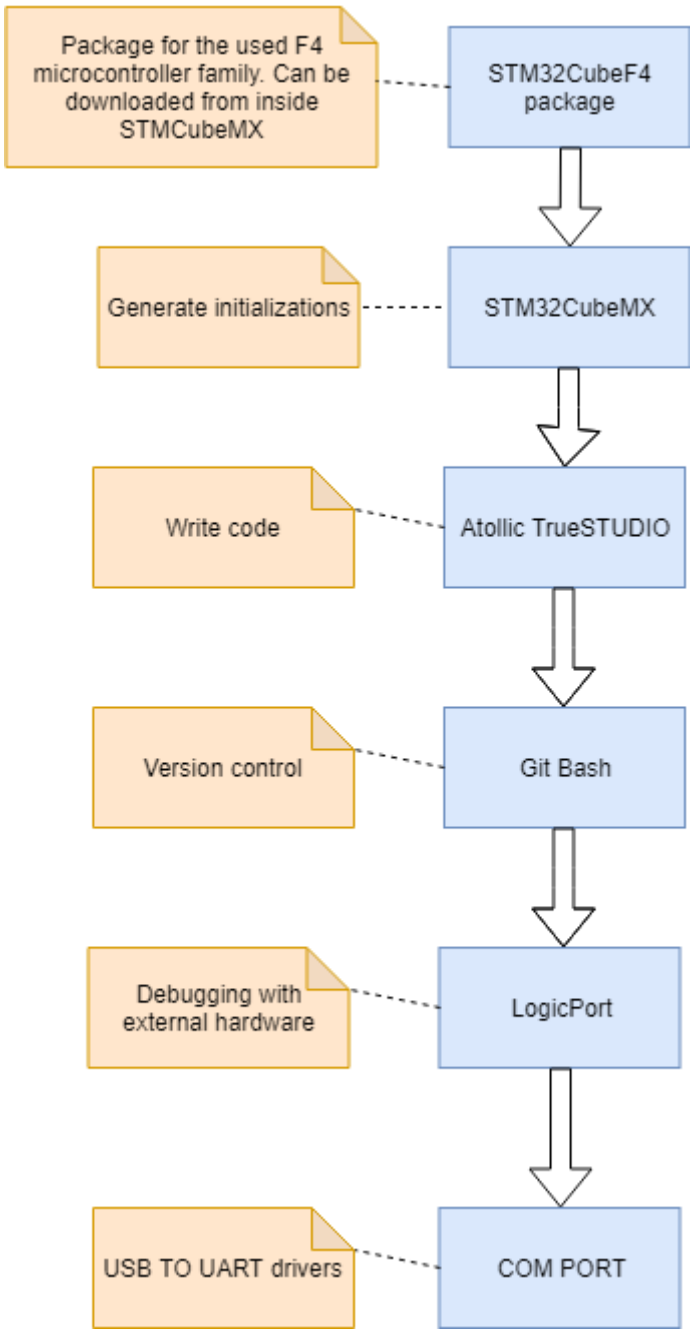

Figure 7. Software setup

In addition, the server side had also to be developed with which the robot would communicate to receive tag location information from. For this purpose, a simple server was setup with Python (version 3.7.1) using the Flask (version 1.0) web application framework, which interfaces with an SQLite (version 3.27.1) database. [17], [18], [19]. Software tools usage order is seen from Figure 8.

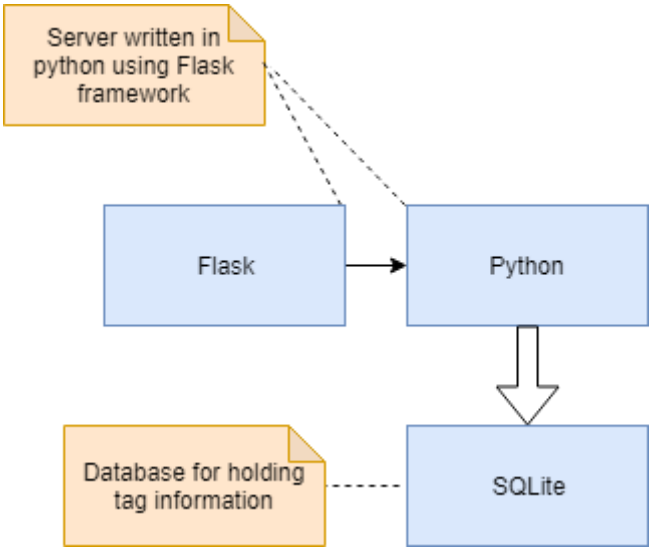

Figure 8. Server software setup

2.3 Exercises' order

The order of the new labs was setup so that more basic concepts are introduced in the beginning and more and more are added to the system the tasks increase in difficulty. For example, the first tasks are about setting up the environment, how to create a project and then adding an LED light that will be made to blink. The full task order can be seen from Figure 9. The navigation task that is one of the main purposes is split between multiple parts, beginning with the motor movement part, in which movement functions are created. With the ultrasonic sensor one way of navigation can be implemented. With the addition of IMU desired movement angle can be calculated. The encoder will finally allow to measure moved distance making the navigation complete.

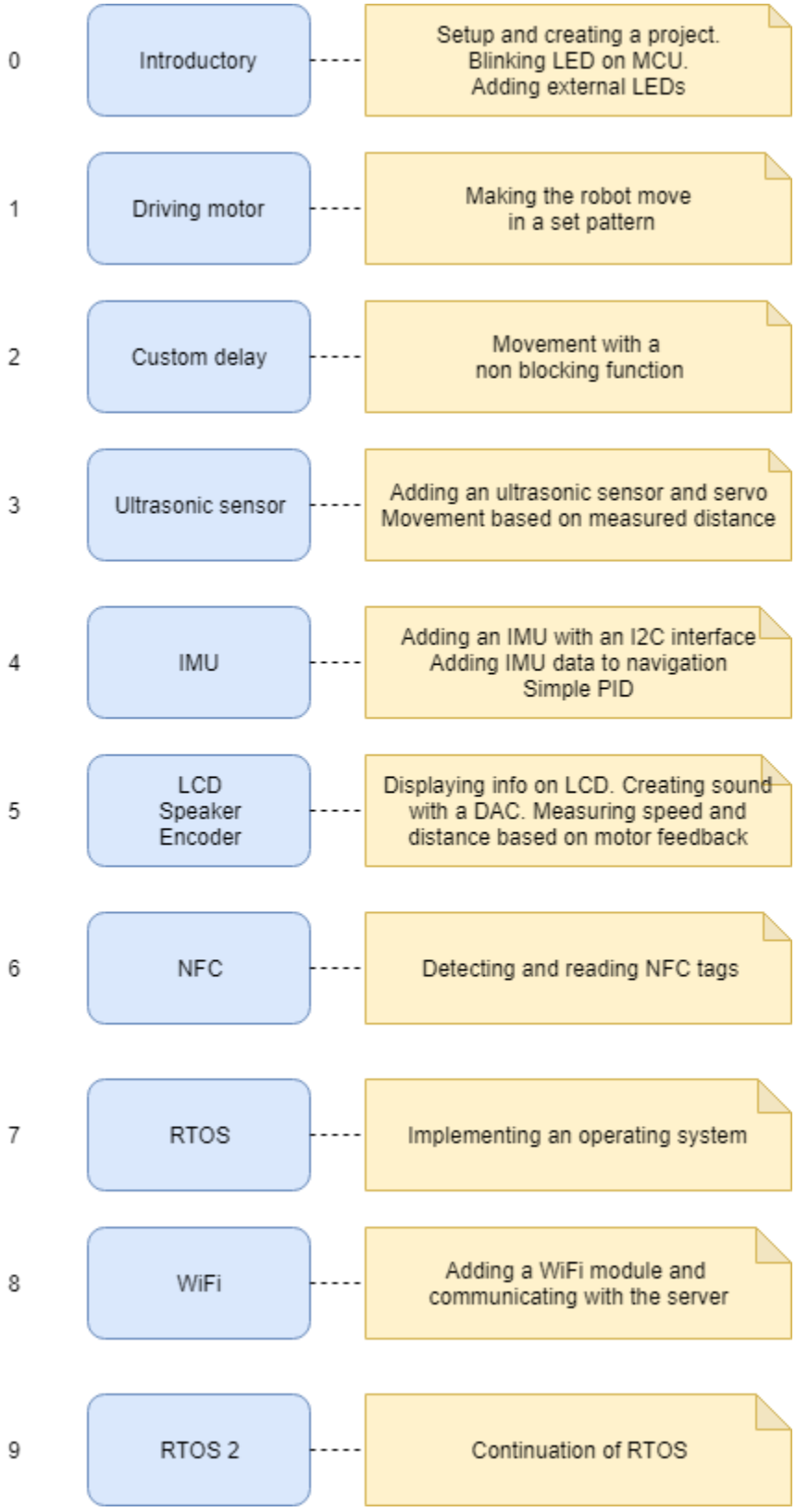

Figure 9. Lab order

The desired connection that should be made can be seen from Figure 10.

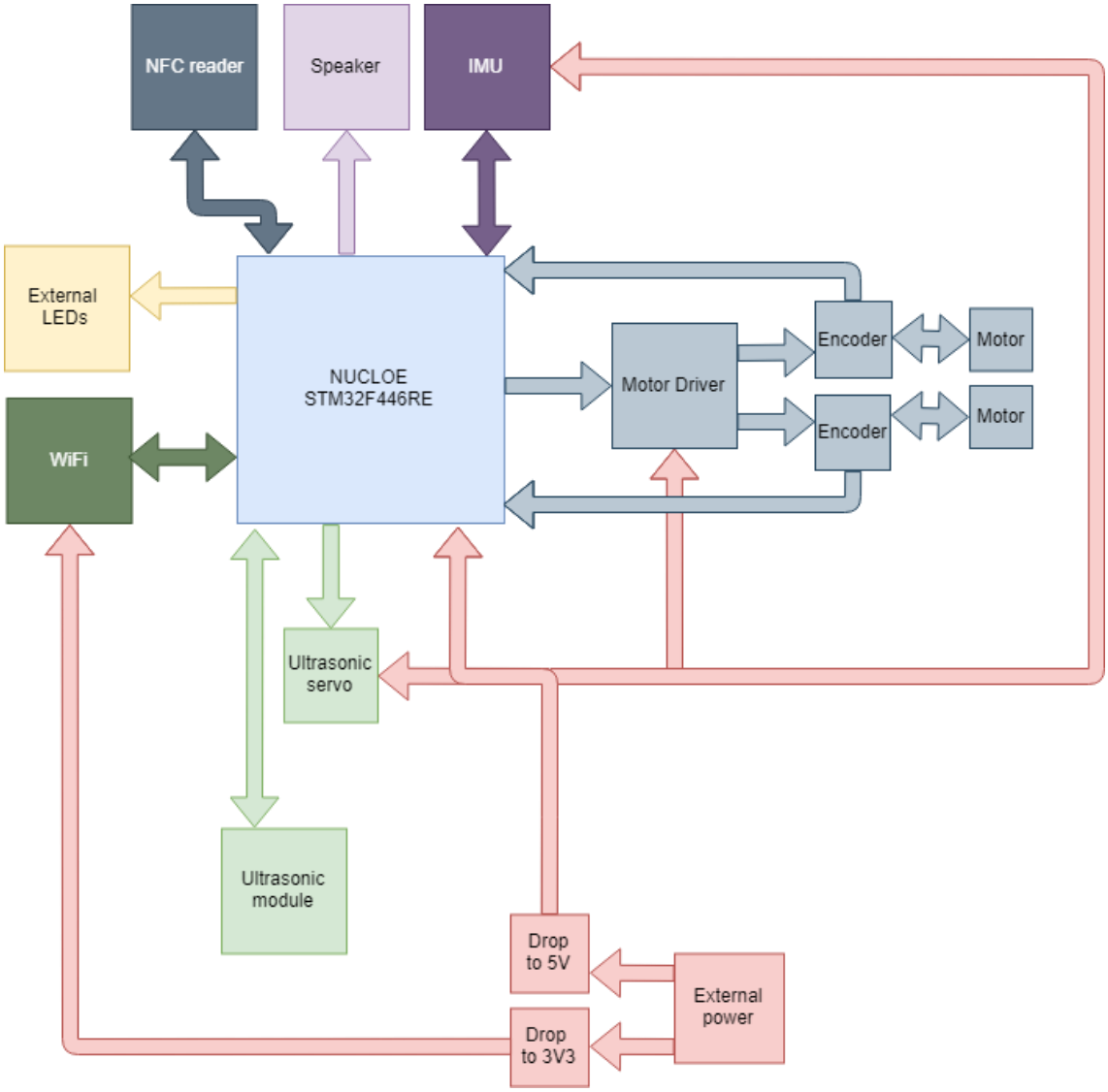

Figure 10. Connection diagram

3 Development

The system was developed iteratively adding parts one by one until the desired system was complete. The development order follows roughly the same order as the one depicted on Figure 9 and as such will be presented in the same order. The task that each part of the system performs is defined and explained alongside with each component.

This section is split in to 11 different paragraphs which will represent a similar order that will be given in the Embedded Systems course. The development begins with running simple LED lights and ends with the implementation on RTOS.

3.1 Introductory

The first step is running one LED on the microcontroller and 3 externally.

The three external LED lights will light up one by one then all turn off and the onboard LED will start flashing ON-OFF every 0.5 seconds once you press the on-board button and will stop flashing after you push it again. The onboard location is seen from Figure 11. The external connection scheme can be seen from Figure 12 which is taken from the STM32 Nucleo-64 family user manual [20].

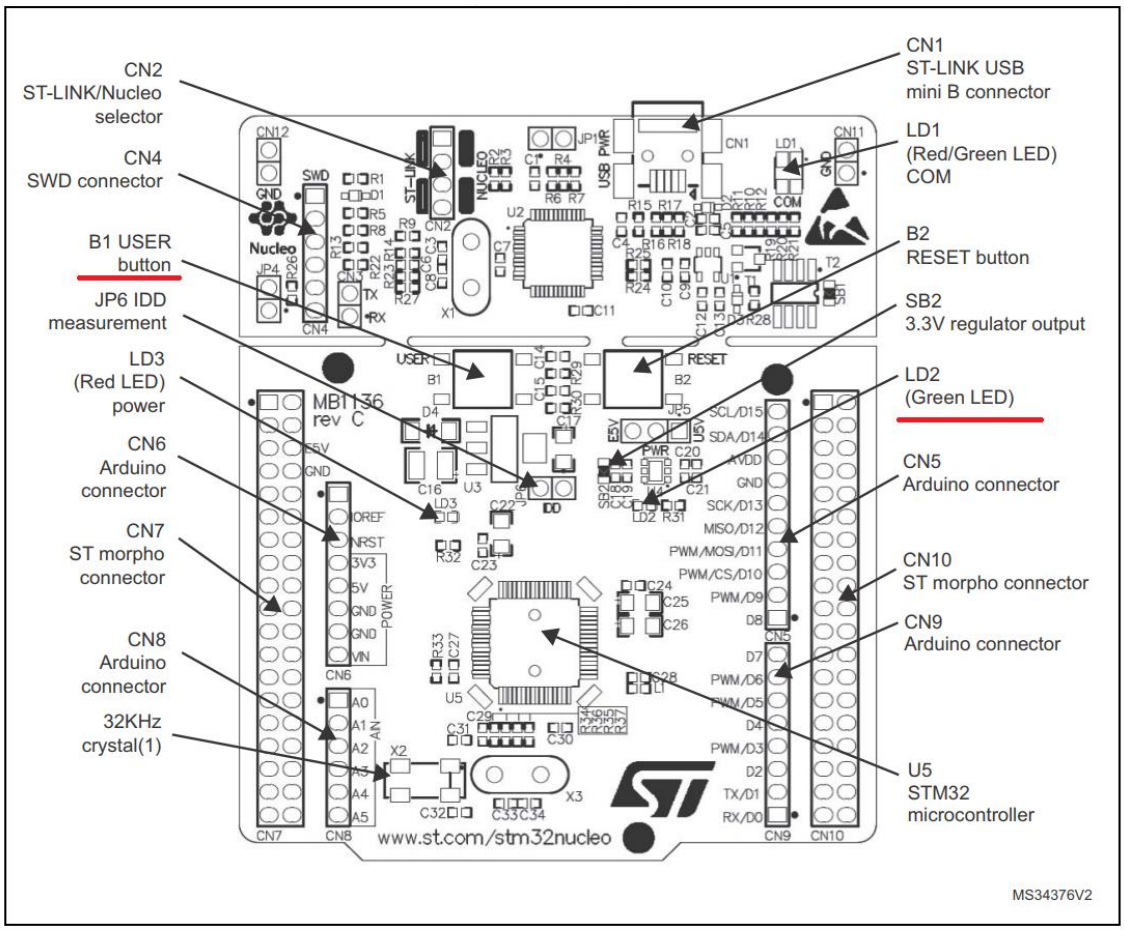

Figure 11. Board top layout [20]

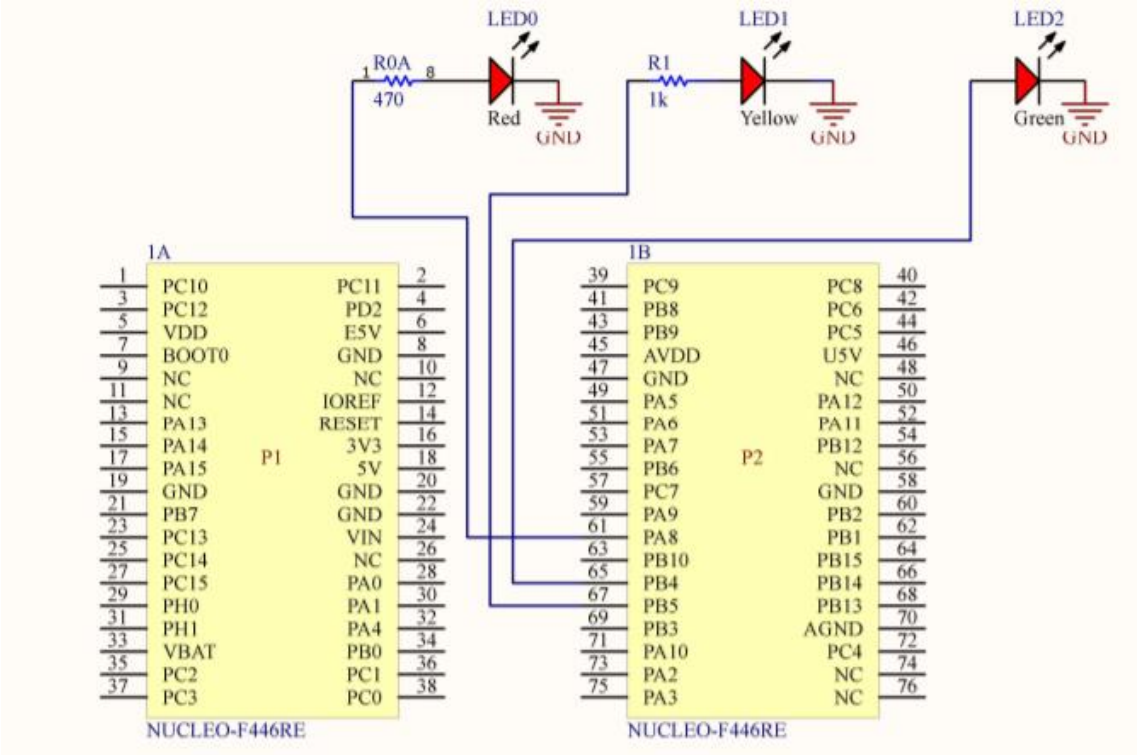

Figure 12. External LED connection

3.1.1 Configuring in STM32CubeMX

In CubeMX the LED and button should already be configured as GPIO_Output and GPIO_Input accordingly. In addition, 3 pins are configured as GPIO_Outputs, these are PB5 as LED_YELLOW, PB4 as LED_GREEN and PA8 as LED_RED. This can be seen from Figure 13.

