|  |
| --- |
| Title: |
| Snake-like Robot |

| candidate(s): | | | | |
| --- | --- | --- | --- | --- |
| Håkon Bjerkgaard Waldum, Marcus Olai Grindvik, Ruben Svedal Jørundland | | | | |
| Date: | Course code: | Course title: | | Restriction: |
|  | IP304814 | Introduksjon til Mekatronikk | |  |
| Study programe: | | | pages/Appendix: | Library no.: |
| Automatiseringsteknikk | | | / |  |

|  |
| --- |
| Supervisor(s): |
| Houxiang Zhang, Guoyuan Li |

|  |
| --- |
| Summary: |
| [The summary here will be identical to the SUMMARY section of the report] |

This report is submitted by students for evaluation at NTNU Ålesund.

Table of contents

[summary 4](#_Toc25314818)

[terminology 4](#_Toc25314819)

[1 introduction 5](#_Toc25314820)

[1.1 Goals of the project 5](#_Toc25314821)

[1.2 Project Planning 6](#_Toc25314822)

[1.2.1 Project timetable 6](#_Toc25314823)

[1.3 Risks and Possible Challenges 7](#_Toc25314824)

[1.3.1 Parts and Movement 7](#_Toc25314825)

[1.3.2 Pictures and lighting 7](#_Toc25314826)

[1.3.3 7](#_Toc25314827)

[2 Background and theoretical basis 7](#_Toc25314828)

[2.1 Reasoning for project 7](#_Toc25314829)

[2.2 Usage in the real world 7](#_Toc25314830)

[2.2.1 Search-and-rescue 8](#_Toc25314831)

[2.2.2 USAGE 2 8](#_Toc25314832)

[2.3 Power Consumption 8](#_Toc25314833)

[2.4 Image processing 9](#_Toc25314834)

[2.4.1 Edge Detection 9](#_Toc25314835)

[2.4.2 Creating Masks for the Color Thresholding 9](#_Toc25314836)

[2.5 Algorithm for Searching 10](#_Toc25314837)

[2.6 Recognizing dead-ends in the maze 11](#_Toc25314838)

[3 methods and materials 12](#_Toc25314839)

[3.1 Parts and Assembly 12](#_Toc25314840)

[3.1.1 Parts Included 12](#_Toc25314841)

[3.1.2 3D Models 13](#_Toc25314842)

[3.1.3 The Maze 16](#_Toc25314843)

[3.1.4 Assembly 17](#_Toc25314844)

[3.2 Software 19](#_Toc25314845)

[3.2.1 Movement 19](#_Toc25314846)

[3.2.2 Commands 19](#_Toc25314847)

[3.2.3 Image Analysis 19](#_Toc25314848)

[3.2.4 Graphical User Interface 20](#_Toc25314849)

[3.2.5 Dead End Detection 20](#_Toc25314850)

[3.2.6 Pathfinding 21](#_Toc25314851)

[3.2.7 Logging 21](#_Toc25314852)

[3.3 Test Setup and Plan 22](#_Toc25314853)

[3.3.1 Initial tests 22](#_Toc25314854)

[3.3.2 Testing of RRT\* 22](#_Toc25314855)

[3.3.3 Testing of “Go to Target” 22](#_Toc25314856)

[3.4 Computer Analysis Programs 22](#_Toc25314857)

[3.5 Communication 22](#_Toc25314858)

[3.5.1 Communication between ESP32 and PC 22](#_Toc25314859)

[3.5.2 Communication between ESP32-CAM and PC 23](#_Toc25314860)

[3.6 Program Structure 23](#_Toc25314861)

[4 results 24](#_Toc25314862)

[4.1 Theoretical Analysis 24](#_Toc25314863)

[4.2 Experimental Results 24](#_Toc25314864)

[4.3 Design Alternatives 24](#_Toc25314865)

[4.3.1 Aluminium Frame 24](#_Toc25314866)

[4.3.2 More sensors 24](#_Toc25314867)

[4.3.3 24](#_Toc25314868)

