**Swarmbot system**

**Document**

A group of vehicles in a row

Description automatically generated

Kiet Hô, Tom Koppelmans

|  |  |  |
| --- | --- | --- |
| Version | Date | Remark |
| 1.0 | 12-11-2023 | First maintained version |
| 2.0 | 15-11-2023 | Update the design, implementation and test of phase 1 |
|  |  |  |
|  |  |  |

Table of Contents

[Introduction 3](#_Toc154003332)

[phase 1 4](#_Toc154003333)

[Design 4](#_Toc154003334)

[Implementation 6](#_Toc154003335)

[Test 10](#_Toc154003336)

[Reflection 11](#_Toc154003337)

[Phase 1 Reflection: 11](#_Toc154003338)

[Personal reflection 11](#_Toc154003339)

[phase 2 12](#_Toc154003340)

[Design 12](#_Toc154003341)

[Timer 12](#_Toc154003342)

[360 servo 13](#_Toc154003343)

[Ultrasonic 13](#_Toc154003344)

[Robot 14](#_Toc154003345)

[PID control 16](#_Toc154003346)

[state machine 17](#_Toc154003347)

[Implementation 18](#_Toc154003348)

[PID 18](#_Toc154003349)

[Integral 19](#_Toc154003350)

[tuning 19](#_Toc154003351)

[Test 20](#_Toc154003352)

[Ultrasonic 20](#_Toc154003353)

[Servo 20](#_Toc154003354)

[reflection 20](#_Toc154003355)

[phase 3 21](#_Toc154003356)

[Design 21](#_Toc154003357)

[Implementation 21](#_Toc154003358)

[Test 21](#_Toc154003359)

[reflection 21](#_Toc154003360)

[Bibliography 22](#_Toc154003361)

# Introduction

This documentation is about the phases of our car project, using servo motors and an ultrasonic sensor.

Throughout this document, there is written about hardware connections, software protocols, implementation strategies, and testing outcomes. The first phase sets the groundwork for coming stages that will make the system's capabilities better.

# phase 1

## Design

**System overview**

The diagram below shows a high level of how the components in the robot talk with each other. There are simply 3 components which are ultrasonic, and 2 servo. The MCU board would send a trigger and receive the echo signal indicating the distance to the obstacle. For the servo, it send the PWM signal with indicate the speed of them.

A diagram of a nuclear reactor

Description automatically generated

**Figure 1. system context diagram**

**Hardware installation**

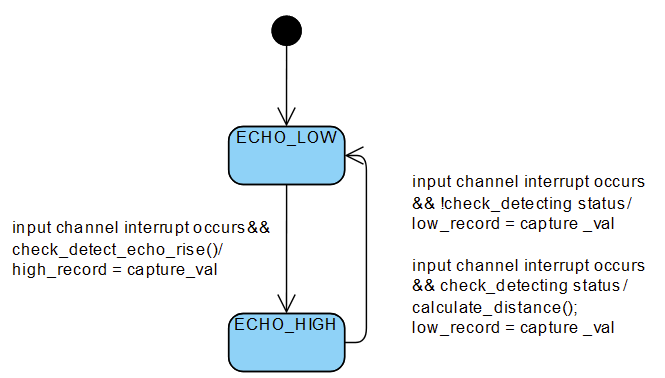
|  |  |
| --- | --- |
| For phase 1 we setup the 2 servo’s and 1 ultra-sonic Sensor as shown in the picture. The servo’s use PA5 and PA1 with both on timer 2, but PA5 on channel 1 and PA1 on channel 2.  The ultra-sonic Sensor uses PB8 for the Echo and PB5 for the trigger. The Echo is set to use timer 4 on channel 3 and the trigger is set to use timer 3 on channel 2. | A circuit board with wires and wires  Description automatically generated  **Figure 2. schematic of robot system** |

|  |  |  |
| --- | --- | --- |
| Protocol | Component and pin | Component and pin |
| PWM | Feedback 360 Servo 1 - signal - Right | NucleoF303RE – PA5 (D13) |
| PWM | Feedback 360 Servo 2 – signal - Left | NucleoF303RE – PA1 (A1) |
| PWM | Ultrasonic – Trigger | NucleoF303RE – PB5 (D4) |
| PWM | Ultrasonic - Echo | NucleoF303RE – PB8 (D15) |
| Connection | Feedback 360 Servo 1 - VCC | NucleoF303RE – VIN |
| Connection | Feedback 360 Servo 2 - VCC | NucleoF303RE – VIN |
| Connection | Ultrasonic - VCC | NucleoF303RE – VIN |
| Connection | Feedback 360 Servo 1 – GND | NucleoF303RE – GND |
| Connection | Feedback 360 Servo 2 – GND | NucleoF303RE – GND |
| Connection | Ultrasonic – GND | NucleoF303RE – GND |