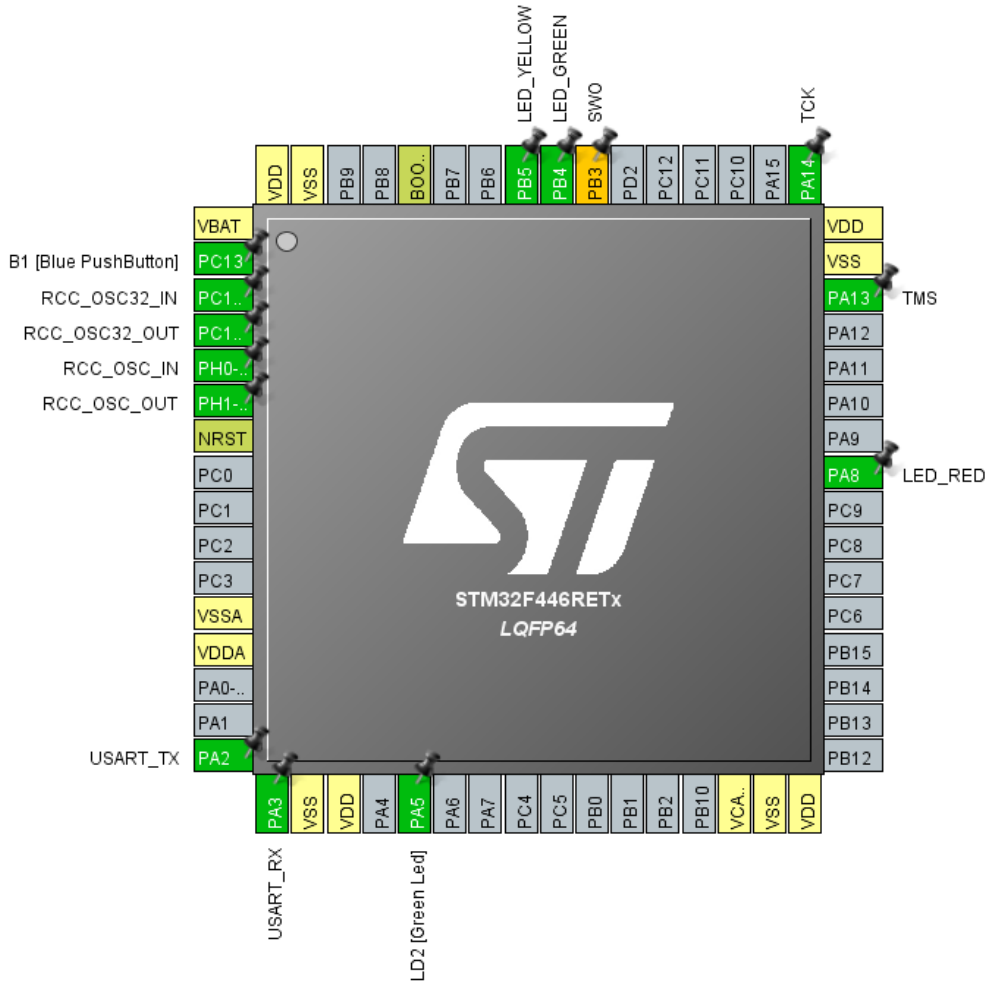

Figure 13. STM32 microcontroller configuration in CubeMX for LEDs

NVIC (nested vectored interrupt controller) interrupts are enabled. “An interrupt is an asynchronous event that causes stopping the execution of the current code on a priority basis.” – from “Mastering STM32”. Interrupts allow to use different approaches to break out of predefined execution order by stopping the ongoing execution, executing the interrupt function then resuming where the execution was stopped. NVIC is a unit responsible for handling interrupts [5].

All configurations that are made in STM32CubeMX are included if the Gitlab repository alongside with the source code [21].

3.1.2 Program

The onboard LED will be toggled with a call from the HAL library function and the three external LEDs will be called with a custom function as seen from Figure 14. For the external LEDs custom functions were created especially so that they could be used as study objects. The code calls each LED to be turned on after a 1000ms delay and in the while loop the LED will be toggled if the button has been pressed. Full source code can be found from a GitLab project [21].

```
LED_On(GREEN);
HAL_Delay(1000);
LED_On(RED);
HAL_Delay(1000);
LED_On(YELLOW);
HAL_Delay(1000);
LED_AllOff();
while (1) {
	if (button) {
		HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
		HAL_Delay(500);
	}
```

Figure 14. Step 1 execution code

3.2 Driving motor

Step 2 covers the assembly and movement of the DCmotors with a PWM (pulse-width modulation) signal. The robot uses 2 DC motors to driver around. This means a motor will require two connections – one VCC (voltage common collector) and one GND (ground) connection.

When no current flows through the motor it will be at a stand-still. Current can be applied on either pin. Depending on which pin current is applied the motor spins one way or the other as seen from Figure 15. The rotation speed is based on how much current is applied by the VCC pin.

In order to make the robot move forward the wheels have to be turning in opposite directions and at roughly same speed.

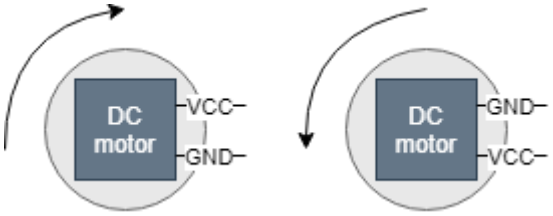

Figure 15. DC motor principle

When connecting wheels directly to the power source there is no control over the speed or direction of the rotations. Instead driver controllers should be used.

A H-bridge driver controller is able to provide voltage both ways to the motors when necessary and can be controller with a PWM signal. Example Figure 16 from the used drv8833 driver's datasheet: [22]

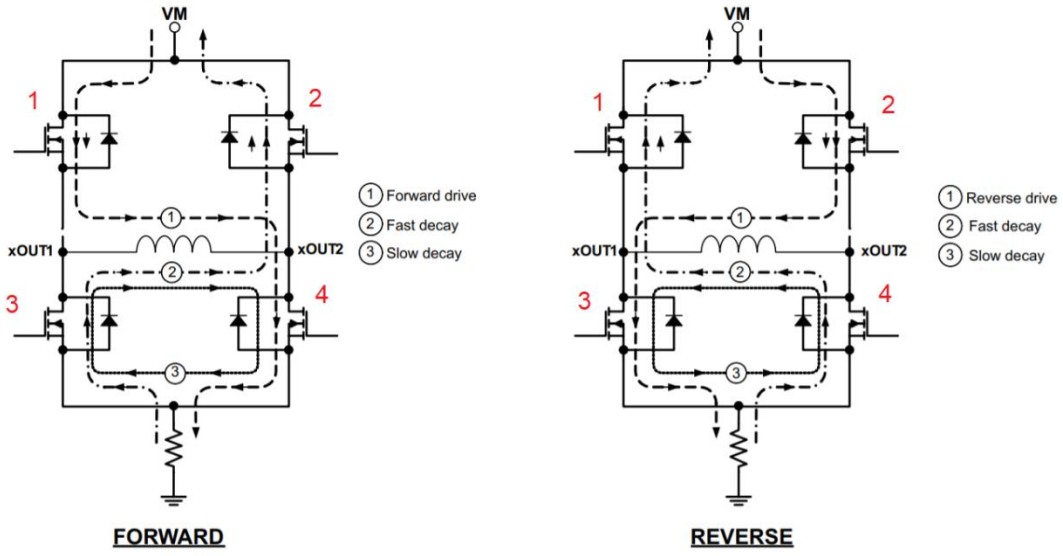

Figure 16. DRV8833 H-bridge [22]

The concept behind H-bridge is that it can allow current to follow to ways. Having gates 1 and 4 closed and 2 and 3 open will allow the H-bridge to drive a motor forward allowing current to flow through one way. Switching the open gates and closed gates will allow the motor to go backwards.

PWM signal to control the H-bridge will be generated by the development board. 4 PWM signals have to be declared but only 1 will only ever be active per wheel. The other inactive signal will represent the GND. A PWM signal is a using digital output that will define how much voltage will be applied to the motor. It essentially shows how great of a percentage of the maximum will be applied as an output.

The recommended voltage for the used motors is 4.5V, but it is specified that it should work well within 3 to 6 V range. For simplicity's sake 5V will be used as our starting base point.

A specific manual on how to assemble the hardware is provided by Pololu and can be found from a GitLab project. The hardware connection scheme can be seen from Figure 17 [21].

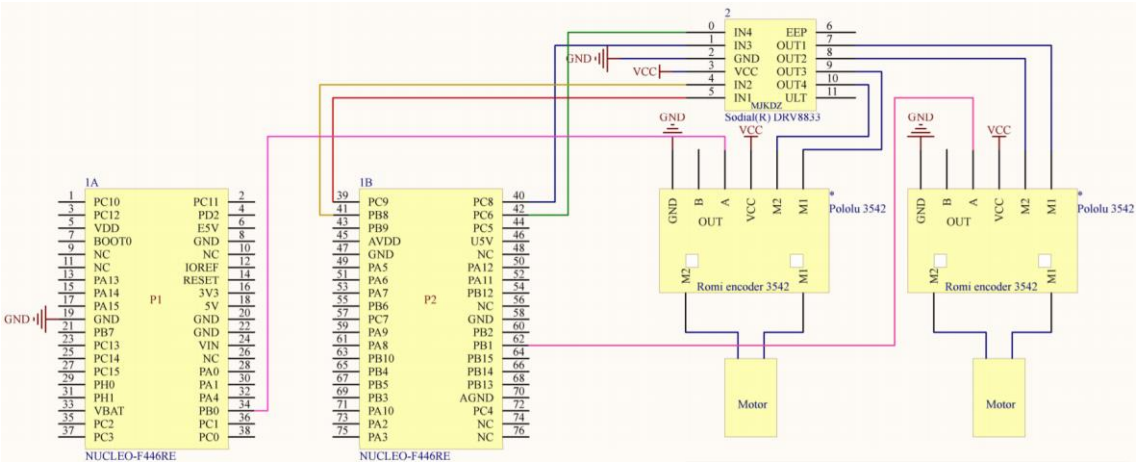

Figure 17. Motor connection

3.2.1 Configuring in STM32CubeMX

In CubeMX to control the motors, 4 PWM signals are defined to be outputted from the GPIO pins. This is done by defining 2 timers, that have a PWM output capability. The timers will be set up as seen from Figure 18. The initial channel pulse value will be 0 for all channels – this will be specified in code, to a desired PWM value, and can change during run time.

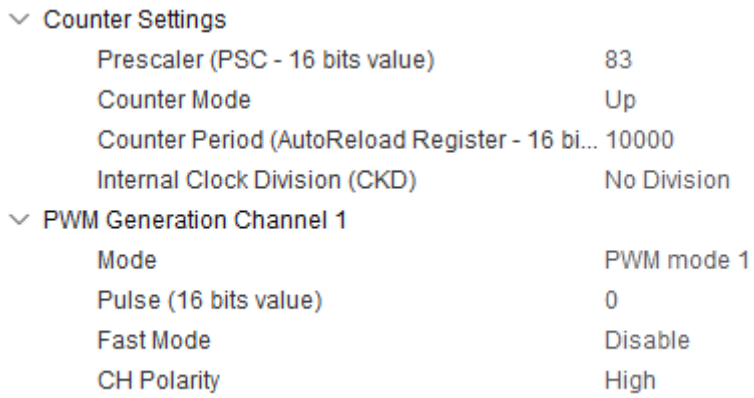

Figure 18. DC motor PWM timer

3.2.2 Program

To move the motors with a PWM signal the timers attached to them must have the pulse values specified first. This is done by functions like done in Figure 19, where the variable PWM value is a predefined number that will represent a part of the whole 10000 that was defined as the full period before.

```
__HAL_TIM_SET_COMPARE(&TimPwmServo, TIM_CHANNEL_1, pwm);
```

Figure 19. Setting PWM value

Once the PWM value has been set on both timers the timers can be started with a function call as seen from Figure 20.

```
void move_straight(uint16_t pwm) {
	start_left1(pwm - 200);
	start_right2(pwm + 400);
}
void start_left1(uint16_t pwm) {
	stop_left2();
	__HAL_TIM_SET_COMPARE(&TimPwmServo, TIM_CHANNEL_1, pwm);
	HAL_TIM_PWM_Start(&TimPwmServo, TIM_CHANNEL_1);
```

Figure 20. Starting PWM timer

From the figure the robot is issued to go forward, for this as an example the left servo is stopped, the PWM variable value set and then the timer is started.

The full source code to set all the timers and start and move functions can be seen from a GitLab project [21].

3.3 Custom delay

Step 3 is to implement a better control system, that will allow the program to use delays without blocking. For this a one new timer is added that will be used for custom delays without blocking that will (almost) always run and check if specific time periods have passed. Delays of various lengths are going to be defined with this single timer, which means the timer is going to be with small period of 10µs (microseconds).

Every 10µs this timer is going to cause an interrupt and with that interrupt the custom delays are going to be checked. The benefit of this can be seen from Figure 21.

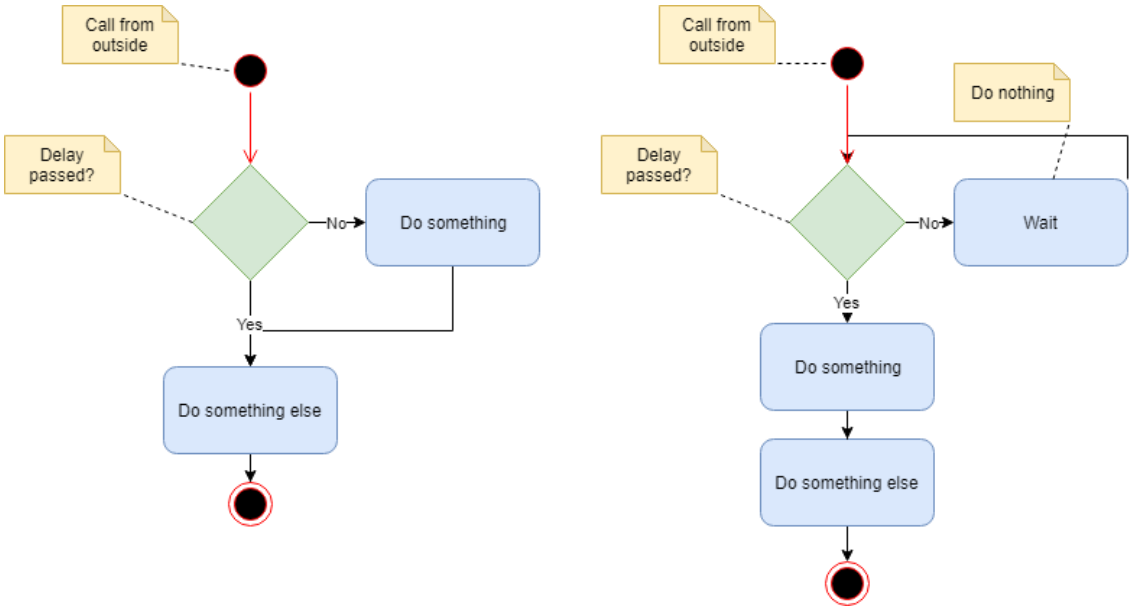

Figure 21. Non-blocking vs blocking delay

3.3.1 Configuring in STM32CubeMX

In CubeMX only 1 time needs to be defined with values seen from Figure 22. This setup will allow for an interrupt every 10µs once the interrupts are enabled.

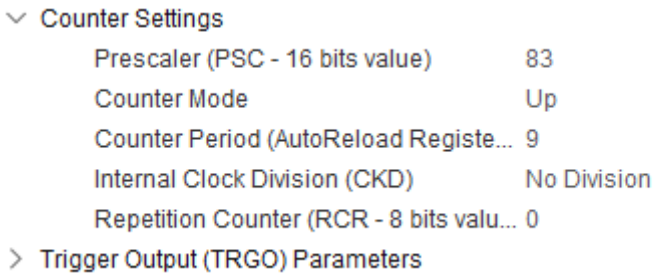

Figure 22. Delay timer

The numbers are calculated based on the following formula:

$$\text{Timer update frequency} = \text{TIM\_CLK}/(\text{TIM\_PSC} + 1)/(\text{TIM\_ARR} + 1)$$

Which when filled in will look like

$$84\,000\,000 / (83 + 1) / (9 + 1) = 100\,000\,(Hz)$$

Resulting in a period of 10 µs.