[4.4 Challenges and Problems 24](#_Toc25314869)

[4.4.1 Communication via TCP vs. UDP 24](#_Toc25314870)

[4.4.2 Communication challenges 25](#_Toc25314871)

[4.4.3 Detection of Dead Ends 25](#_Toc25314872)

[5 Discussion 26](#_Toc25314873)

[5.1 Predicted Time vs. Used Time 26](#_Toc25314874)

[5.2 Result vs. Plan 26](#_Toc25314875)

[5.3 Improvements 26](#_Toc25314876)

[5.3.1 Design Improvements 26](#_Toc25314877)

[5.3.2 Program Improvements 26](#_Toc25314878)

[5.3.3 kkkkk 26](#_Toc25314879)

[5.4 What worked vs. What did not work 26](#_Toc25314880)

[5.4.1 Choice of how to iterate through dead ends 26](#_Toc25314881)

[5.4.2 How to detect dead ends 26](#_Toc25314882)

[5.5 Challenges in usage 27](#_Toc25314883)

[5.6 Comparison with real snake 27](#_Toc25314884)

[5.7 Comparison with other robots with image processing 27](#_Toc25314885)

[6 Conclusion 27](#_Toc25314886)

[7 ReferENCES 27](#_Toc25314887)

[Appendix 29](#_Toc25314888)

*[This report contains predefined styles that you can use for the most common sections. The following styles are defined:*

*Heading 1 Heading for level 1 (hotkey Alt-1)*

*Heading 2 Heading for level 2 (hotkey Alt-2)   
Heading 3 Heading at level 3 (hotkey Alt-3)   
Body Standard text in a paragraph. Use this for all "normal" text (Shortcut Alt-A)   
Definition used mainly in the section entitled "TERMINOLOGY"   
References used in the Reference section.   
AppendixList used in APPENDIX section.   
Comment in blue text. Remove all the text of this type of report.]*

*[NB! This template provides a suggested structure of the main report. The main structure shall be followed.. However, the report must be structured by creating sub-chapters under main shapters. To some extent uou are free to decide how many sub-chaøpters and levels you want. Try anyway to avoid too many levels – normally 3 levels are enough. The level is meant the number of under-the chapter, for example, chapter 4.3.4 is on level 3, while section 3.2 is on level 2]*

summary

*This report is the final project report of Best Practice Course – System engineering. The project of our team is about*

# terminology

RRT – Rapidly Exploring Random Tree

RRT\* - A Variant of the RRT-method

GUI – Graphical User Interface

ESP32 – A type of Microcontroller

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

BLE – Bluetooth Low Energy

IR - InfraRed

# introduction

*[This is the first chapter in the scientific report. It should treat the background for the project, the contractor, the problem / problem history and / or task to be solved. Here you should also say something about the scope or boundaries of the project.*

*Finally, you should briefly describe what the report further includes, amongst other things, what can the reader expect to find in the report.*

*Comment: This is where you will provide an introduction or a kind of presentation of the whole assignment. And it is also where you are going to present the issue to be resolved and any refinements made.*

*If the task has been dealt a specification of requirements, the main features from the requirements should be outlined here, with reference to the full requirements.]*

The project is based on modular robots. The task is to craft a modular snake-like robot that can find an object in a maze with the help of a overhead-camera as well as a front-mounted camera.

This is a task given to us by Houxian Zhang in the course “Introduksjon til Mekatronikk” as a final project for the course.

The goal is that the snake-like robot will be able to find its way around a maze to find an object, by having pre-existing knowledge of the maze by getting a feed from the overhead camera. During a new search after successfully finding the object, it should still be able to find its way around the maze if the maze has been changed, and the position of the object is changed.

There is also a goal of remote monitoring/control GUI. The wish is to be able to remotely control it via WiFi, and remote monitor what happens with the robot.

This report will go through the entire process for the task, from the beginning with building a theory for how this all should be achieved, till the final steps of testing and seeing it through to the end.