**Software**

**Ultrasonic**

To increase the precision in detecting the low and high echo signal, which determine the distance of the closet obstacle. A simple state machine has been implemented. **Figure 4.** There are 2 states in total. ECHO\_HIGH and ECHO\_LOW. The state machine starts at the ECHO\_LOW state and jumps to the ECHO\_HIGH only when the edge detecting occurs in the period between 400us and 500us. Other detecting time is considered as noise. In contrast, the ECHO\_HIGH jumps back to ECHO\_LOW by any edge detecting events. However, the distance is only calculated when the time does not exceed 38ms after the rising occurs.



**Figure 4. state machine of detecting rising and falling edge of echo signal**

**Robot**

The sate machine below simply show how robot react with the distance value receiving from the ultrasonic sensor. It would jump to TURNING, if the distance lower than 10cm that robot would turn randomly to a direction and jump back to AHEAD state, in which robot is going to move forward with the set speed, if the interval is over and the distance is longer than 10cm.

A diagram of a machine

Description automatically generated

**Figure 5. state machine of avoiding obstacle behavior**

## 

## Implementation

**Pusle width set calculation.**

To control both servo and Ultrasonic, it required to send a suitable pulse width for example with 10us to trigger the ultrasonic. our approach is to set the pulsed width following the rate of the whole period. Since the pulsed width is set by specifying the compare value on the channel with PWM output, the compared value is specified as the percentage of the max counting, max counting.

To do that, we simply use the function as in the **Figure 6** to set the compare value with the intention of generating right pulsed width. Since the auto reload value of all timer configurations is set to max of 16 bits, which indicates the length of the PWM period. Compare value would by a percentage times of of the 16 bits max – 65535.

**Figure 6.a set the compare value following the PWM percentage**

A diagram of a graph

Description automatically generated

**Figure 6. the pwm duty cycle**

**Distance calculation**

The data sheet of the ultrasonic is already documented using the converting formula to calculate the distance based on time. **Figure 1.c** This formula is indeed just based on the normal distance calculating formula. **Figure 1.a**

**S = V \* T**

**S**(m): the travelling distance of the object

**V**(m/s): the travelling velocity of the object

**T**(s): the travelling time of the object

**Figure 1.a the formula of the travelling distance of an object**

However, the distance must be measured in cm and the time measurements are following the mentioned mechanism and by applying the sound speed - 383 m/s. We getthe formula as in the figure. **Figure 1.b.** It needs to be mentioned that the travelling distance of sound is doubling the distance to the closet object, since the sound needs to travel to the object and echo back.. The Constance numbers has been round back to 158 as in **figure 1.c**

Distance (cm) =

**Figure 1.b   distance formula to the nearest object based on time velocity.**

****

**Figure 1.c Distance to the closest object based on the time interval. (datasheet)**

**Speed calculation**

Based on the datasheet of parallax 360 feedback servo, corresponding PWM signal must be sent to set the desired servo speed. the information of the pusle width range corresponds to the range of speed can be seen in the **Figure 8.a**. Apart from that the PWM period is 20ms.

Since the pulsed width is set as a percentage of the whole period in the PWM calculation. These ranges must be converted to percentage of the whole period. By applying the formula of **Figure 6.a** we can easily get the value of the pulsed width in the percentage of the whole period. **Figure 8.b**

Thanks to the method of calculating based on the percentage, we can easily specify the compare value on timer channel to get the desire speed through the duty cycle percentage. the counted number don’t have to be figured out exactly to get the corresponding pulse width. Otherwise, if the prescaler and auto reload value is changed, the counted number might has to be changed to get the same PWM pulsed. However, with calculating based on the percentage of the max counting, this is not necessarily.