3.3.2 Program

Each new delay has a definition in the .h files enumerator and a Boolean value as an extern volatile (meaning it is defined somewhere else and it should not be optimised, because its value is going to be changed in an interrupt). An example can be seen from Figure 23. In the figure there are two variables defined. The *delay_servoMove* value is the length of the set delay and the delay*_servoMovePassed* signifies if the time period has passed. The function in the figure is an interrupt *callback*, which is called once at the end of a timer interrupt. Each time the function is a called the delay period is reduced until it is complete, and the Boolean value is set true.

```
volatile uint32_t delay_servoMove;
volatile bool delay_servoMovePassed;

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
	if (htim == &TimDelay) {
		if (delay_servoMove > 0) delay_servoMove--;
		else delay_servoMovePassed = true;
	}
}
```

Figure 23. Delay example

This delay will allow to set period for example to movement functions on how long they should last as seen from Figure 24. A move function will have a specific time period attached to it and after it is passed another move function will be called.

```
void move_turnRight() {
	start_left1(3750);
	start_right1(4000);
	delay_1ms(450, DELAYNAME_servoMove);
}
void move_cyclic() {
	if (delay_servoMovePassed) {
		if (!boolCheck) {
			delay_1ms(100U, DELAYNAME_servoWait);
			boolCheck = true;
		}
		if (delay_servoWaitPassed) {
			if (moveState != MOVE_IDLE) moveState++;
			else moveState = MOVE_T180;
			move_testCyclic();
			boolCheck = true;
		}
	}
}
```

Figure 24. Move function with delay

Full source code can be found from a GitLab project [21].

3.4 Ultrasonic sensor

This step adds another servo and a way to measure distance using the newly created delay. This will allow to create a basic navigation system for the robot.

The servo will move the ultrasonic module through 3 points where it stops briefly, then moves to the next and so on as seen in Figure 25. Every time the servo stops the HC-SR04 sensor takes a measurement. The sensor however can take up to 38ms (millisecond) to return a value. This means that another check needs to be performed before turning the servo. The take measurement block will add another 50ms delay adding some additional buffer time during which the sensor will do the actual measuring as seen in Figure 26. Before the servo movement to the next position starts a check for making sure the measuring has ended should also be added, before initiating the movement process.

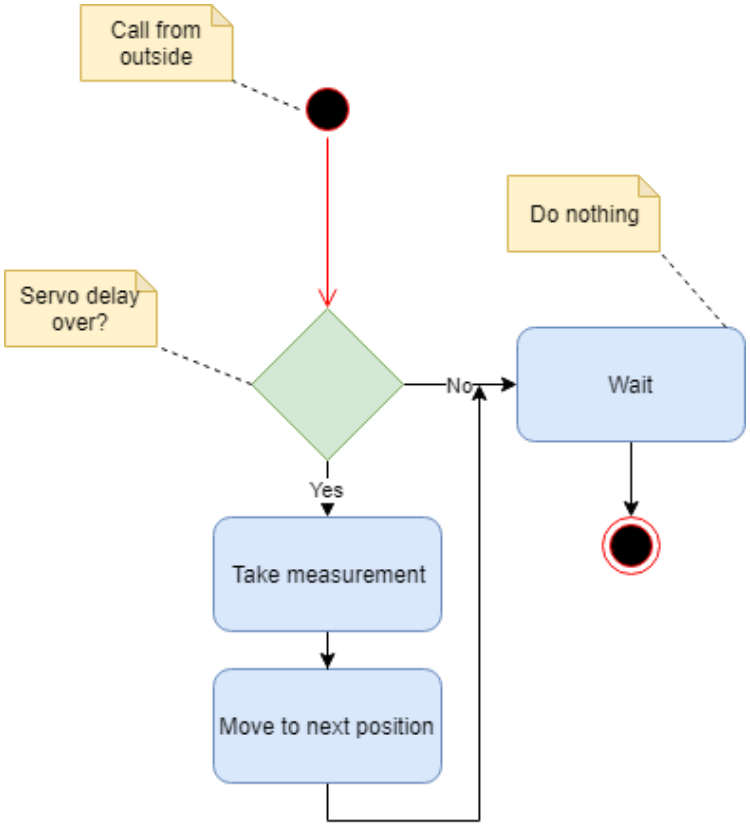

Figure 25. General ultrasonic measurement concept

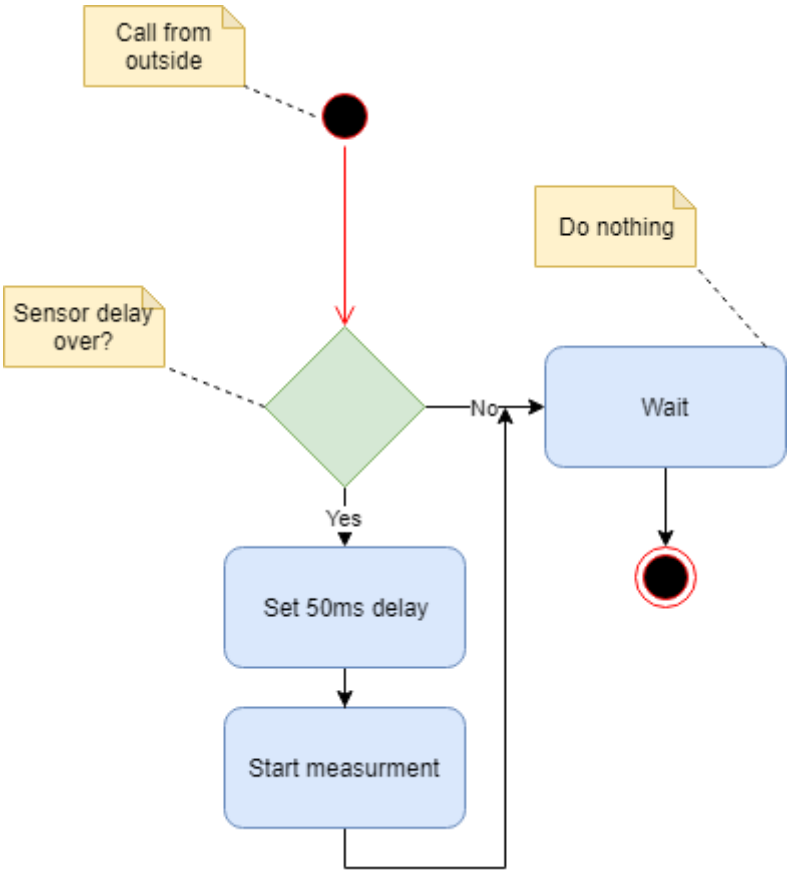

Figure 26. Take measurement block

The physical connection scheme can be seen from Figure 27.

Controlling the servo requires a PWM signal like with the motors, but this specific servo requires the low time between each high period to be 20ms. The high time will specify to what position the servo will move. To accomplish this the register values for the used timers will be changed before each movement [23].

The values for each servo may be different and should be calibrated manually – this means calculating the proper values required to bring the servo to the desired positions.

The ultrasonic sensor works by having a short signal sent to it, after which it will send out a signal and based on how fast the signal returns to the sensor is how big is the measured distance [24].

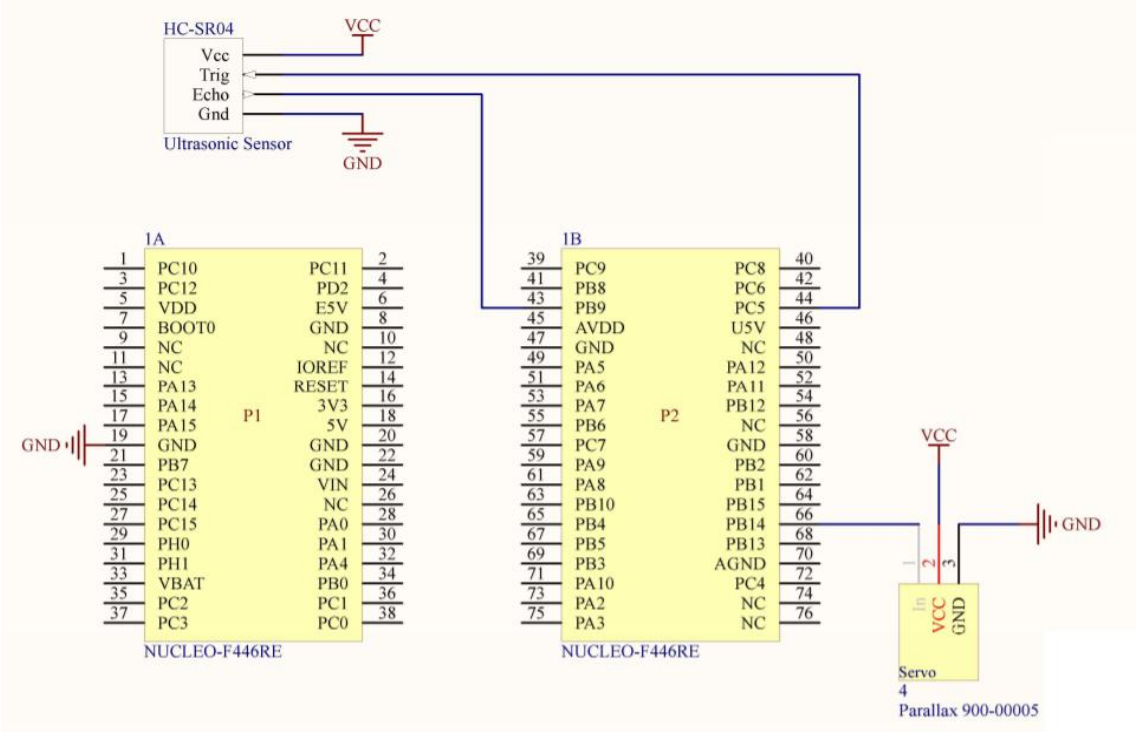

Figure 27. Ultrasonic connection

3.4.1 Configuring in STM32CubeMX

Three pins have to be defined for the ultrasonic sensor and one for the servo it is stood on. The pins are defined in STM32CUBEMX as US_TRIG_Pin as GPIO_output and US_ECHO_ Pin as GPIO_EXTI9 (Which is an external interrupt). Pins can be seen from Figure 28. In addition to the pins a timer is defined that will measure the length of the signal that is delivered back by the ultrasonic module. For this timer TIM12 is defined with the initial parameters as seen from Figure 29.

The timer will use the internal clock source as its base clock, which is 84MHz. We will use the prescaler 83. Using 83 will allow us to get 1MHz frequency rate. This means that the counter will be incremented every 1uS. 83 is used because an extra 1 cycle is included in the system, so the frequency is 84MHz/84 resulting in the desired 1MHz frequency.

We want the counter counting from 0 and going up so it will be easier to calculate the measured distance.

Counter Period is set as 40000. This value means that when the counter reaches 40000 it is reset to 0. 40000 μs = 40ms and the HC-SR04 sensor will send back at maximum a 38ms pulse. We will have 2ms as an extra cushion.

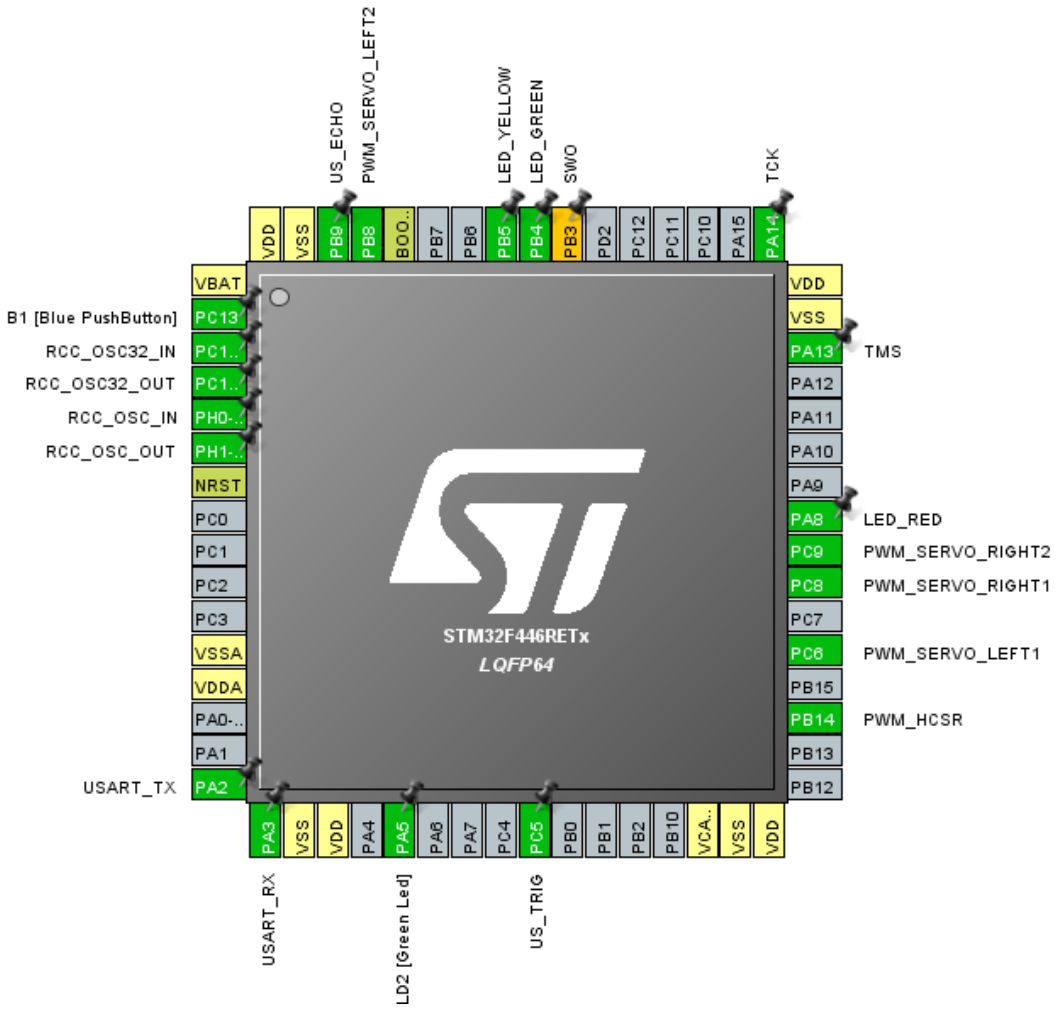

Figure 28. STM32 microcontroller configuration in CubeMX for ultrasonic module

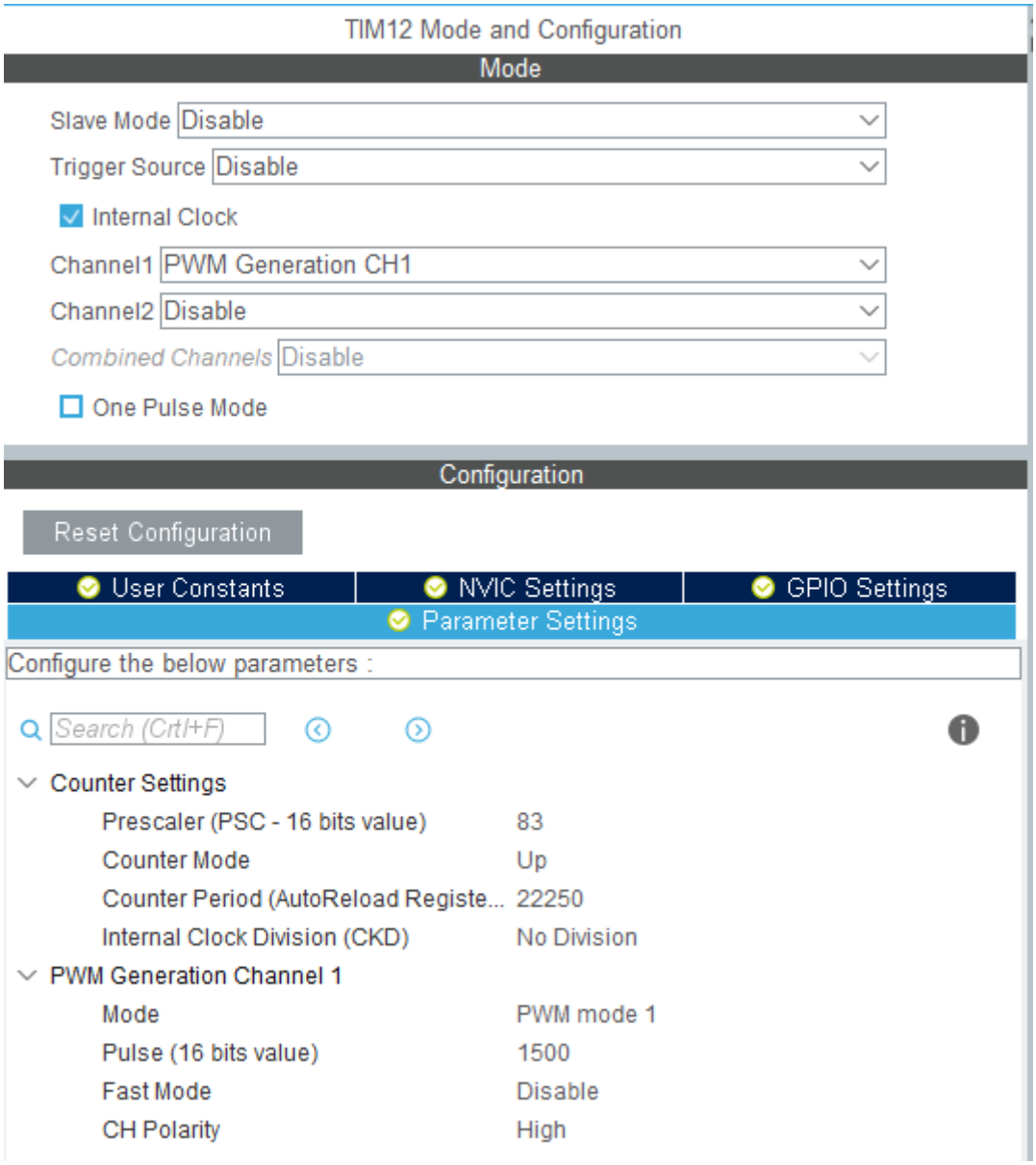

Figure 29. Ultrasonic timer settings

3.4.2 Program

The ultrasonic module works by measuring the signal length with an interrupt as seen from Figure 30. Between each measurement the servo the ultrasonic module is stood on turns 90 degrees by utilizing the custom defined delay. During the turning process no measurement takes place.

```
volatile uint32_t rawVal;
volatile bool measured;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* US ECHO interrupt */
  if(GPIO_Pin == US_ECHO_Pin)
  {
       if ( HAL_GPIO_ReadPin(GPIOB, US_ECHO_Pin) == 1) //Rising
       {
              /* Start timer */
              HAL_TIM_Base_Start(&TimHandle1USres);
       }
       if ( HAL_GPIO_ReadPin(GPIOB, US_ECHO_Pin) == 0) //Falling
       {
              /* Get counter value */
              rawVal = TimHandle1USres.Instance->CNT;
              measured = true;
       }
  }

}
```

Figure 30. Ultrasonic measurement

The full source code can be found from a GitLab project [21].