## Goals of the project

The specific goals of the project are as follows;

* Make a modular maze
* Use overhead camera to take picture of the maze
* Feed maze through pathfinding algorithm
* Send this path to the snake so it can efficiently search through it
* Search for object in maze using front-facing camera on snake
* Find the wanted object

The goal is that one can build the modular maze however one wants and the snake should be able to find the object in the most efficient way possible.

## Project Planning

The project planning consisted of separating the project into its base parts; hardware and software. From there subcategories were created for each of the categories.

Hardware was separated into;

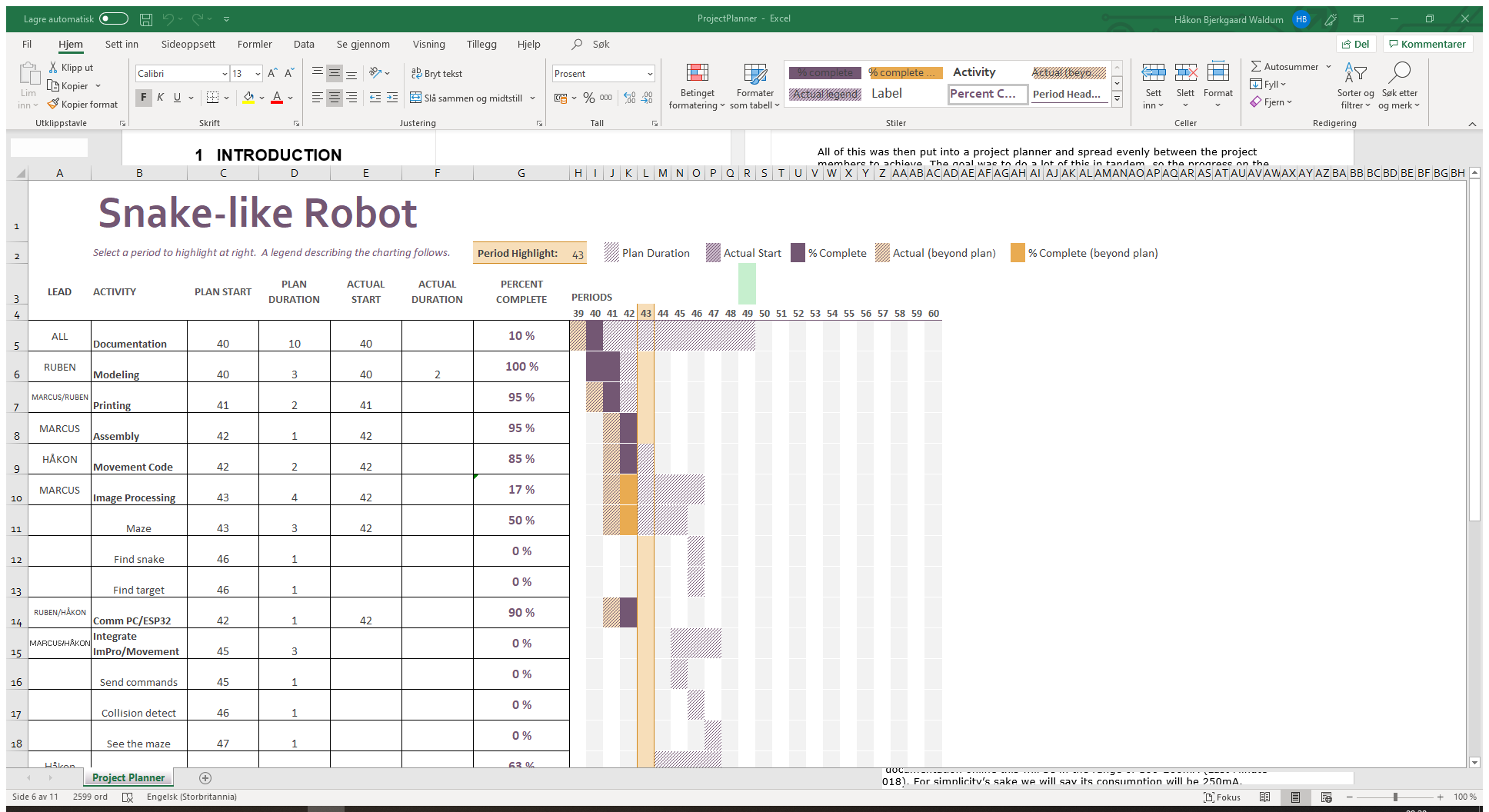
1. Modelling
2. Printing
3. Circuitry
4. Assembly
5. Maze-building