The speed can be converted to the pulsed width percentage of the PWM period by using a map function. This is implemented as can be seen in the **Figure 8.c & 8.d**

A screenshot of a number

Description automatically generated

**Figure 8.a the corresponding ranges of pulsed width with the speed**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Clockwise(faster to slower)** | **stop** | **Counterclockwise (sloer to faser)** |
| **Pulse width control** | 6.4% … 7.4% | 7.4% … 7.6% | 7.6% … 8.6% |
| **RPM** | 140…0 | 0 | 0… -140 |

**Figure 8.b converting range of PWM from us to percentage**

**A computer code with text

Description automatically generated**

**Figure 8.c mapping e from the speed (rpm) to the PWM rate**

A black text on a white background

Description automatically generated

**Figure 8.d mapping function**

**Robot behavior**

In phase 1, the robot simply moves forward at a preset seed 60rpm and turns away to another random direction the same position. The state machine in Figure **5** shows the robot behavior responds to the distance value taken by the ultrasonic though the movement.

To say, the movement of the robot strongly based on the set speed on the servos both sides. Separate set of speed settings on 2 servos will let to different movements. Basically, there are 3 types of movement: linear movement, turning around an abstract reference point and turning around its center between two wheels.

Starting with the statical movement, this movement is easily to represented by setting the speed of both servos similar with each other which get the robot moving forward or backward. One noticeable about this movement is that, two servos are faced two different directions, which result in the turning direction of wheels are in the opposite direction. As a result, one must be set to turn backward again the clockwise, while the other following the clockwise or vice versa to perform this movement. **Figure 9.a**

The second movement was turning around one abstract point apart from the robot. This movement can be performed by let the turning speed of a servo faster than the other. The point is called Instantaneous Centre of Rotation(ICR). An illustration can be seen in the **9.b** along with necessary for calculation **Figure 9.c.**This kind of movement is not yet implemented in this phased and be planned to apply in the next phase with the formulas.

The last movement is moving around itself at the same position. This performance can be performed by let the servo turning in the opposite direction at the same speed. This movement is implemented in turning into random direction function. **Figure 9.b**

A diagram of moving direction

Description automatically generated

**Figure 9.a linear movement**

A diagram of a square with lines and letters

Description automatically generated

**Figure 9.b illustration of circular movement**

A math equations with numbers

Description automatically generated with medium confidence

**Figure 9.c Circular movement formulas**

A drawing of a triangle

Description automatically generated

**Figure 9.d turning at the same position**

## Test

In this phase, the test is performed through visual as the main method. The component functions is tested through printing the value it get and response. However, testing with logic analyzer and other more concrete is not perform yet. This suppose to be tested clearer in the next phase.

|  |  |  |
| --- | --- | --- |
| **Test** | **Description** | **Status** |
| Stops going forward when close to object. | The car should stop moving forward when it’s getting close to a certain object. | success |
| Can changes direction. | The car can change direction. | Success |
| Changes direction randomly when close to object. | The car can change a random direction when it’s too close to an object. | Success |
| Print distance. | Prints the distance to a certain object to check if the ultra-sonic is working properly. | Success |
| Print speed. | Print the speed to check if it was set properly. | Success |
| Print turning. | Print the speed to check if it was set properly to the right turning speed. | Success |
| Print speed change, | Print the speed to check if it changes at the correct time when an object is close. | Success |

## Reflection

### Phase 1 Reflection:

**Group perspective**

Phase 1 laid the groundwork for our car project, using servo motors and an ultrasonic sensor.

#### Success Highlights:

1. **Hardware & Software Integration**: Setup of servo motors and the ultrasonic sensor and setup code for the servo’s and ultra-sonic.

2. **Functional Validation**: Testing confirmed usefull functionalities such as obstacle detection, direction changes, and speed settings.

#### Next Steps:

**Refinement Goals**: Focus on refining speed calculation accuracy and optimizing its efficiency by using PID.

Personal reflection:

**Tom**

I learned a better way and more readable way of setting up the timers for the ultrasonic and the servos. Kiet also learned me a nice way of using structs and I really like it. I will use it more often in the future since I really like that way of programming.