Based on the measured distances from three angles it is possible to implement a simple navigation algorithm. The implemented algorithm in this step will try to move the robot as long as possible forward, if an obstacle is found, sides are checked for further objects if none are found, the robot will turn to that side. If both sides have obstacles a 180 degree turn will be done instead. The diagram for this can be seen in Figure 31.

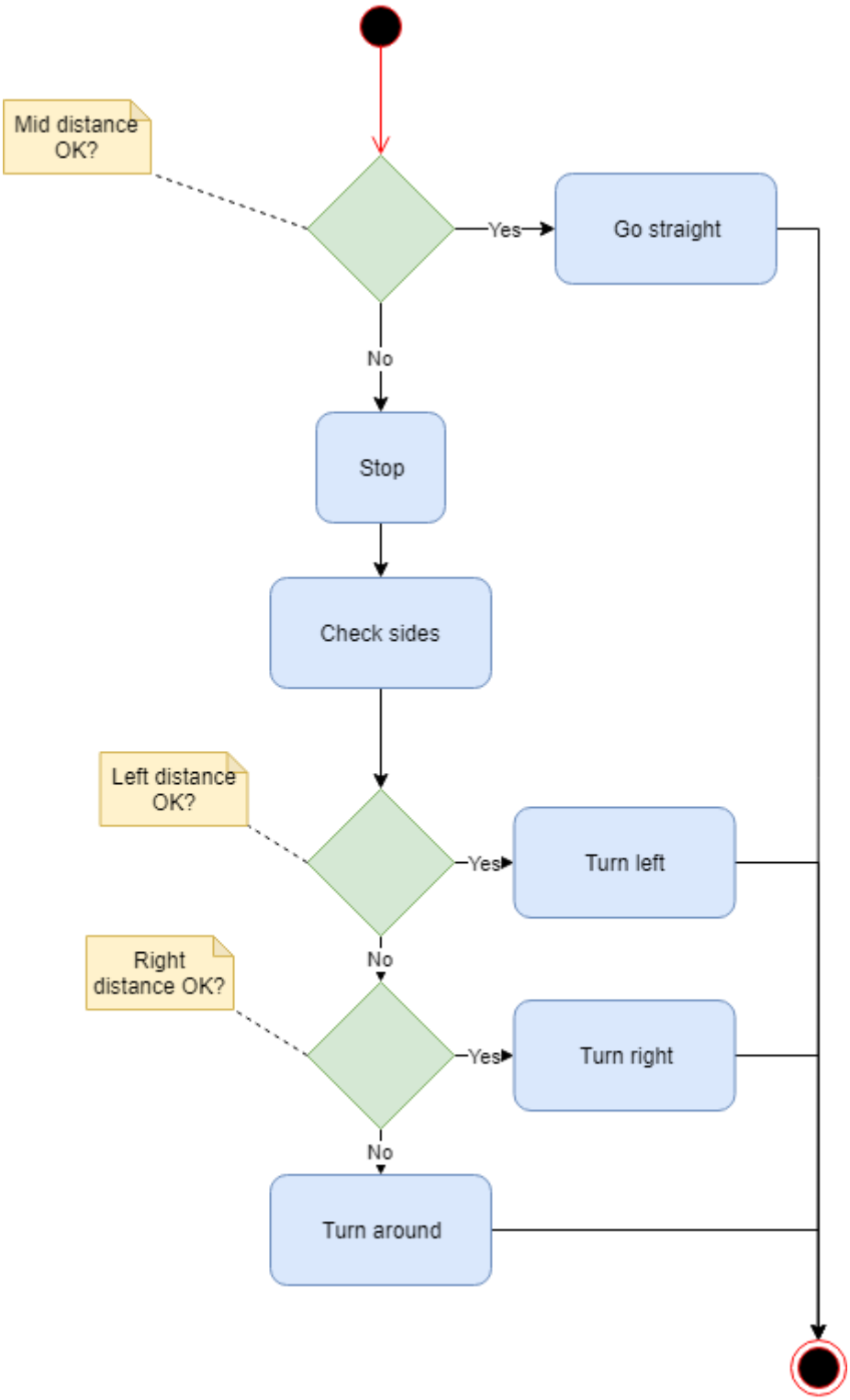

Figure 31. Basic navigation diagram

3.5 IMU

This step adds an IMU to the robot. The IMU is capable of 9-axis sensor fusion. This means that we can use local and global positioning angles for the robot for heading and use the accelerometer to check for movement.

Using an IMU will enhance the navigation capabilities of the robot. With the IMU it is possible to get the direction of the robot. It is also possible to get the acceleration of the robot. The acceleration however will not be used because it would be unstable. Theoretically it is possible to get the speed and position from it by integrating the acceleration over time. This would require implementing very precise filters. For example implementing a Kalman filter would require a large amount of time and the theory behind it is something that should be covered in another course. We can however just check if the robot is moving by checking the acceleration.

The IMU module can be used in conjunction with the previously implemented ultrasonic sensor to create a better navigation system. By using the two walls on the field and using IMUs gyroscope for orientation we can detected which wall the robot is facing and by reading the distance to the wall with the ultrasonic sensor, calculate the robot's location on the field.

3.5.1 Hardware connections

Connection scheme can be seen from Figure 32. For selecting I2C (inter-integrated circuit) as the communication protocol PS0 and PS1 pins have to also be set to low (grounded).

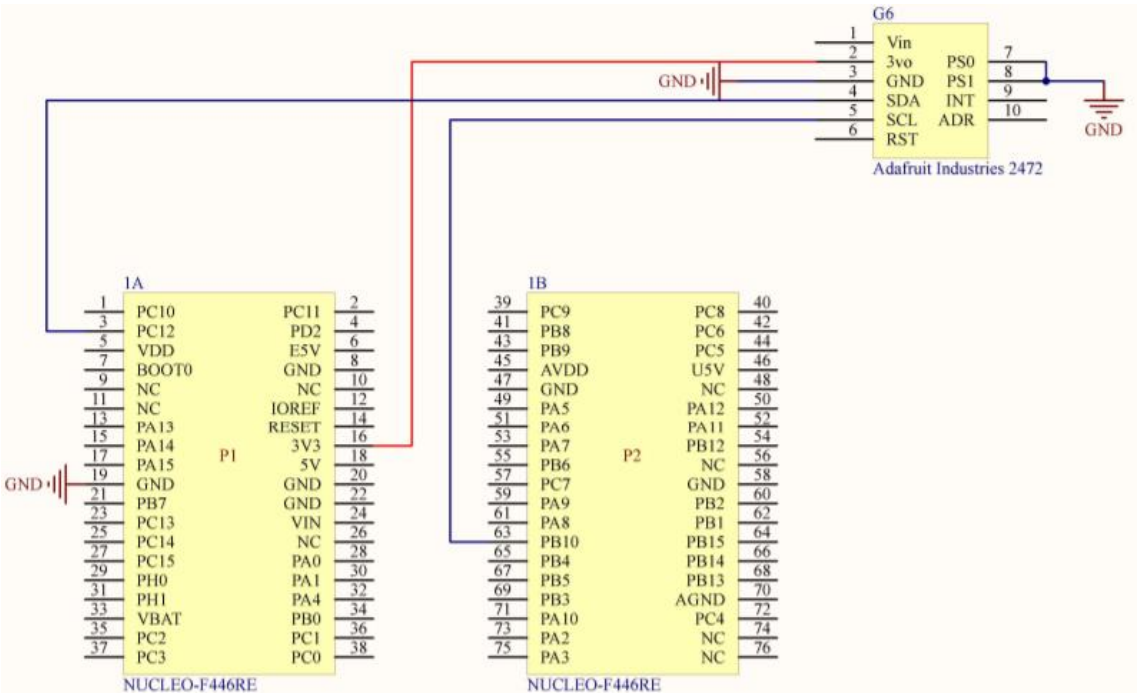

Figure 32. IMU connection

3.5.2 I2C

I2C is a communication protocol which requires only 2 wires for communicating with potentially hundreds of devices. The two wires are SDA (serial data line) and SCL (serial clock line). I2C can be used with even devices that don't support it by manually configuring the ports to detect and or output signals at specific rates.

I2C data transmission begins with a high to low transition on the data line while clock line is high. Data is sent with a specified, 7-bit in this case, length bit by bit, with ack's in between each. Data transmission ends with a low to high transition while clock is low. Timing for I2C as described in BNO055's datasheet can be seen also on Figure 33 [25].

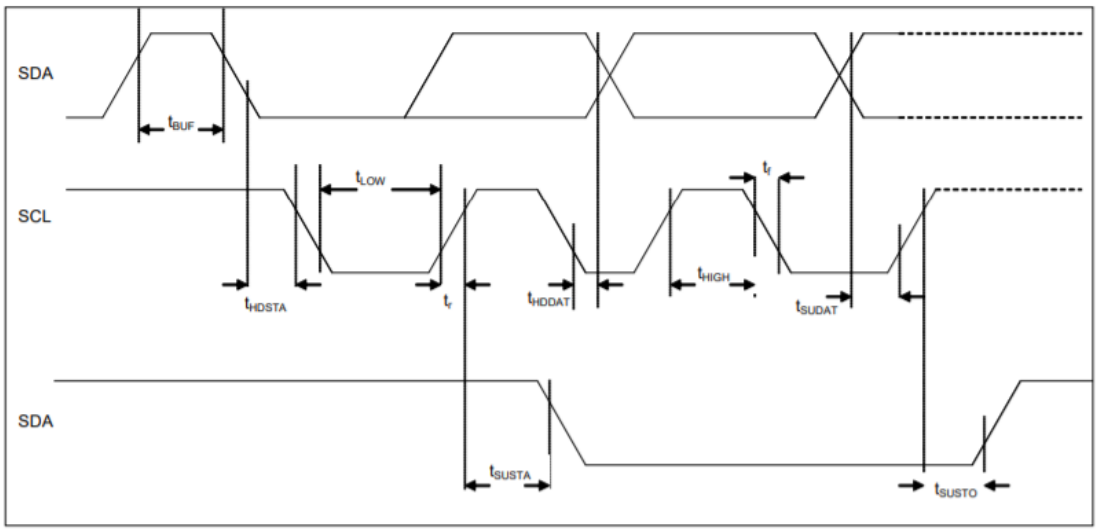

Figure 33. I2C timing diagram [25]

I2C is fortunately supported by the NUCLEO-F446RE board and can be configured with STM32CubeMX.

3.5.3 Configuring in STM32CubeMX

NVIC, DMA (direct memory access) and GPIO setting should however be changed. NVIC I2C 2 interrupts should be both activated. Under DMA settings 2 DMA requests should be added – both RX (receive) and TX (transmit) request. This allows for a non-blocking communication. These DMA settings allow for faster and non-blocking data transfers between the microcontroller and the specified peripherals with the communication protocol.

3.5.4 Program

Using I2C to communicate with IMU requires only the specific HAL functions to be called with the appropriate parameters as seen from Figure 34.

```
HAL_I2C_Mem_Read_DMA(hi2c_device, BNO055_I2C_ADDR_LO << 1,
                     BNO055_MAG_DATA_X_LSB, I2C_MEMADD_SIZE_8BIT, str,
                     IMU_READ_NUM_OF_BYTES);
```

Figure 34. I2C HAL function call

The first parameter to this function is the used I2C device. The function call itself reads data from the IMU at a specified *BNO055_I2C_ADDR_LO << 1* address, at a specific location at the target of a specific size. The read data is stored is the *str* buffer and a size to be read is the last specified parameter.

In addition to the I2C communication a simple PID (proportional-integral-derivative) controller is implemented to try to keep the robot moving straight. A PID controller is a feedback-based controller using a setpoint, current value and integrating over time to adjust a value.

To accomplish this the PWM value given to the motors is changed based on the read data from the IMU as seen from Figure 35.

```
uint16_t tmp = (uint16_t)g_imu_data.eul_head;
	switch (current_dir) {
		case MOVE_NORTH:
			if ((tmp == 360) || (tmp == 0)) {
				drift_coefficent = BASE_COEFFICENT;
			}
			else {
				if ((tmp > 355) || (tmp < 5)) ;
				else if ((tmp >= 5)
						&& (tmp < 100))
					drift_coefficent += 2;
				else if ((tmp <= 355)
						&& (tmp > 250))
					drift_coefficent -= 2;
			}
			break;
```

Figure 35. Simple PID

The read direction data is checked if it falls between a certain values. If the value starts to fall off the PWM values are adjusted with a coefficient to only one wheel.

Full source code can be found from a GitLab project [21].

3.6 LCD, speaker, encoder

This step adds an LCD, a speaker and an encoder for the motors to the system. The LCD will help display messages since the system will not be connected to the PC (personal computer) all the times. The speaker will be used with a DAC to show how data can be converted to analog signal. The encoder will allow to calculate the speed and travelled distance of the robot.

The speed and distance can be calculated based on the circumference of the wheels. Knowing the distance travelled can help position the robot and further improve the navigation system.

3.6.1 Transmission

The LCD data transmission operates at a bit by bit level in synchronization with the clock signal. See Figure 36 [26].

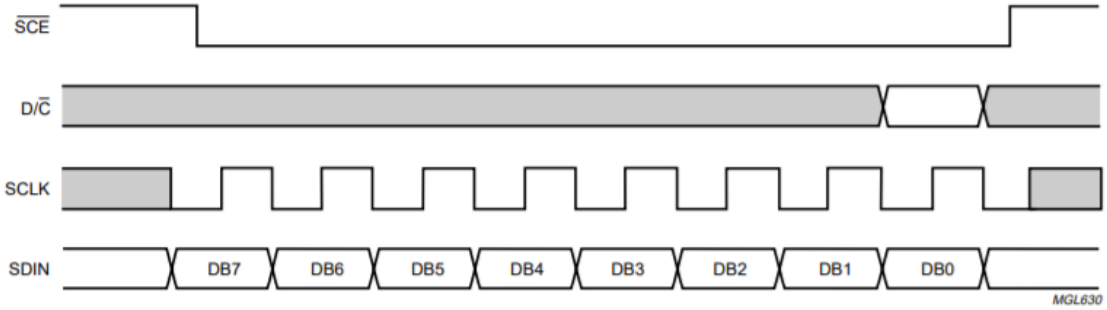

Figure 36. Transmission of one byte [26]

SCE (chip enable)
D/C (mode select)
SCLK (serial clock line)
SDIN (serial data line)

3.6.2 Speaker

Using a speaker requires the use of a DAC. The speaker is operated by giving an oscillating signal to it. The oscillating speed determines the output sound pitch.
To get for example an output of 2092 Hz:
A sine (or triangle, square or sawtooth) wave must be defined. The sine wave will consist of 8 points and will be with a 4 bit point size. The points of the sine wave that are taken can be seen from Figure 37. The accuracy of the wave is small for this example because of the small number of points. More points should be use for better accuracy.

NB! the used piezo speaker has a maximum frequency of 1800+- 20% Hz, so these values are not valid for the used speaker [27].

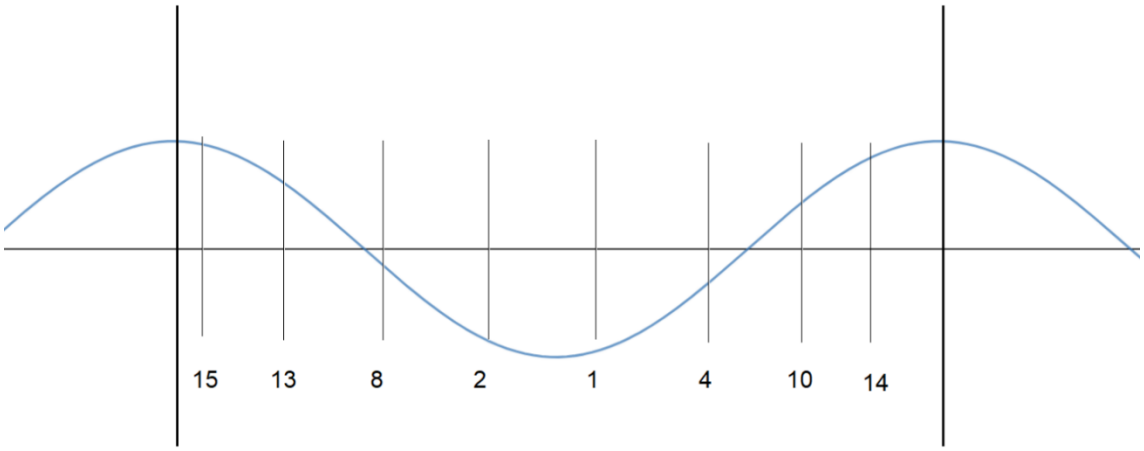

Figure 37. DAC sine wave example

After the wave points have been defined, the points have to be outputted with DAC at a rate that is x times faster than the number of chosen points. In this case 8 times faster. This means 2092 * 8 = 16736 Hz. 16.736 kHz is the rate at which the DAC must output the defined points.

For this purpose, a timer designed specifically for this purpose has to be implemented. To make the timer run at the desired rate it should be calculated as:

$$\text{Timer update frequency} = \text{TIM\_CLK}/(\text{TIM\_PSC}+1)/ (\text{TIM\_ARR} + 1)$$

$$16736 = 84\,000\,000 / 1 / x$$

Where is in this case will be 5019. This 5019-1 will be the counter period when defining the timer.

3.6.3 Encoder

The encoder will output 12 counts per revolution, as per its name. This means that the two outputs that the encoder has will each output 6. Based in the order which output comes first, the direction of the rotation can be determined. When counting all revolutions in a specific time period the speed of the robot can be calculated based on the wheel's circumference.