Software was separated into:

1. Image processing
   1. Maze
   2. Find snake
   3. Find Target
2. Communication ESP32/PC
3. Pathfinding
4. Remote GUI/Monitoring
5. Giving commands to the snake

All of this was then put into a project planner and spread evenly between the project members to achieve. The goal was to do a lot of this in tandem, so the progress on the project would be efficient and swift.

### Project timetable



Timetable for project

The timetable (found in the appendices) was created and the tasks spread evenly between the project members. Here there are goals with corresponding predicted times for starting and finishing each task. There is also information here about how long the tasks actually took, which is shown compared to what was predicted.

During the project this was regularly updated so each member could see how the others were doing with their task if the work was done separately instead of together at the university.

## Risks and Possible Challenges

During this project there are a lot of risks and possible challenges that will be faced. This will be detailed and reflected upon during this report.

### Parts and Movement

During the movement of the snake-like robot there is always an inherent risk in parts being damaged by the movement or unintended collisions. This is something that has been thought of, and all parts are fastened tight during assembly to ensure that movement of parts during the snakes movements will be minimal.

There is also an inherent risk in the cabling of the electronics, that these will get worn by all the movement in the modules during a run. To mitigate this the modules have gotten guide rails for the cables to run via, so the wear will be reduced. The length of the cables have also been extended by a bit, so there is no risk of them being ripped apart during a movement.

### Pictures and lighting

As this will be done in different environments and at different times of day, there has to be some failsafe in that the pictures and video streams will not necessarily have the same lighting every time the program is run.

This must be fixed by doing the filtering and edge detection so general and failsafe that given any environment (with some form of normal lighting) the edges will be crisp every time. This will be done by doing the Gaussian Blur properly, as well as adjusting the thresholds for the Canny Edge detect.

### 

# Background and theoretical basis

## Reasoning for project

The reason that this is the project that was chosen was the challenges that would arise during the building and programming. Making the snake solve a maze is in itself not that big of a challenge, but incorporating a over-head camera to give the snake directions and finding the most efficient search-path is quite a challenge.

The combination of using the over-head camera with a front-facing camera presents quite the interesting problem; how does one synch the information from one camera with the other, and when the object in the maze is found, can we make the over-head camera be able to present this on just from the picture and position of the snake?

## Usage in the real world

This project has several possible uses outside in the real world. Here some of them will be presented;

### Search-and-rescue

In a potential case of an earthquake hitting a city, one could have a drone fly over a location with a lot of rubble, to give a search-robot a general layout of the area, and guide it through the most efficient way of searching the area. Then by synching the robot and overhead drone, it could be possible to show on a video stream from the flying drone where there are possible locations for what the search-robot thinks are humans.

### USAGE 2

Fill text

## Power Consumption

When choosing the amount of batteries and their size, there are several things to take into account. The power consumption of the ESP32, as well as the 5 servos, the boost converter and the front facing camera will all factor into how long the battery time of this snake-like robot will be.

The ESP32 has several different modes it can run in to save on power consumption. For our purpose where we want access to its WiFi-capabilities we have to run it in what’s called “Active Mode”, which is the mode where its power consumption is biggest. According to documentation online this will be in the range of 160-260mA (Last Minute Engineers, 2018). For simplicity’s sake we will say its consumption will be 250mA.

The servos power consumption will all depend on how much torque it will need to apply to change its position. The producer of the servo has documented that the idle power consumption is 170mA, and at stall it will consume 1200mA (Tower Pro, 2019). From our experience we can see that it generally pulls around 200-300mA.