**Kiet**

In the beginning of the project, I have counter problems with timers configuration, which I have not countered much in the previous, since this project require much more timer configuration needed more careful in managing them together. Apart form the technical problem, I learnt more of working as a group more efficiently rather than doing everything alone. By listening to Tom, we figure out the problems which might take time if we do it alone and gets more new idea outside of thinking range.

# phase 2

## Design

### Timer

In order to have a better structure code, modules have been implemented. There are 5 modules in total: “Timer”, “360\_servo”, “Ultrasonic”, “Robot” and “PID\_control”.

Start with “Timer” module. There is in total one structs which basicallly an abstract timer to store the tick count and interval of a time period. Go along with this struct are the **create\_timer** & **timing** function that support for timing task of a timer. Apart from that there are 4 timer init function having the purpose of configuring the timers.

The **set\_pulse\_width** function here plays a crucial role in the whole project, since it help to set the pulse width for the PWM output mode, configuring on a timer channel. The function changes the Pulse width for the PWM output signal that control the speed of the 360 servo that allows the robot to move at different speeds based on it current state.

A computer code with text

Description automatically generated with medium confidence

**Figure 1a. Timer struct**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Return type** |
| create\_timer | Initializing the Timer struct and return to the success state of initialization. | Uint8\_t |
| timing | Timing the abstract timer based on the data stored in the Timer structure. If time is out, the tick value is reset and the timeout variable is set to 1 as true. | Uint8\_t |
| SysTick\_init | Configure of the SysTick. | Uint8\_t |
| Timer2\_init | Configuration of the timer 2 and set the PWM mode output on 2 channels which have the purpose of controlling the servo speed. | Uint8\_t |
| Timer3\_intit | Configuration of the timer 3 and set the PWM mode output on 1 channel which has the purpose of sending the trigger signal to the ultrasonic sensor. | Uint8\_t |
| Timer4\_init | Configuration of the timer 4 and set the input mode on 1 channel to detect the rising and falling edge of the echo signal from ultrasonic. | Uint8\_t |
| Set\_pulse\_width | Set the compare value on the channels configured as PWM output in order to generate different pulse width of PWM output singnal | Uint8\_t |

### 360 servo

Coming to the next module “360\_servo”. The module includes code of controlling a servo. The module includes a struct name Servo, including the necessary variable for controlling the servo.

Since the speed of the servo is configured based on the PWM output signal. This is mentioned in the **speed calculation** section of the previous part. During the program, the speed is kept track and control with the rpm unit. As a result, to convert the rpm speed unit to the corresponding pulse width rate, the **set\_pwm\_signal\_servo** is create to map the rpm value to the right pulse width rate. Since the range of pulse width have an idle zone between the pulse width range for moving backward and forward. The **set\_pwm\_signal\_servo** allows to choose the direction in other to map on the corresponding pwm range.

A computer code with black text

Description automatically generated

**Figure 2b. Servo struct**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Return type** |
| Map | Mapping the value from a range to a corresponding value on another range. | uint8\_t |
| Create\_servo | Initializing the Servo struct and return to the success state of initialization. | uint8\_t |
| Set\_pwm\_signal\_servo | Mapping the speed of servo from the rpm unit to the corresponding pulse width in order to generate the expected servo speed. | uint8\_t |
| Check\_update\_speed | Check update the speed of the servo based on the indicated speed via actual reset the pulse width of the PWM ouput3 signal | uint8\_t |

### Ultrasonic

With the ultrasonic, have specific 2 structs. The first one is Echo\_record while the other are the distance. Echo\_record take responsibility to record the data of the echo signal sending back from the ultrasonic, while distance is where it stores the result of the final distance calculation.

Beside that, there are four functions that executes the task of the ultrasonic. The two most

Important functions are the **check\_ultra\_sonic\_info and** **calculate\_distance. Check\_ultra\_sonic\_info** has the responsibility of reading the echo signal through the interrupts. The records then is going to be used to calculated the last result of the detected distance.