Taking in to account the 120:1 ration of the output shaft to motor shaft, it is possible to divide the number of counts the encoder makes by 1440 to get the revolutions of the wheel.

Multiplying the revolutions with the wheels circumference the distance is calculated. Further dividing the distance travelled by time the robots speed is calculated.

3.6.4 Hardware setup

When connecting the LCD, it should be placed on the edge of the breadboard. The connections for the LCD are as seen from Figure 38. The LED connection is used for the backlight and is not necessary and is thus connected directly to ground.

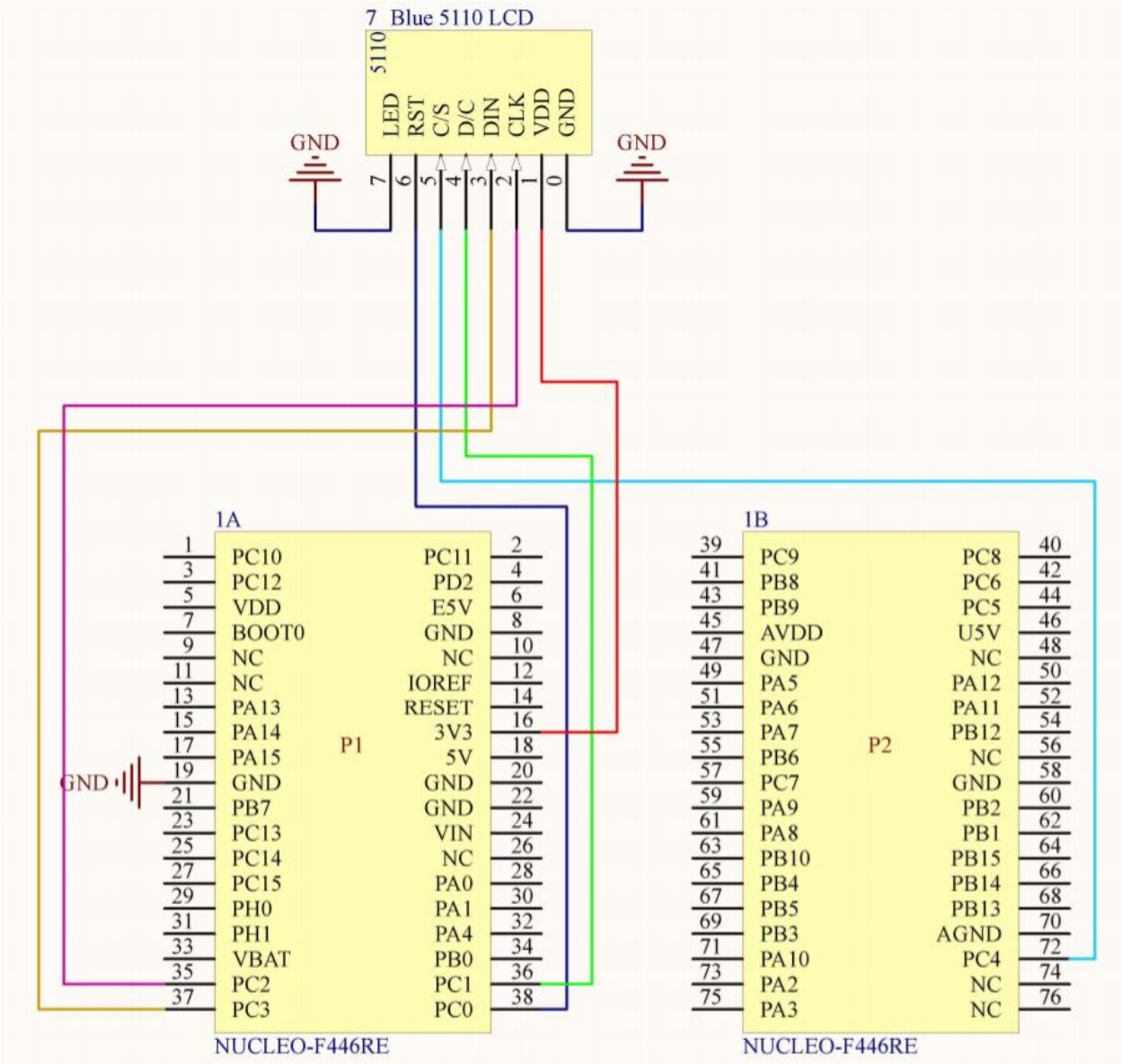

Figure 38 LCD connection

Connecting the speaker is easy, since it has only 2 connections. The negative side will be connected to ground and the positive to the output of the DAC as seen from Figure 39.

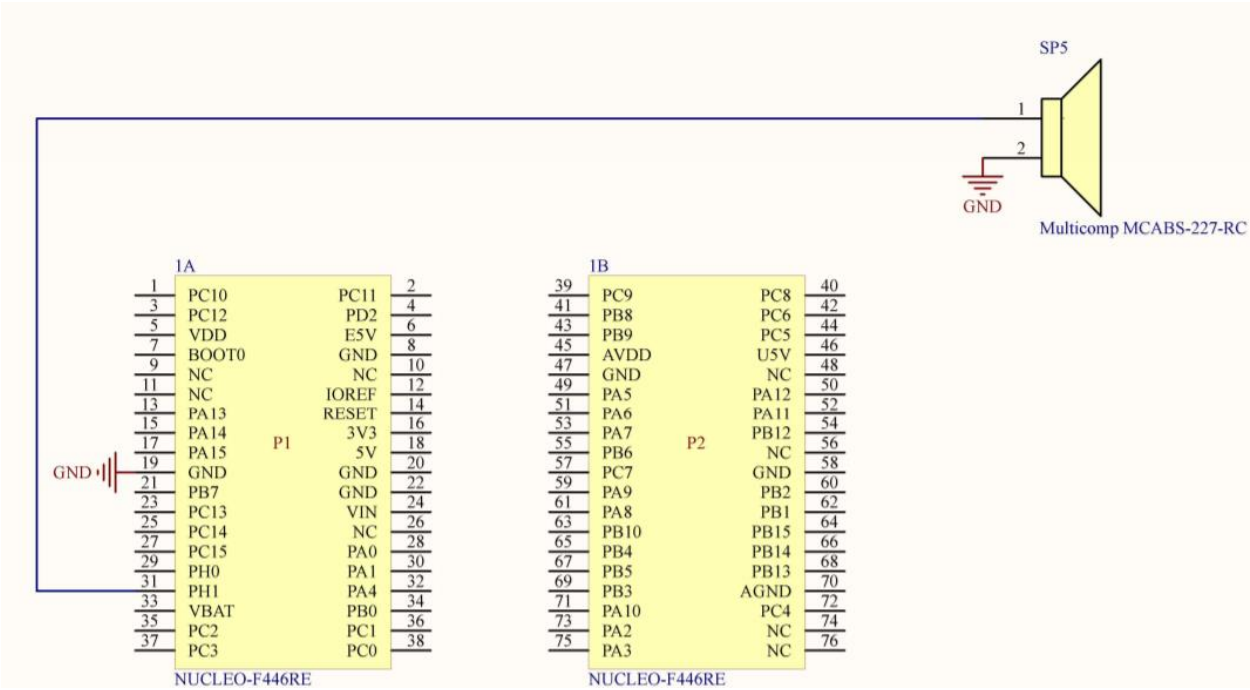

Figure 39 speaker connection

Half of the connections for the encoder should have already been done in a previous step. The rest of the connections if not already connected should be connected as per Figure 40.

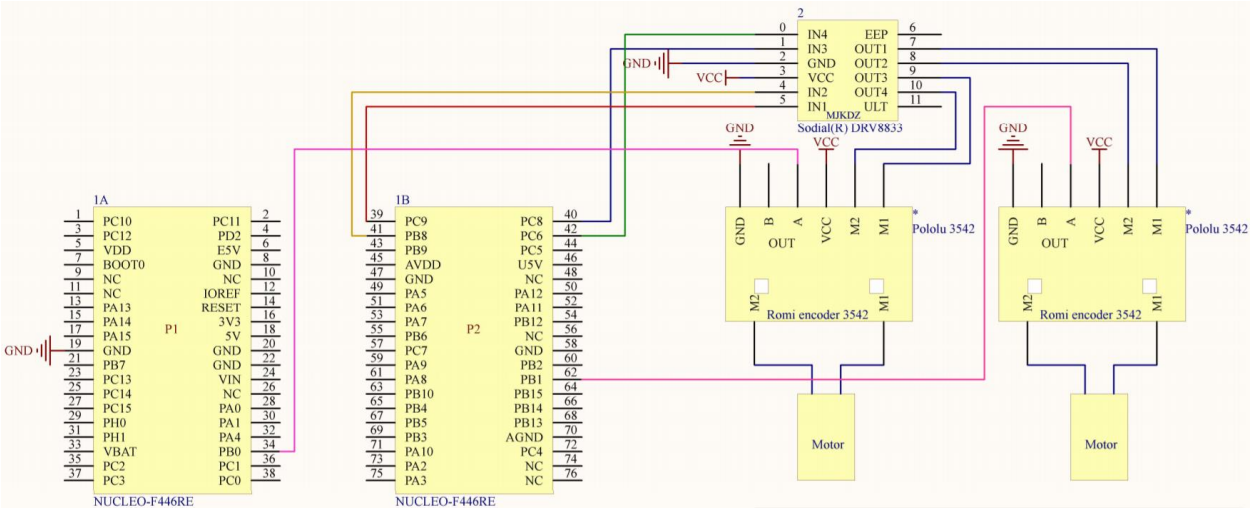

Figure 40 Motor connection

3.6.5 Configuring in STM32CubeMX

Setting up the LCD requires all the used pins to be set as GPIO_Outputs with default settings. The rest of the pin location are seen on Figure 41 along with the used DAC pin location.

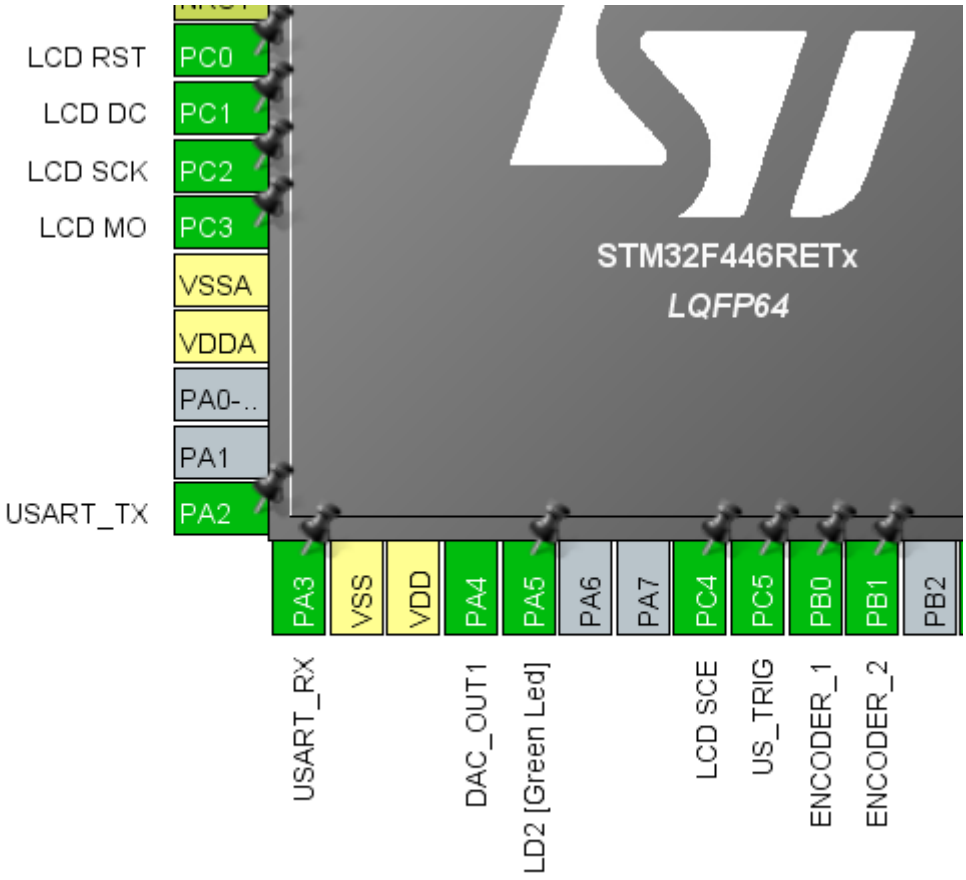

Figure 41. STM32 microcontroller configuration in CubeMX for LCD, DAC, Encoder pins

Configuring the DAC requires enabling timer 6 with a period of 0x139A, which is 5018 in decimal. This value was calculated as seen in the speaker paragraph before. The trigger output will be an update event. See Figure 42. NVIC settings interrupts should also be enabled.

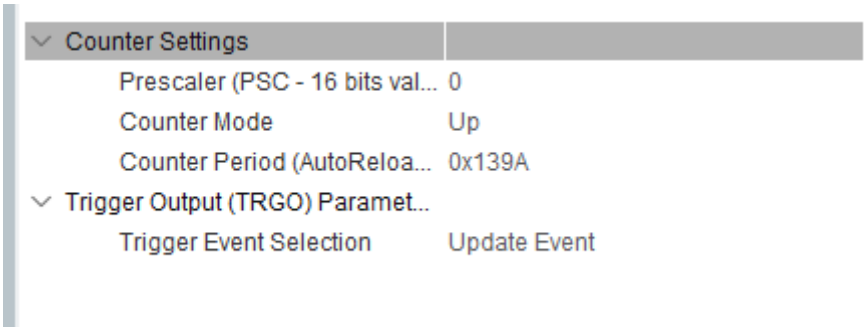

Figure 42 Timer 6 parameters

After configuring the timer that the DAC will use the DAC itself is configured from the Analog tab. Only OUT1 will be used as only 1 speaker is used.
The settings for DAC are seen from Figure 43.

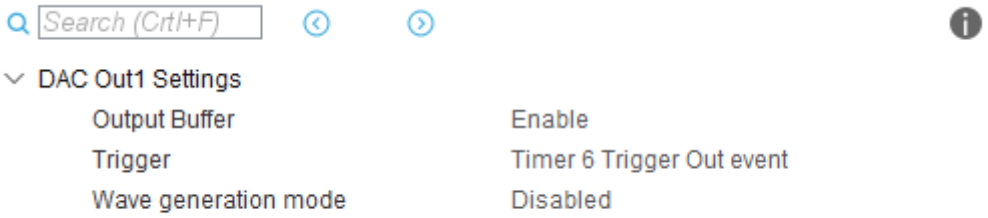

Figure 43 DAC OUT1 settings

Encoder configuration requires 2 external interrupts per wheel. Or 1 per wheel if tracking rotation direction is not necessary. For now, 1 interrupt per wheel will be used. PB0 as EXTI0 and PB1 as EXTI1.

3.6.6 LCD program

The LCD module originally was developed by TalTech IoT development centre. The LCD module functions output data to the LCD 1 character at a time as seen from Figure 44. The code disables output, checks that the mode is correct for data input, then from a predefined font sends bits to the LCD while generating a clock signal. After the characters have been sent, output to the screen is reenabled.

```
void LCD5110_LCD_write_byte(uint8_t dat,uint8_t mode) {
	uint8_t i;

	LCD5110_SCE(0);//LCD_SCE = 0;

	if (0 == mode)
		LCD5110_DC(0);//LCD_DC = 0;
	else
		LCD5110_DC(1);//LCD_DC = 1;

	for(i=0;i<8;i++)
	{
		LCD5110_MO(dat & 0x80);//SPI_MO = dat & 0x80;
		dat = dat<<1;
		LCD5110_SCK(0);//LCD_SCK = 0;
		LCD5110_SCK(1);//LCd_SCK = 1;
	}

	LCD5110_SCE(1);//LCD_SCE = 1;
```

Figure 44. LCD write byte

```
}
```

3.6.7 Speaker program

The speaker requires only to starting it after it has been initialized. The speaker will be activated once the robot reaches the correct place on the field. The value for starting the DAC in DMA mode, are, the DAC itself, 1 of 2 channels, the audio wave and how we aligned the values in the wave, in this case 12 least significant bits are used. Usage is seen from Figure 45.

```
void speaker_cyclic() {
	if (move_GETdst()){
		HAL_TIM_Base_Start(&htim6);
		HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
		HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*)
sine_wave_array, 32,
		DAC_ALIGN_12B_R);

	} else if (speaker_active) {
		HAL_DAC_Stop(&hdac, DAC_CHANNEL_1);
	}
}
```

Figure 45. Speaker code

3.6.8 Encoder

The encoder requires only interrupts that will count how many often they happen, based on which it is possible to calculate the speed.
Because the encoder has 6 magnets inside it and 2 output pins to determine rotation direction 12 interrupts will ideally be generated per rotation. Counting the number of interrupts generated and based on the circumference the distance travelled is calculated.

3.7 NFC

This step adds an NFC module to the system. NFC is a closer range communication protocol. In addition to active devices communicating, an NFC capable device can read and write to an NFC tag, which is a small embedded chip. This is what will be done in this lab. The nucleo NFC reader will be used to read data from NFC tags to get information. Reading a value from an NFC tag will allow the robot to confirm its location by checking whether the data it reads from a tag is what it expects.

Figure 46 shows the basic work concept behind the NFC reader. PCD (proximity coupling device) is switched on to detect nearby tags. Every tag type is searched for differently so when a tag is found its type is already known. After finding a tag it is read.

Communication between the NFC module and the microcontroller is done over SPI (serial peripheral interface). SPI works by sending a data bit in synchronization with a clock signal. The used SPI here is the same as in the LCD module, but with the LCD module the signal was generated manually. Whereas now it used through the HAL library and is used both ways.