When it comes to what camera is used, it ended up being an ESP32-CAM, which has a OV2640 camera installed onto an ESP32. From what documentation is found about this unit, it seems that the maximum draw from it will be around 310mA (AI-Thinker, 2019). This is with flash on, as well as brightness set to maximum. So realistically it will draw less, but it’s better to calculate with the worst case scenario.

The boost converter has an efficiency of 90% according to the manufacturer (Texas Instruments, 2019).

We can therefore summarize the power consumption;

|  |  |  |  |
| --- | --- | --- | --- |
| **PART** | **POWER CONS.** | **AMOUNT** | **TOTAL CONS.** |
| ESP32 | 250mA | 1 | 250mA |
| ESP32-CAM OV2640 | 310mA | 1 | 310mA |
| MG995 | 300mA | 5 | 1500mA |
|  |  |  |  |
|  |  | TOTAL: | 2060mA |

The total draw at constant power consumption will be around 2060mA. When accounting for the boost converter we have to calculate this over to Watts, and see how much watt the boost converter will have to produce to reach this amount.

This is the general formula for calculating power. The voltage this all will run on is 5V.

With an efficiency of 0.9, the boost converter will have to produce;

Which then brings us back to;

So our current draw will be 2289mA.

Our snake-like robot has mounted four 2000mAh batteries. This means that our snake-like robot will in ideal situations run for just about 3.5 hours before running dry.

## Image processing

### Edge Detection

For edge detection the canny edge detect is used. This is based on a relatively simple process:

1. Find intensity of gradient in X-direction
2. Find intensity of gradient in Y-direction
3. Find the edge gradient
4. Apply non-maximum suppression
5. Apply thresholds

The Canny Edge detect works by detecting the intensity of the gradients in X- and Y-directions, before it finds the edge gradient by using this formula:

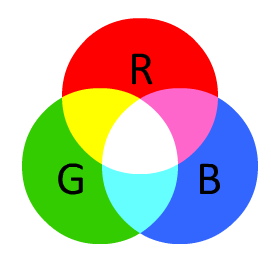
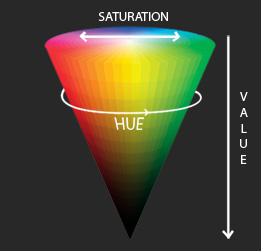
After this the non-maximum suppression is applied. This means that it tries to find the largest edge by removing other weaker edges. It compares the edges to edges in the positive and negative gradient directions, i.e. for vertical edges it compares to vertical edges over and under it.

The last step is then to apply the thresholds to remove unwanted edges. This is done by setting two thresholds, a lower and a higher threshold. Everything over the higher threshold is counted as an edge, everything between the thresholds are counted as an edge if it is nearby a strong edge, and everything under the lower threshold is automatically suppressed.

### Creating Masks for the Color Thresholding

To separate the different parts needed in the maze, i.e. the snake, the target and the maze, masks are used. For the maze this is simple; everything that is a wall has a black edge on top, and the rest of the walls/floors are white. This is to make them easily distinguishable.

For separating the different parts of the snake different colors are used. The frontmost part is red, the middle part is green, and the back-part is purple. To be able to just see these different parts, different color masks are made. These are made in the HSV-spectrum, as it is easier to distinguish colors there, because you look at more than just how many of the primary colors are in the given color (Glover, 2016).

  
Comparison of HSV vs. RGB, (Glover, 2016)

Then by using different calculators online (like [this](http://colorizer.org/) one) it was quite easy to find a lower and higher threshold for the differing colors, which then made it easy to separate them out of the pictures.

## Algorithm for Searching

After getting a picture of the maze, the picture is run through several filters before being used in a RRT\*-algorithm, which will find a path through the maze.

RRT is based on getting a node to start from. From this it will create a random node inside a given area based on input parameters ([Xmin, Xmax] and [YMin, YMax]), this is to stop it from creating nodes that are impossible to reach.

From there it will find the nearest node, and then try to iterate forward to the random node created, before checking if this path will collide or intersect with a obstacle (in this case a wall), as well as checking if it is to close to an obstacle (given by a parameter).