A screenshot of a computer code

Description automatically generated

**Figure 1c. Echo record struct**

A computer code with text

Description automatically generated

**Figure 1d. Distance struct**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Return type** |
| Create\_ultrasonic\_datas | Initialization of Echo\_record and Distance struct. Then return to the success state of initialization. | uint8\_t |
| Check\_detecting\_status | check to detect state is the next interrupt of echo signal is the rising or falling edge, which update the state machine for detecting the rising and falling edge of echo signal. | uint8\_t |
| Calculate\_distance | Calculate the distance base on the record time of the falling and rising edge via a converting formula from time to distance in centimerter. | uint8\_t |
| Check\_ultra\_sonic\_info | Consist of state machine of detecting and record the time of falling and rising Echo signal. This function must only be used in the interrupts handler where the interrupts created by echo signal is happened. | uint8\_t |

### Robot

The robot module consists of struct and functions that executes the robot’s task. As what has been mentioned in the previous phase. Since the robot a set of sensor and actuators (ultrasonic and servo) and a set of actions based on a certain condition, it is best to divide the robot data to be stored in two structures: Robot and Movement. The robot whole the data of from the sensor, whereas the Movement keeps track on the moving variable of the robot such as speed and movement mode.

Beside that there are 4 functions in total. The drifting robot function is where the logic of robot movement behavior is performed.

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Return type** |
| robot\_init | Intialise the robot struct and return to the initialization state. | uint8\_t |
| Set\_speed | Set the speed total for the robot, which also specifies the speed of every servo. The speed between them must be in the opposite direction since | uint8\_t |
| Set\_turn\_speed \_random\_direction | Set the turn speed for the robot, which randomly set the sign for the speed value of both servos. The minus would indicate the robot turn to the opposite side comparing the plus sign. This because the servos are install in the opposite direction. | uint8\_t |
| drifting\_robot | Performing the logic of robot movement behavior belong to the distance to the nearest object. This is implemented by a state machine as in the **figure** | uint8\_t |

A computer code on a white background

Description automatically generated

**Figure 1e. Robot struct**

A screen shot of a computer code

Description automatically generated

**Figure 1f. Movement struct**

### PID control

Lastly is the PID control, this model is where PID tasks function and data is organized. The struct PID basically stores the data of PID control factors including integral, proportional and derivative go along with these constants kp, ki and kd.

The PID loop execution is stored in the PID function where the PID controller get the feedback from sensor and calculating the power after a period of time. These contants of the PID controller can be tunned via the tuning functions which make it convenient to reset the tunning variable through trior and error to figure out the best suit for the current control context, in this case is for the robot smooth break.

A close-up of a computer code

Description automatically generated

**Figure 3g. PID struct**

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Return type** |
| PID\_init | Create the PID struct and return to the success state of initialization. | uint8\_t |
| Reset\_setpoint | Reset again the setpoint for the PID controller. | uint8\_t |
| tunning | Tunning the constants value of the PID control factor. | uint8\_t |
| PID\_controller | Performing the logic of PID controller to get the right power after a preset period of time. | uint8\_t |

### state machine

The state machine has now 3 states:

|  |
| --- |
| AHEAD |
| SMOOTH\_BREAK |
| TURNING |

Compared to the older version the smooth break state is added.

The ahead state will change to the smooth brake state when an object is getting closer. When the distance is smaller than breaking distance the state will change into smooth break. The smooth break will start slowing down using a PID controller, which is used so it will break smoothly without any shaking effects. If the object is getting too close, when the distance is smaller than the turning distance, the state machine will switch to turning and it will turn the car until there is no object in the range. So once the distance is bigger than break distance the state machine will change back to ahead and the car will go straight again.

A diagram of a diagram

Description automatically generated

**Figure 2. State machine**

## Implementation

### PID

Using a PID controller for braking is like having a smart system that helps a car slow down smoothly and safely. It's all about making sure the car stops nicely without any sudden jerks or stops.

**Precise Control:**

The PID controller keeps an eye on how fast the car should be slowing down (that's the setpoint) and how fast it's actually going. Then, it adjusts how hard the brakes should be pushed to match those speeds just right.

**Adaptive Response:**

This smart system uses different ways to figure out how much brake pressure to use. It looks at what's happening right away (like if the car is going too fast), what's been happening over time (like if it's been going too fast for a while), and even guesses a bit about what might happen next.

**Stability Enhancement:**

The integral term within the PID controller helps the car stop smoothly without wobbling or staying too fast for too long.