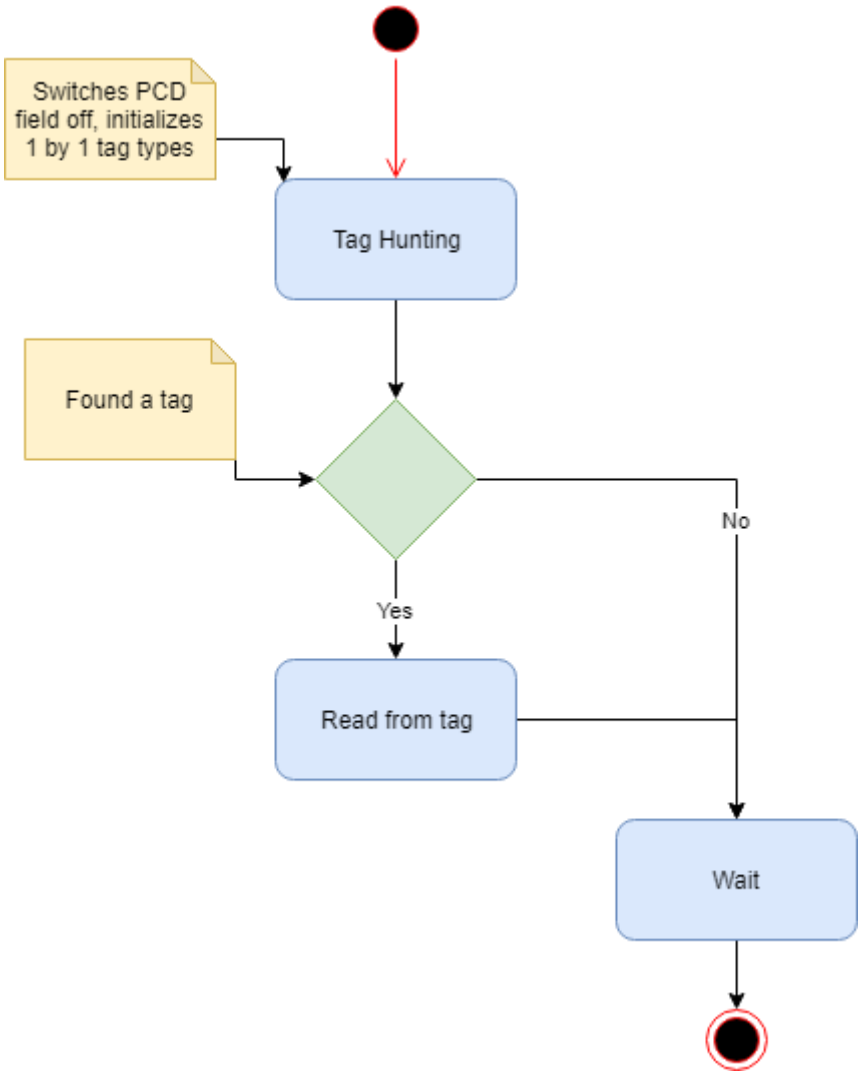

Figure 46. Tag detection concept

3.7.1 Hardware connection

The NFC reader should already be connected to the robot chassis at this point. The reason why the reader was placed under the robot was because of the proximity limitations of NFC technology. NFC communication works best in a range from touch to 4 cm. [28] There are more connections that the board can support but because of the placement of the NFC reader board these connections will not be made. During testing the NFC board can instead of being placed under the robot be situated on top of the development board in the Arduino connector. The nucleo connections can in addition to the first lab be seen in the following Figure 47.

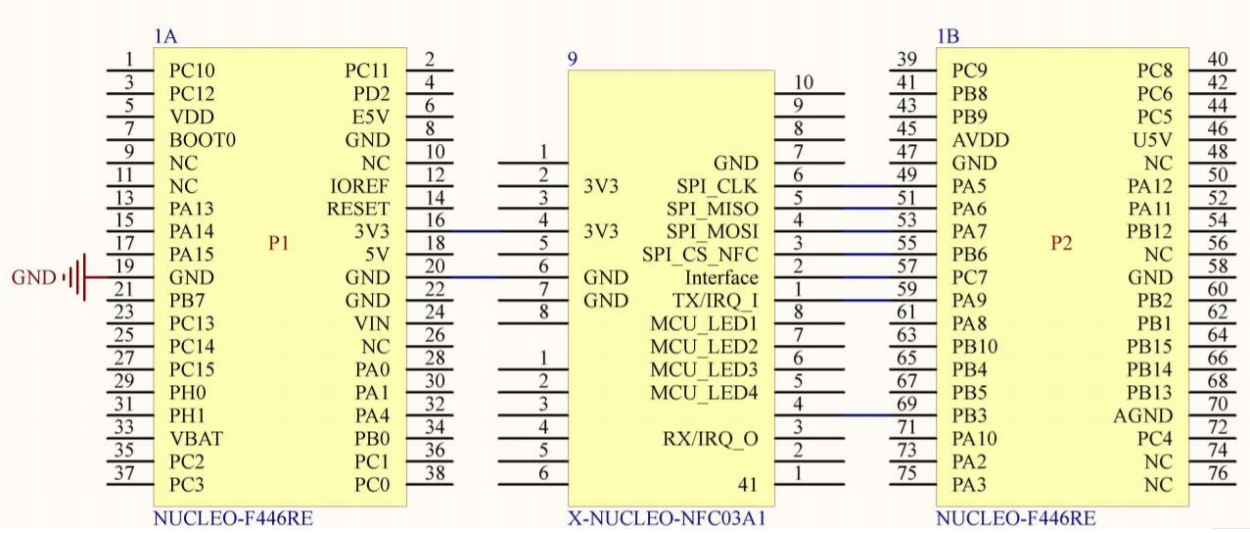

Figure 47 NFC connection

3.7.2 Configuring in STM32CubeMX

To configure first the LED on pin PA5 has to be reset to allow that pin to be used for SPI. Having used UART and I2C already, connection to the NFC reader will be made over SPI.

Under the Connectivity tab in STM32CubeMX SPI1 will be used is Full-duplex mode with the parameters seen in Figure 48.

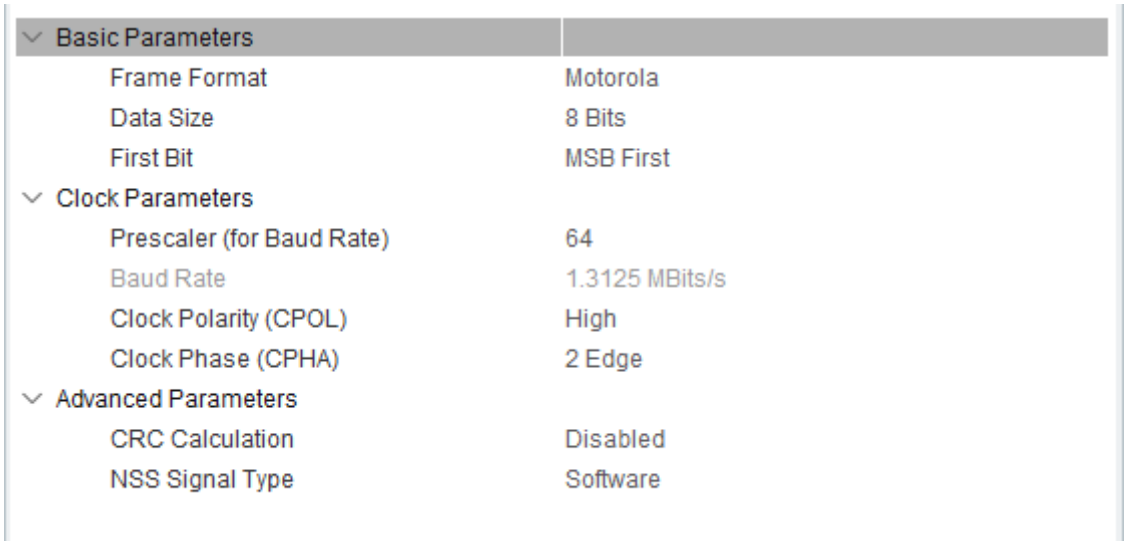

Figure 48. SPI parameters

2 new timers will be defined for the NFC module. One will be used to create a new delay with a smaller range of a microsecond, another will be used to create a timeout period

which functions to sense if communication between the development board and the 95HF (NFC reader/transceiver chip) chip still is active.

3.7.3 Program

The program code is based on the STM MyLiberty example code. The NFC module is treated as an external library and is in the driver's folder. It composes of code for the CR95HF chip (which is the chip that the NFC reader is based on), modules for communicating different types of NFC tag types and NDEF (NFC data exchange format) library for tag structures.

Mifare classic tags are not covered by the code but they can be detected as type 2 tags. These classic tags require extra steps for communicating.

There are similarities between tag types and their codes as such detecting, reading and writing from tag type 2 will be used as an example [29].

The code adds new timers using interrupts to guarantee communication timings between the microcontroller and the 95HF device. Initialization functions have been redone with the STM32CubeMX to ensure consistency between all modules.
Detecting a tag requires calling the *ConfigManager_TagHunting* function with a desired parameter (for example TRACK_ALL) for all tag types. The function returns the found tag type, which it searches for type by type.
If a tag is detected the reader is initialized for the specific tag and data can be read and written from it. Reading is done by calling the found tag types read function *PCDNFCTx_ReadNDEF* with no parameters, where x is the tag type 1-5.
Source code can be found from a GitLab project [21].

3.8 RTOS

This step adds a RTOS to the system. The used operating system is CMSIS-RTOS (cortex microcontroller software interface standard), which consists of a STM provided wrapper for FreeRTOS (open source RTOS).

FreeRTOS provides a real-time kernel for deeply embedded real-time applications using microcontrollers to allow the created systems to meet hard real-time requirements.

Using an operating system will also simplify the programming process for example the previously created delay module, will become almost completely replaced by semaphores. Implementing a delay in a function will be done using the operating system instead. When calling an operating system-based delay, the operating system instead of blocking and waiting will automatically switch over to another thread.

Decision to which thread to switch will be done by the scheduler.

3.8.1 Threads

Applications are setup so that they consist of independently executing threads. Because the used microcontroller only has one core, the threads can only be executed one at a time. The kernel will decide the thread execution order based on priority so that hard real-time threads are ensured execution time ahead of soft real-time threads. This means that modules that require more frequent calls are called more often. This in effect also reduces power usage. See Figure 49 versus Figure 50 [30].

An operating system will replace the while loop that up till now ran everything. An operating system will switch between threads if necessary, whereas a non-operating system-based system may lose important time executive mundane functions before reaching a critical function.

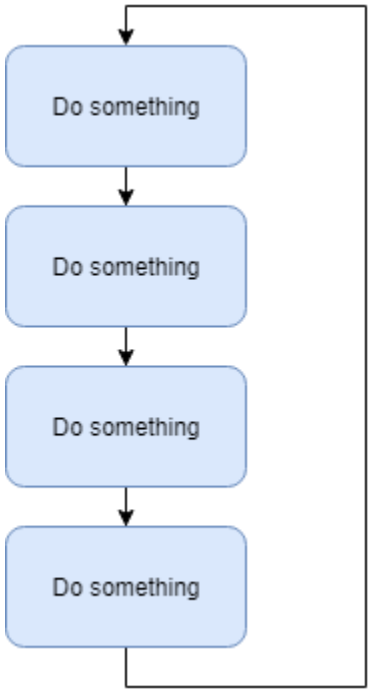

Figure 49. Non threaded example

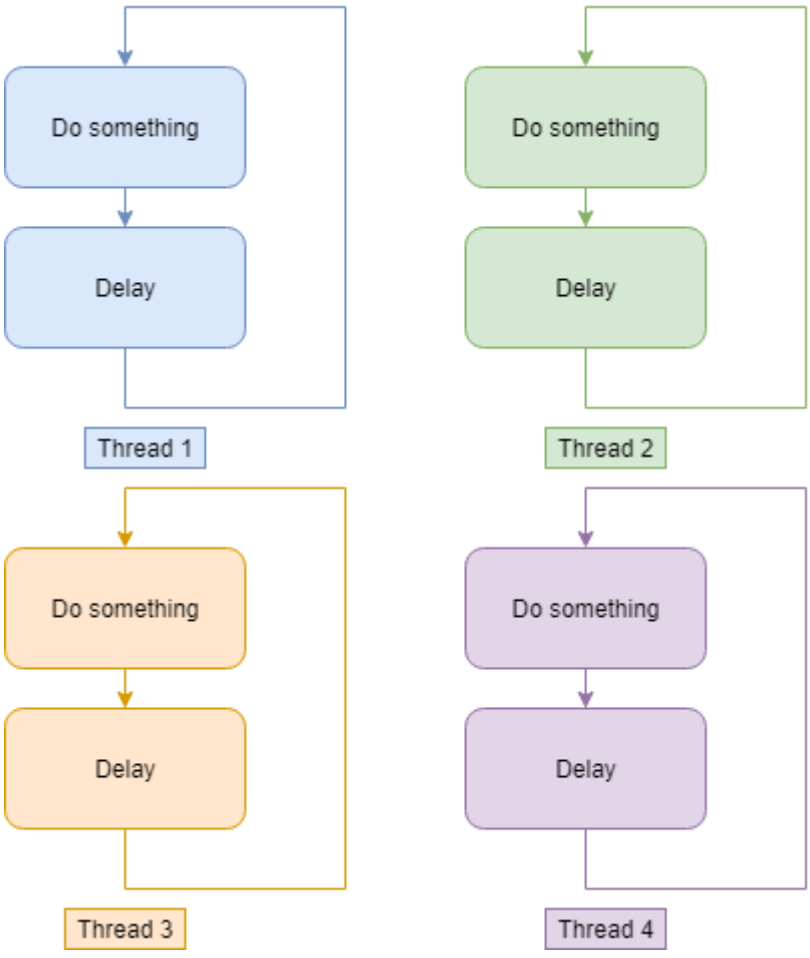

Figure 50. Thread example

3.8.2 Semaphores

In Figure 51 thread 2 is waiting for a semaphore that is given by an interrupt. Because the semaphore is not available yet, the thread blocks further running until the semaphore is given by the interrupt during the execution of another task. When the semaphore is given the thread waiting on it is executed as soon as the ISR (interrupt service routine) ends and then priority is given back to the originally running thread [30].

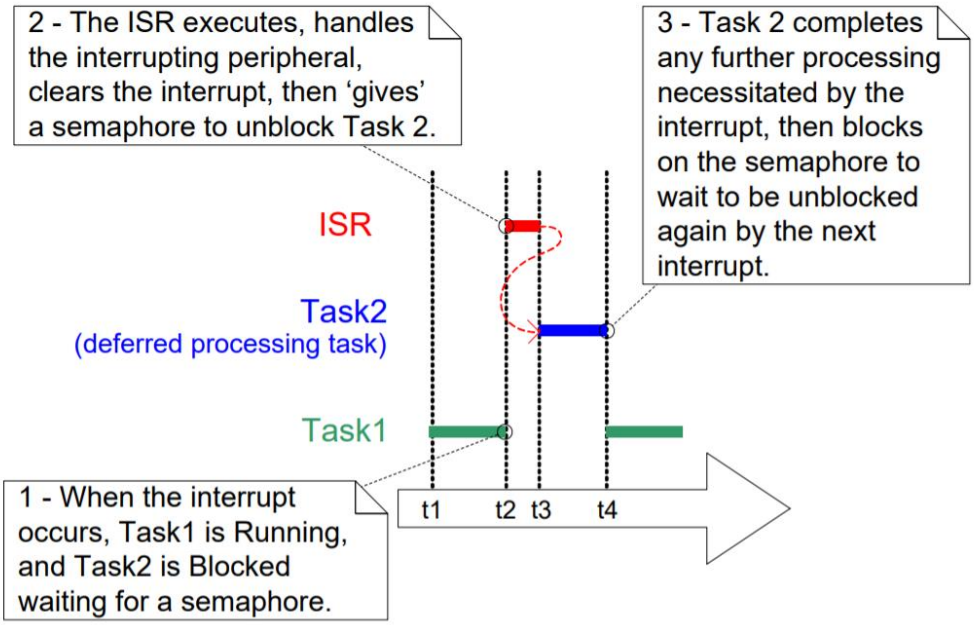

Figure 51. Binary semaphore with interrupt [30]

3.8.3 Queues

Queues allow data to form as the name says queues which can then be taken out one by one and used. The advantage of queues is that instead of disrupting an important task with doing something menial, the tasks can be put into a queue and done at another opportune time. Queues can be accessed by more than one thread, meaning that all non-important tasks can be called instead later. Example of queue insertion can be seen from Figure 52 where three higher priority tasks need to quickly finish what they are doing insert data to a queue and from Figure 53 a lower priority task takes the data from the queue and does the tasks instead after higher priority tasks have finished

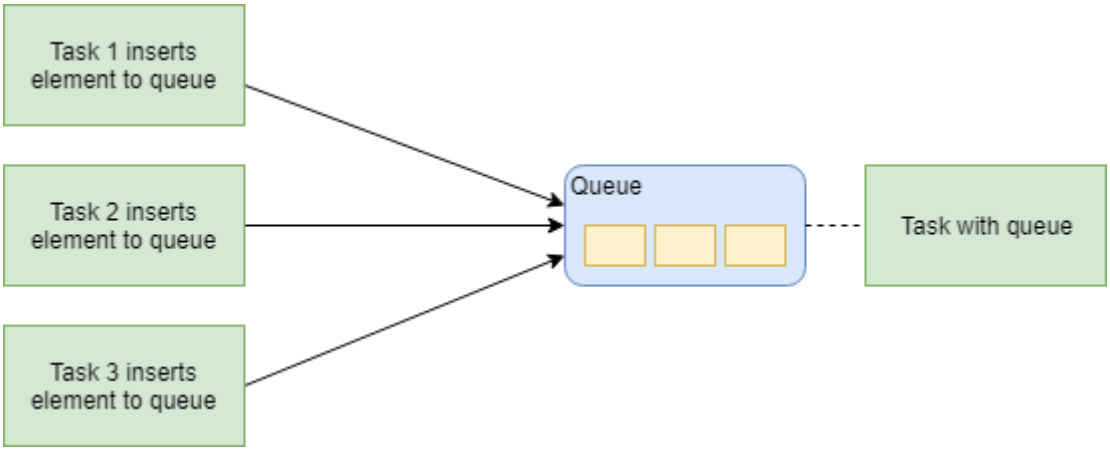

Figure 52. Insert to queue

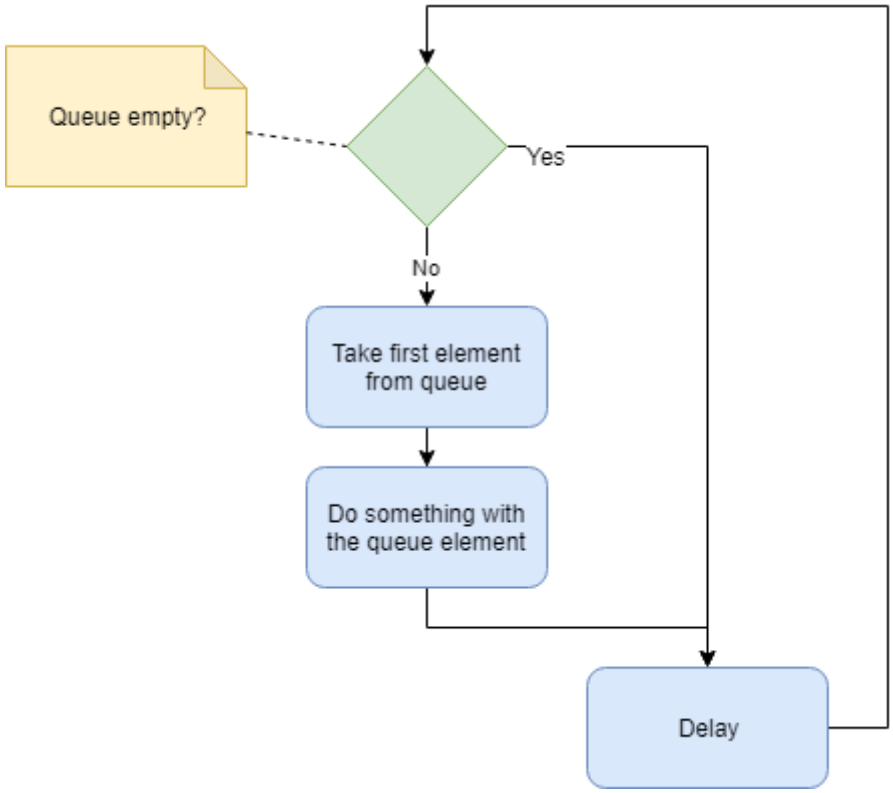

Figure 53. Take from queue

3.8.4 Configuring in STM32CubeMX

Adding an operating system can be done also with the STM32CubeMX software. Adding an operating system is done under the middleware tab by enabling FreeRTOS. Most of the config parameter setting can left as default. For debugging purposes enabling *malloc* and checking for stack overflow should be enabled.