**Optimal Performance:**

You can tweak this system to work better for different cars or situations by adjusting its settings. This way, the brakes always work as best as they can in different situations.

**Smooth Deceleration**

The best thing is that with this system, the car slows down smoothly. That's super important for people inside and for anything the car might be carrying.

So, using a PID controller for braking is like having a helpful system that makes sure the car slows down just the way it should, making driving safer and more comfortable for everyone.

**Understanding How It Controls:**

The PID controller has two special parts that work a bit like a smart system managing a car's speed while braking:

**Proportional (P) Control**: Think of driving and trying to maintain a constant speed. If your car goes faster than intended, you ease off the gas pedal to slow down. The more it exceeds the desired speed, the more you adjust the pedal. This part of the controller does something similar—it responds directly to how much the car's speed differs from the ideal slowing down speed.

**Derivative (D) Control**: Picture your car's cruise control sensing changes in road incline. As the road goes uphill or downhill, the cruise control adapts to maintain a steady speed. If there's a sudden change, like a steep hill ahead, it anticipates this change and adjusts the speed smoothly. This part helps the system react to changes in the car's slowing down speed.

These special parts of the controller make braking smoother and smarter by quickly responding to how fast the car is slowing down and how far it is from the ideal speed, ensuring a safer and more comfortable driving experience.

### Integral

In this context, our PID control solution does not have the integral factor due to one specific reason is that integral is an factor that accelerates the system power when the environment show resistance on the robot. An example case for this is the detected object is moving or a drone need a certain power to hoover in the air. However, in our swarm bot project, there is just a weak friction, which is not considered to be big enough to create any significant resistance for the robot to moving forward.

Beside that, the integral contributes into the unstable of the system where there is no resistance. To reduce the unstable of that, the kI constant can be reduced. But coming back to that mater, if the Constance is too low, it becomes useless. As a result, we chose to keep the I factor out of our control system.

### tuning

To tunning the constant in our system. The approach with trial and error is chosen with pre-estimation. To detect the right value, one by one constant is tested.

First we start with kP. Since the robot is going to start to enter the smooth break process within the distance range of 20cm and stop at 10cm. The error term must be from 10 to 0 cm. Based on the power calculation using kP. **Figure a.** And the Power, which is the speed in this case must be smaller than 150. Hence, we got the highest kP are 15.

In additional to that, since the default speed for moving forward was 100rpm. We limit the constant back to be smaller than 10. However, the reduce of speed must be in the large range. If the start point of smooth break was too small, it has no reason at all. In the end the reasonable value for kP must be from 5 to 10. After some trial and error. The value of 7.3 is chosen.

The formula to calculate with power are:

Power = kP \* ErrorTerm

**Figure a**

Our tunning value:

kP: 7.3

kD: 0.5

## Test

### Ultrasonic

For the ultrasonic we tested to see if it works using the logic analyzer.

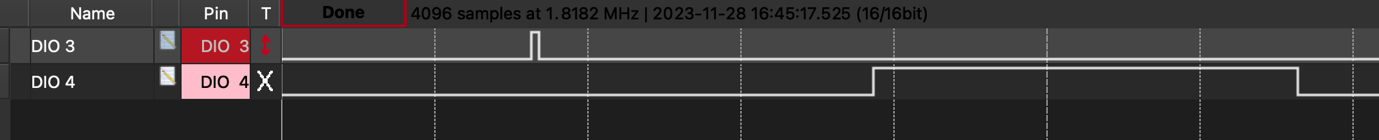
We first checked if the signal send really is 10 microseconds:

A screenshot of a computer

Description automatically generated

**Figure 3a. Servo struct**

Then we check if the signal is really received and if it changes if an object is closer by the sensor, which was the case.



**Figure 3b. Servo struct**

### Servo

For the servo w checked if the wavelength wave length becomes larger when an object is closer to the ultrasonic.

When there is nothing in range the wavelength is short and the servo spins fast.



**Figure 4a. Servo analyser**

When there is something getting closer the wavelength will become longer and the servo will spin slower



**Figure 4a. Servo analyser**

When there is something really close the wavelength will become really long and the servo will spin very slow.



**Figure 4a. Servo analyser**

## reflection

# phase 3

## Design

## Implementation

## Test

## reflection

# Bibliography