This will help to identify problems that may arise – such as when trying to print floating point numbers (which by nature are not thread safe) and threads causing a stack overflow when they have too little stack assigned to them.

Enabling these two hooks will create two respective call-back functions which can be used as marking points that when the program enters them it is obvious that something has gone wrong.

Memory management heap_4 is used. This means that each thread, queue is statically declared and dimensioned by the TOTAL_HEAP_SIEZ parameter in an array. This makes it look like the code uses up a lot of memory before the system is even run. This allow however to limit the amount of memory used also, because a thread cannot in FreeRTOS use up more than what was allocated to it. If a thread does run out of memory it will cause a stack overflow. On Figure 54 5 threads can be seen allocated in memory in the described way [30].

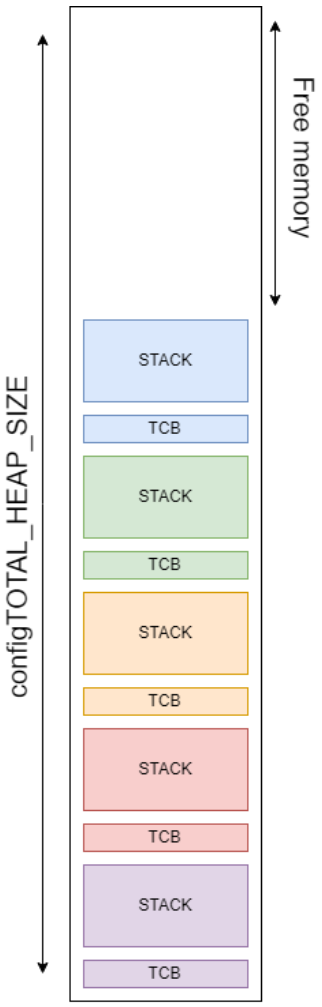

Figure 54. RTOS memory management

TCB (task control block) stores information about whether the stack for the attached thread is running.

The stack overflow has to be handled by the user, for example setting the stack size bigger or re-allocating memory or changing the memory management scheme.

3.8.5 Delay code

The delay function *osDelay* waits for approximately the specified number of milliseconds. This function makes the custom delay obsolete. In addition to providing a non-blocking delay to the system the *osDelay* function allows for threads to be switched during the delay time. The function should be called also at the end of every thread. Calling for a delay ensures that only one thread won't be stuck running. The delay time assigned at the end of each thread allow to in conjunction with a threads priority to manage how often a thread needs to be called. An example is from Figure 55.

```
void StartTaskNFC(void const * argument) {
  /* USER CODE BEGIN StartTaskNFC */
       /* Infinite loop */
       for (;;) {
              NFC_cyclic();
              osDelay(100);
       }
  /* USER CODE END StartTaskNFC */
}
```

Figure 55. Thread with osDelay

The example thread ends with a call to the *osDelay* with 100 as the specified time length. This allows to check for nearby tags 10 times per second. If the movement speed of the robot is increased the delay can be made smaller so that a tag won't be missed accidentally.

3.8.6 Semaphore code

Semaphores are implemented in such a way that when one thread requires something it won't use a Boolean type check for example but rather a semaphore. A semaphore example can be seen from Figure 56.

```
void HCSR04_cyclic() {
	HCSR04_measure();
	if (xSemaphoreTake(HCSRechoSemHandle, 40 / portTICK_PERIOD_MS))) {
	…
	}
}
```

Figure 56. Semaphore example

In this example after initiating a measurement processes for the ultrasonic module a check for a semaphore takes place. This semaphore is set by the falling edge of the ultrasonic modules signal. The signals maximum length is 38ms if it doesn't detect any obstacles. So, the semaphore will block the thread for up to 40ms as to make sure a result is received. After the semaphore is received normal process in the if clause follows. The example follows the same principle as described in the Queues paragraph.

Another way to solve the same problem could be by setting the semaphore blocking delay to 0 and adding a delay function call before checking for the semaphore like in Figure 57.

```
void HCSR04_cyclic() {
	HCSR04_measure();
	osDelay(40);
	if (xSemaphoreTake(HCSRechoSemHandle, 0)) {

	…
	}
}
```

Figure 57. Semaphore and osDelay

The idea remains the same, but in this case the delay will always be 40ms and the interrupt being set can't return the program to this thread, making the program essentially slower.

3.8.7 Queue code

A queue is used exactly in the way that was described previously. By having tasks that should finish what they are doing quickly put information in the queue that is not so important. The queue used is for outputting data to the LCD. The LCD thread being set as the lowest priority will only run when nothing else needs to run. This allows for multiple items to be entered into the queue before the thread is ran. Other threads besides the LCD thread will give data to the queue and when the LCD thread runs it will take things out and output them to the display. A sending data example can be seen from Figure 58.

```
strcpy(g_msg.msg, dataOut);
g_msg.pos = MSG_POS;
g_msg.clear = MSG_LEN;
xQueueSend(LCDQueueHandle, &g_msg, 0);
```

Figure 58. Sending to LCD queue

The value to be sent to the queue is set up in a struct that contains values for where to print the desired message, clear length, and the message string.

Once the LCD thread runs a call to the *xQueueReceive* function checks for information in the queue and stores it to a local variable if there is any and then the data can be used as desired.

3.9 WiFi

This step is adding a WiFi module for communication with an external server. The developed robot is going to be mobile, so it needs to have some sort of a communication with the outside world if we want it to act on new instructions on the fly. The NFC module would work but because of the limited range and the limited information carrying capabilities of NFC tags, communication would be very slow and tedious. Using WiFi it is possible to easily retrieve data from the outside environment.

In this case the robot will be communicating with a local WIFI network which will provide information to the robot.

3.9.1 Hardware connection

Connecting the WIFI module requires only 2 connections to the board, these are RX and TX. Connection scheme is depicted on Figure 59. The 3V3 VCC connection should be an external power source to ensure the stability of the WIFI module.

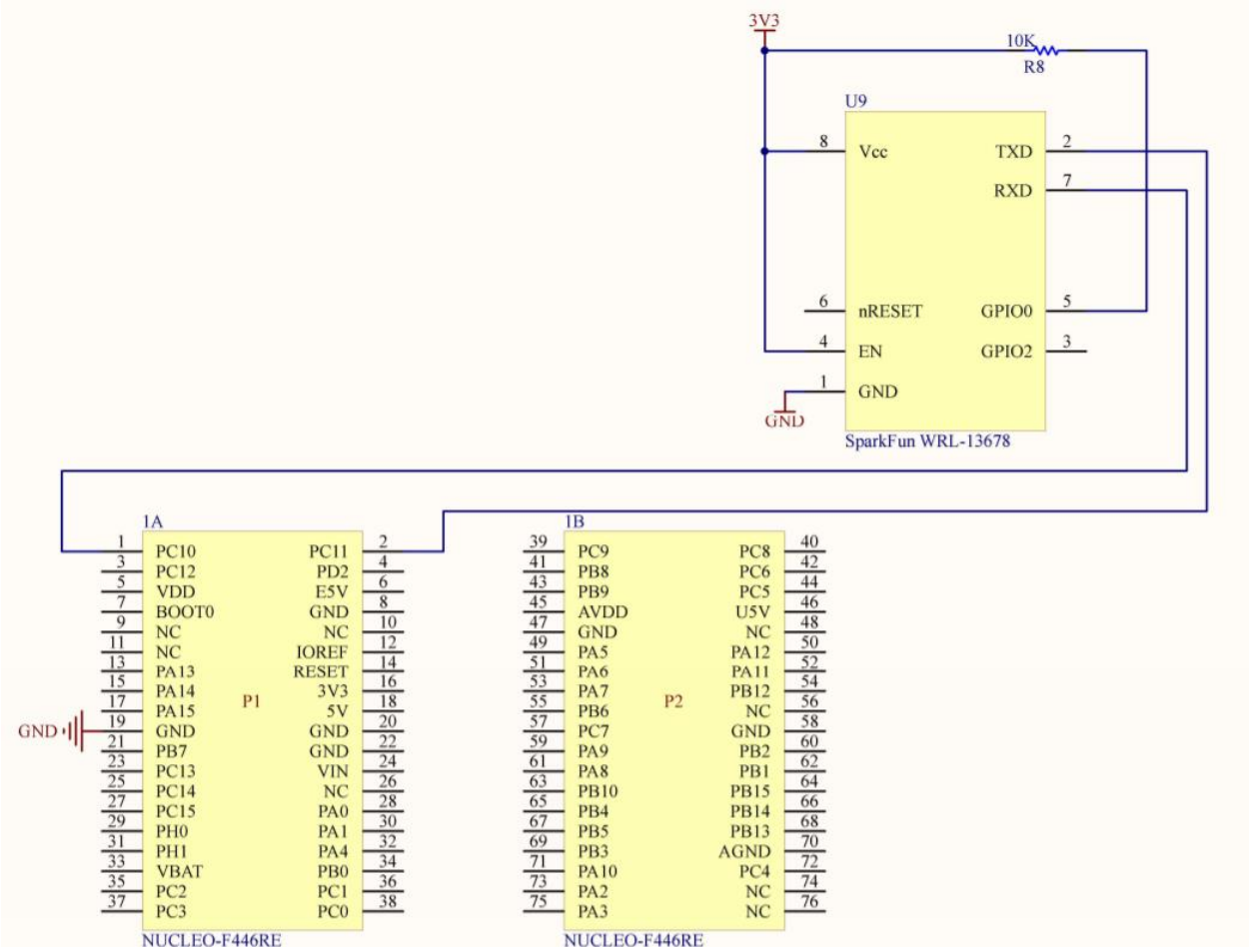

Figure 59. WiFi connection

3.9.2 Configuring in STM32CubeMX

To add the WIFI module to the project only a new UART connection has to be added from under the connectivity tab. USART3 in Asynchronous mode with default parameter settings will be used.

3.9.3 Program

The WIFI module is made to work as a client for current purposes – the module will connect to an existing WIFI network which will host a small server. The server can be sent info and based on sent information a response is sent back.

The WIFI modules workflow is set up to only connect once to the WIFI network - this is done in the initialization stage, after which the workflow is divided to three states: idle, waiting and working. State workflow is depicted on Figure 60.

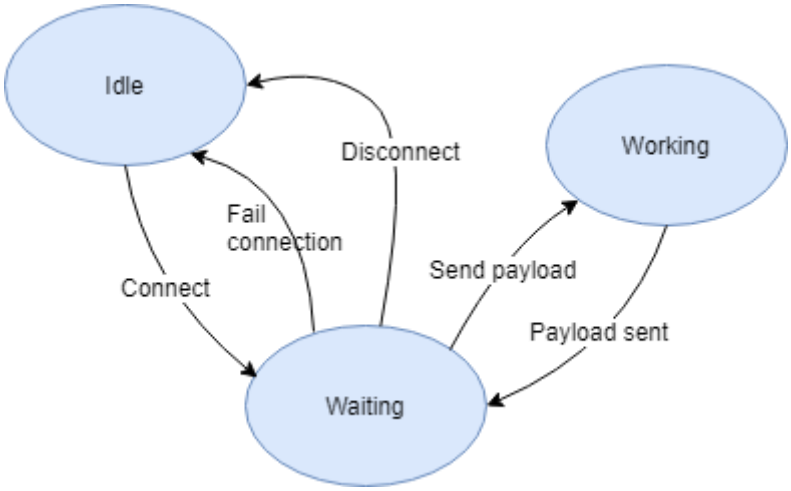

Figure 60.Wifi workflow

After initialization if a successful connection is made to a specified IP (internet protocol) address the WIFI module will wait for further instructions what to send over the established connection.

Specific request can be made using the *esp8266_sendPayload* function, which sends a TCP (transmission control protocol) payload to the connection. Once a payload is sent and a response, which can be sent at any time, is received it can be searched processed.

The received response should contain a specifically formatted string, that will contain information meant for the robot. The information will contain either coordinates where the robot should move, or whether the last sent payload contained correct information.

3.10 Workflow

The full workflow and operations of the robot were implemented gradually over all previous paragraphs. The ideal workflow can be seen from Figure 61. The robot will ask for information from the server, based on received coordinates calculates angle to the tag, turns towards the tag and starts driving. With the encoder coordinates are calculated to position the robot on the field. When the position is reached the robot will try to find the

NFC tag, if no tag is found small movements are made to try and position the robot better. Once the tag is found information is read from it and sent to the server. If the server responds with the next tag coordinates the process is repeated.

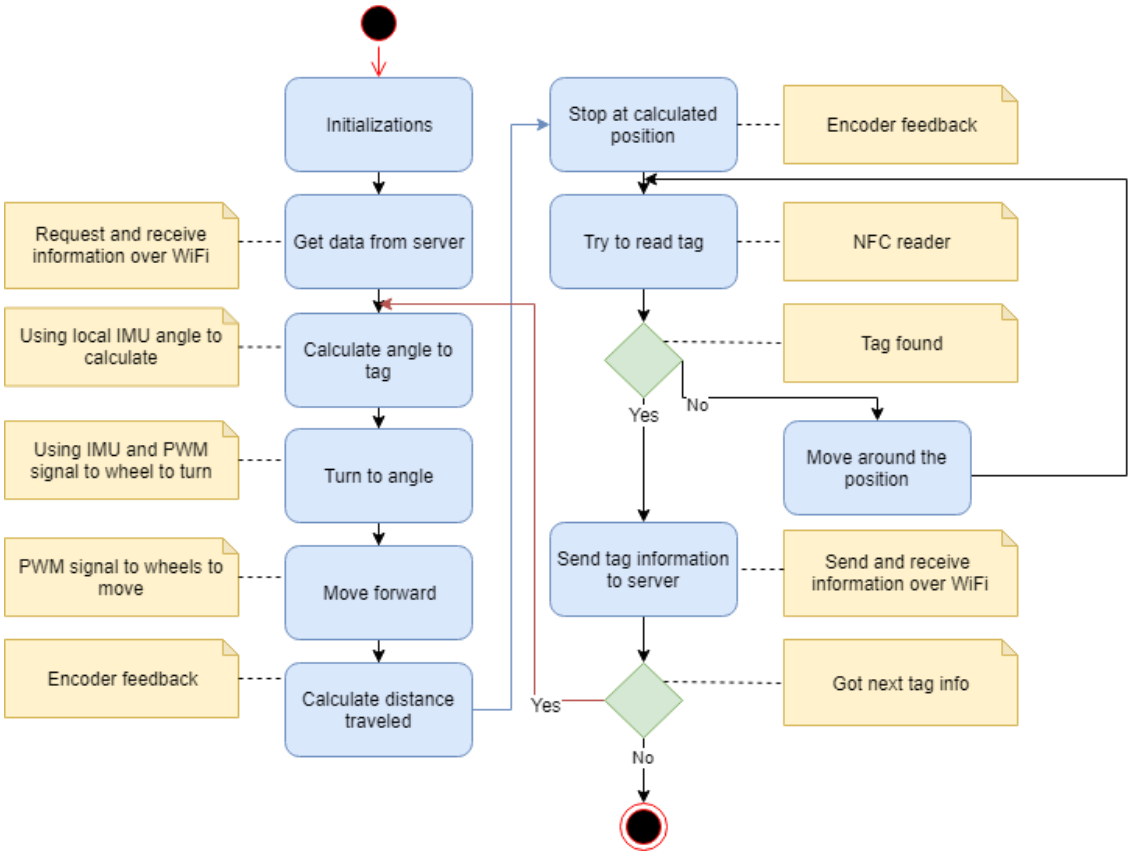

Figure 61. Robot workflow

In case the server responds with the previous tag coordinates and the robot coordinates are correct the robot will instead try to calculate its position using the ultrasonic sensor by measuring distance to the walls. If that happens, movement will happen instead not straight towards the next tag but along the x and y axes.

If the server however responds with a finished message the workflow ends as all tags have been found.

3.11 Server

The last step involves setting up the local server. The server was created in Python using Flask and running a small SQLite database.

The server was created as a webserver so that the request format can also be tested without a working system and just sending commands through the browser.

The workflow for the server is depicted in Figure 62.

The database that the server interfaces with is set up as seen from Table 2. The database has a start and end time column to time the whole process from asking the first tag location to sending the last correct response. X and Y columns are for 2 dimensional coordinates of the field where the robot operates, and the answer is the string that is written to an NFC tag.

Table 2. Database structure with example information

Start time	End time	X	Y	Answer
		56	39	test_str1
		89	94	test_str2
		43	57	test_str3
		64	30	test_str4
		10	48	test_str5

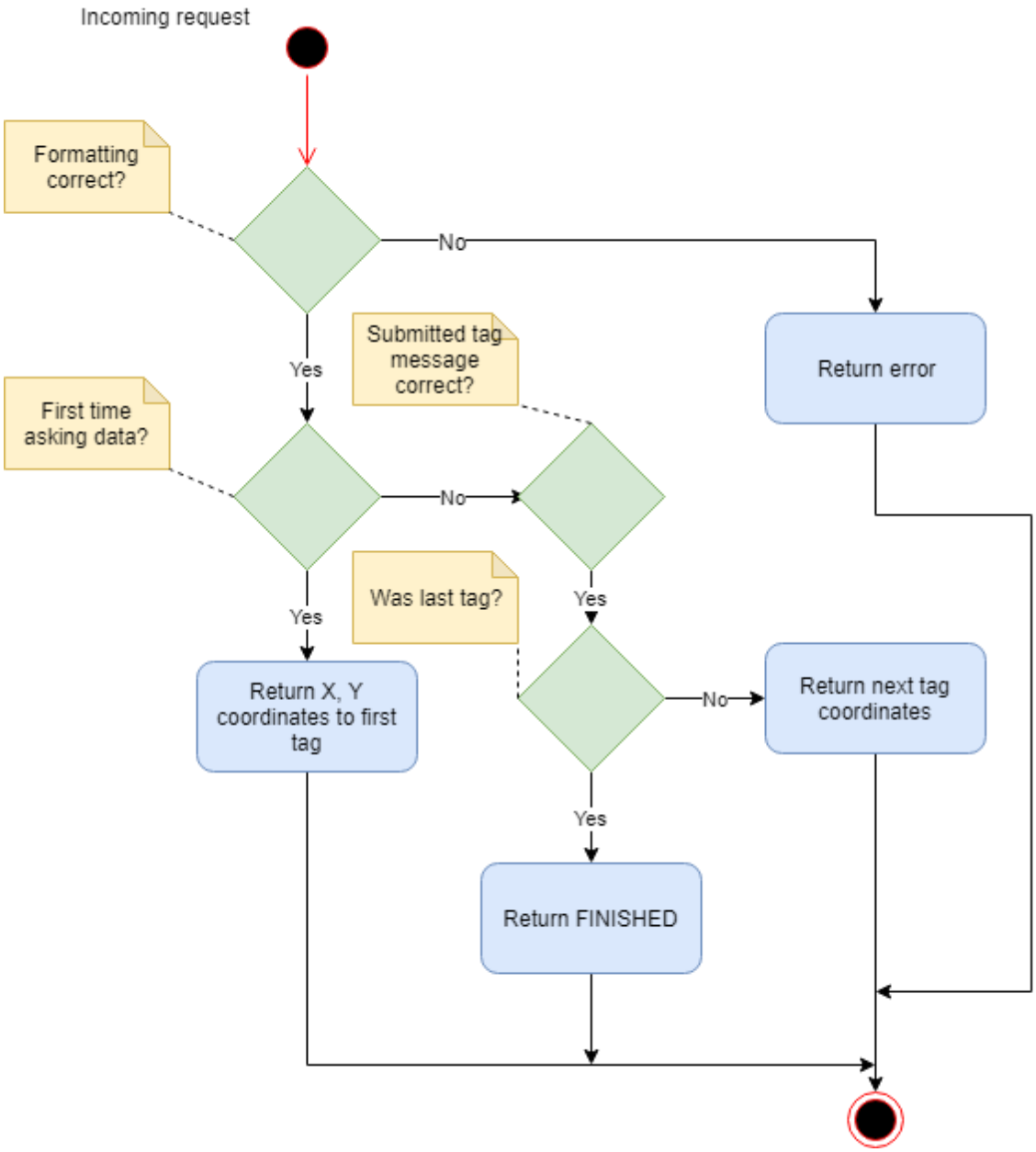

Figure 62. Server function scheme

The server has three calls that can be made:

1. Index page, which will return a message how to ask information (2)
2. Initial call to /robot/ID, which will return the first tags location, and starts a time if it is not already started.
3. Following calls to /robot/ID/tagNumber/tagMessage

Call order starting from the index page, getting first tag location, getting second tag location and finishing is shown on Figure 63.

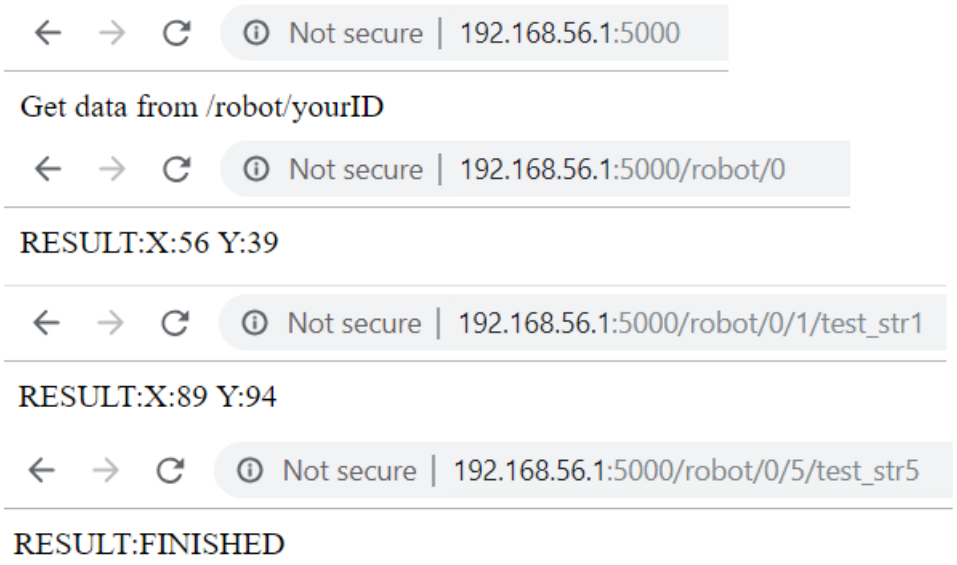

Figure 63. Server call and answer example

3.12 Result

The end result is a robot that asks a NFC tag location from the local server over WiFi, moves to the location, reads the NFC tag, sends the result to the server, waits, for a response then moves to the next received tag location until a finished message is received. Version one of the robot can be seen from Figure 64.

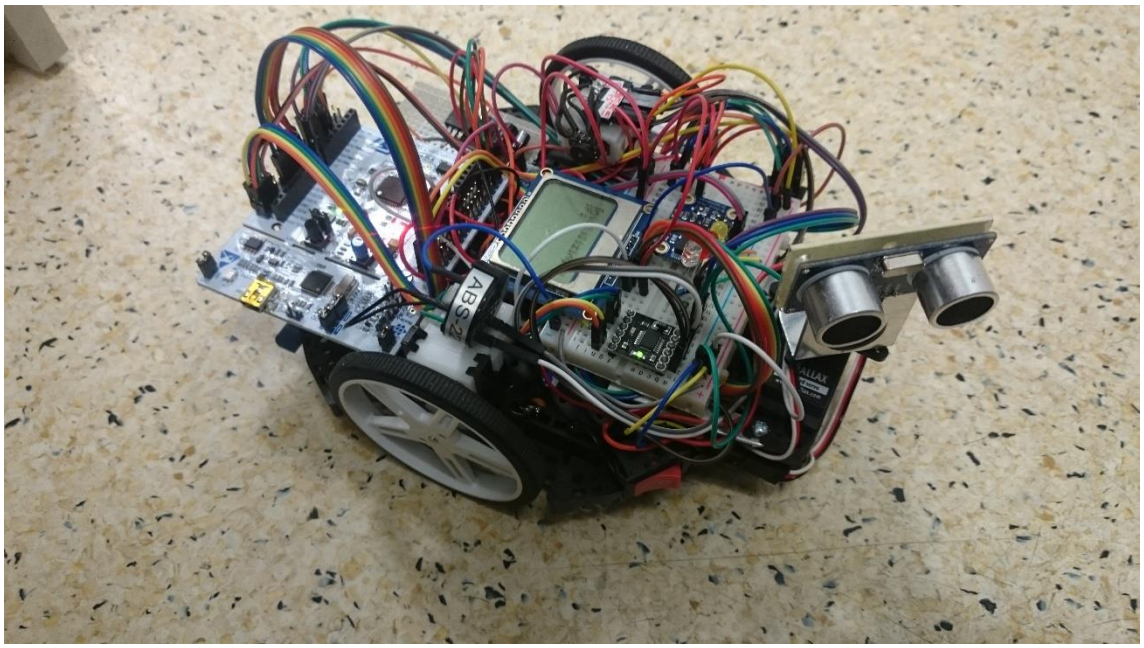

Figure 64. Lab robot

The robot is able to blink LEDs to indicate what it is doing at a time. The LEDs are used to indicate an ongoing NFC tag search and if a tag is found a combination of LEDs light up based on the tag type that was found. In addition to LEDs the robot can give feedback to the user using a speaker or an LCD screen.

The robot can drive around using DC motors which are driven by a H-bridge driver controller so that it can drive the wheels both forward and backward. The driver controller is connected to the robot via an encoder which gives feedback about the motor speed. This allows for more precise control over the direction and rotation speed manipulation using PWM. The robot can turn to a certain angle using data from the IMU. The IMU can also provide rough information about whether the robot is moving. The angle can be based off magnetic north or local heading.

To detect objects the ultrasonic sensor can be used. The sensor can also be used for basic navigation – measuring distances from the two walls the location can be calculated.

The robot is capable of wireless communication. Communication over WiFi allows to get information from a server and NFC allows to read information from tags.

3.12.1 Project

Students taking the embedded systems course will learn in the form of a project to put together a similar system as described in previous paragraphs. The timeline of getting acquainted, developing the system, testing and presenting it based on the 16-week semester can be seen from Table 3.

Students are given initially some time to get adjusted to embedded systems and figure out their teams for the project. After which they have to start already working on their projects as expected per Table 3 figuring out how to make the robot move and adding navigation capabilities as weeks go by. During the second half of the course for the labs the navigation system should be somewhat done, and students ought to start working on how to read from NFC tags, implementing required functionality on FreeRTOS and starting to establish communication between the robot and the server.

By weeks 14 and 15 all functionality should be implemented at which point the students systems are tested against each other. The whole system should work and tested to accomplish the task. The last week is the week for presenting the work done.

Table 3. Course lab timeline

Week	Topic	Description
1	Introductory	Students are introduced to the development environments. Students start by lighting up LEDs and getting and adding external ones as a warmup in the first week. Teams are formed.
2	Introductory	
3	Driving motor	Robot is made to move with DC motors using external power.
4	Non-blocking functions	To ensure that the robot can be used for more than one thing at a time, students have to implement a timer-based delay.
5	Ultrasonic sensor	Students start to implement navigation for the robot by adding another servo with an ultrasonic sensor. Create navigation logic with data from the sensor.
6		
7	IMU	To further improve navigation an IMU is to be added. This will help with maintaining a course and give information which way the robot is facing. To maintain a course a simple PID controller has to be implemented.
8	Feedback	So far, the robot moves but the user gets little feedback while it's running. An LCD and a speaker are added to give feedback to the user.
9	Encoder Feedback	Final step in adding navigation to the robot. Using motor feedback students have to be able to locate where the robot is on the field.
10	NFC	After navigation has been completed NFC reader functionality has to be implemented.
11	RTOS	After students have acquainted themselves with basics embedded system concepts, they have to move on to working with an operating system. Converting created functionality to work on FreeRTOS
12		
13	WiFi	Once the robot works with test data, WiFi can be added to get real data over WiFi from the server.
14	Building navigation	Time to interface all modularity if necessary, debug, and test functionality.
15	Presentation/ Result	Presenting work done. Timing robot runs.
16	Extra time	In case something happens, and weeks must be shifted.

4 Summary

The main goal of this thesis was achieved. Hands-on exercises and system for students were created to use in TalTech Embedded Systems course. The developed system for the course is made in a way that when students' progress through the course they keep building on top of the previously completed tasks. During the new course students can create a more complicated system. And since all students are given the same task the grading process becomes easier than when all students developing their own different systems.

The problem of solutions being found on the internet is also being solved by default with just developing new exercises.

For the new exercises development cycle from an idea, to research, development and testing was done. The idea was specified task by task to cover a certain aspect about embedded systems. The tasks covered ranged from blinking simple LEDs as introduction, driving DC motors, creating and using interrupts to using different sensors and implementing an RTOS for the system. In addition to working on individual modules, the system as a whole also has to be capable of navigation in a specified area.

The tasks for the students a server for the system to interact with and a more complete step by step guide was put together as an instructional material for the lecturers which can be found from along with the code from GitLab [21].

The developed system can and most likely will be improved upon before being presented to students. For this it will go through more testing and restructuring if necessary.

5 References

[1] J. Valvano and R. Yerraballi, “Embedded Systems - Shape The World,” [Online]. Available: http://users.ece.utexas.edu/~valvano/Volume1/E-Book/. [Accessed 30 04 2019].

[2] Berkeley - university of California, “Berkeley - Embedded Systems,” [Online]. Available: https://bcourses.berkeley.edu/courses/1464526. [Accessed 30 04 2019].

[3] ETH zürich, “ETH zürich - Embedded Systems,” [Online]. Available: https://www.tec.ee.ethz.ch/education/lectures/embedded-systems.html. [Accessed 30 04 2019].

[4] Embedded related, “Embedded related - Introduction to Microcontrollers,” [Online]. Available: https://www.embeddedrelated.com/showarticle/453.php. [Accessed 30 04 2019].

[5] C. Noviello, Mastering STM32, Leanpub, 2018.

[6] G. Brown, Discovering the STM32 Microcontroller, 2016.

[7] L. E. Carlson and J. F. Sullivan, “Hands-on Engineering: Learning by Doing in the Integrated Teaching and Learning Program,” *The International Journal of ENGINEERING EDUCATION,* vol. 15, no. 1, pp. 20-31, 1999.

[8] C. E. Wieman, “Large-scale comparison of science teaching methods sends clear message.,” *Proceedings of the National Academy of Sciences of the United States of Americ,* vol. 111, no. 23, pp. 8319-8320, 2014.

[9] L. C. Scavarda-do-Carmo and M. A. da Silveira, “Sequential and Concurrent Teaching:,” *IEEE Transactions on Education,* vol. 42, no. 2, pp. 103-108, 1999.

[10] P. C. Blumenfeld, E. Soloway, R. S. Marx and J. S. Krajcik, “Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning.,” *Educational Psychologist,* vol. 26, no. 3/4, pp. 369-398, 1991.

[11] M. J. Prince and R. M. Felder, “Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases,” *Journal of Engineering Enducation,* vol. 95, no. 2, pp. 123-138, 2006.

[12] J. E. Mills, D. F. Treagust, N. Scott, R. Hadgraft and V. Ilic, “ENGINEERING EDUCATION – IS PROBLEM-BASED OR PROJECT-BASED LEARNING THEANSWER?,” *AUSTRALASIAN JOURNAL OF ENGINEERING EDUCATION.*

[13] Adafruit, “IMU comparision,” [Online]. Available: https://learn.adafruit.com/comparing-gyroscope-datasheets. [Accessed 01 05 2019].

[14] STMicrolectronics, “STM32CubeMx,” [Online]. Available: https://www.st.com/en/development-tools/stm32cubemx.html. [Accessed 01 05 2019].

[15] Atollic, “Atollic TrueSTUDIO,” [Online]. Available: https://atollic.com/truestudio/. [Accessed 01 05 2019].

[16] Intronix, “INTRONIX PC-BASED Test and Meaurement,” [Online]. Available: https://www.pctestinstruments.com/. [Accessed 01 05 2019].

[17] Python Software Foundation, “Python homepage,” [Online]. Available: https://www.python.org/. [Accessed 01 05 2019].

[18] The Pallets Projects, “The Pallets Projects - Flask,” [Online]. Available: https://palletsprojects.com/p/flask/. [Accessed 01 05 2019].

[19] The SQLite Consortium, “SQLite,” [Online]. Available: https://www.sqlite.org/index.html. [Accessed 01 05 2019].

[20] STMicroelectronics, “STM32 Nucleo-64 boards user manual,” [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf. [Accessed 08 05 2019].

[21] H. Juhanson, “GitLab,” [Online]. Available: https://gitlab.pld.ttu.ee/bes/master-2019/stm32f446.

[22] Texas Instruments, “DRV8833 datasheet,” [Online]. Available: http://www.ti.com/lit/ds/symlink/drv8833.pdf. [Accessed 02 05 2019].

[23] Parallax, “Parallax standard servo,” [Online]. Available: https://www.parallax.com/sites/default/files/downloads/900-00005-Standard-Servo-Product-Documentation-v2.2.pdf. [Accessed 02 05 2019].

[24] Oomipood, “Oomipood HC-SR04,” [Online]. Available: http://data.oomipood.ee/kasutusjuhend/arduino/HC-SR04.pdf?lang=en. [Accessed 02 05 2019].

[25] Bosch, “BNO055 datasheet,” [Online]. Available: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BNO055-DS000.pdf. [Accessed 08 05 2019].

[26] Philips, “Adafruit LCD datasheet,” [Online]. Available: https://cdn-shop.adafruit.com/datasheets/pcd8544.pdf. [Accessed 02 05 2019].

[27] Farnell, “Farnell - Speaker,” [Online]. Available: http://www.farnell.com/datasheets/1768515.pdf. [Accessed 02 05 2019].

[28] NFC forum, “NFC in action,” [Online]. Available: https://nfc-forum.org/what-is-nfc/nfc-in-action/. [Accessed 02 05 2019].

[29] STM NFC, “STM X-CUBE-NFC3,” [Online]. Available: https://www.st.com/content/st_com/en/products/embedded-software/st25-nfc-rfid-software/x-cube-nfc3.html. [Accessed 02 05 2019].

[30] R. Barry, “Mastering the FreeRTOS™,” [Online]. Available: https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf. [Accessed 03 05 2019].

Appendix 1 – NFC tags