It will try to go straight towards the goal with a percentage chance given by a parameter. It will do this with the same procedure as iterating forward towards a random node, with checking collision, intersection and how near the path is to the obstacles.

The RRT\* method is a continuation of the regular RRT-method, it implements the same methods, but overrides how they are used to find the paths as well as implementing some new methods for how the nodes are handled.

The big difference between them is that RRT\* will look at old nodes when continuing its path to check if it can rewire the path to be more efficient. It does this by giving each node a “cost”, which is based on its distance to its parent node. It therefore compares the cost to its current parent, to another node that is close to check if the cost can be reduced.

This all results in a more linear path to the goal, instead of the path which would come from the regular RRT-method, which almost always will be very full of twists and turns and will not be as efficient.

Under the search path of the RRT-method is shown.

Et bilde som inneholder tekst

Automatisk generert beskrivelse

As the picture shows the path is not very linear in some points, and there are quite a few twists and turns all over the pathing.

Et bilde som inneholder tekst

Automatisk generert beskrivelse

When compared to the RRT\*-method over, one can see why the RRT\*-method gives a better end result. The rewiring can be clearly seen all over, where there are a lot more sharp turns with straight paths after. This makes the end result from the RRT\*-method a lot more desirable than that of the original RRT-method.

## *Recognizing dead-ends in the maze*

To detect the dead ends templating is used. Templating is where one compares features from templates on a picture to see if there is a resemblance. By setting a threshold it is possible to narrow in what the result of the templating will be.

When templating compares these templates (or features) with the pictures it will give a value of how much it thinks this matches what the template is.

By setting the threshold to the correct amount, it is possible to get out what is most likely the dead ends. It will give out the coordinate for the lower left corner of the point where it thinks the dead end is. Finding the center then has to take into account the height and width of the template to correctly find the center point of the dead end.

# methods and materials

## Parts and Assembly

### Parts Included

The snake-like robot has a lot of parts included in it to work properly. It consists of a total of 5 modules that are connected to make it able to move as we want. The modules are basically all the same, but some of them have some unique flavor for all the parts to be able to fit properly. The front and the back are also unique in that the front has to be able to fit the front-facing camera, and the back has to have a end-plate so the batteries do not fall out.

|  |  |  |
| --- | --- | --- |
| **PART** | **AMOUNT** | **FUNCTION** |
| ESP32 | 1 | Controlling the movement of the snake-like robot, as well as communicating with the PC which does all the image processing, pathfinding etc. |
| Boost Converter | 2 | Converts the nominal voltage from the batteries to 5V for the servos and the ESP32. |
| ESP32-CAM | 1 | ESP32 with a camera mounted on it, mounted in the front of the snake-like robot, used for finding the object in the maze. |
| Battery 2000mAh | 4 | Used to supply the robot with the power required. |
| TP4056 LiPo Battery Charger | 1 | Used to slowly recharge the batteries between usage when connected through a USB. |

#### Servos

The servos used are a type which are called TowerPro MG995. These are quite small, and have can turn from 0° to 180°, and run on 4.8V to 6.6V. These make them ideal for this project, as we do not need more than 180° turning for the movement of the robot.

The power consumption of them also make them ideal, they use quite a bit as discussed earlier in the report, but they do not consume so much that it is a problem for the life time of the robot.

Their update time also make them quite good for this, as they only use 0.2s to turn 60° at ~4.8V (Tower Pro, 2019). This make them quite fast, which in return makes the robot able to move quite fast if need be.

#### ESP32

The microcontroller used in this project is a SparkFun-microcontroller with the ESPRESSIF ESP32 chip. When you compare this to an Arduino, the Arduino falls quite short when it comes to computing power and speed.

It has a dual-core processor with up to 240MHz clock frequency, as well as integrated WiFi-transceiver and dual-mode Bluetooth (both classic and BLE) (SparkFun, 2019).

This all makes the controller quite fast, and ideal for a situation like this where a lot of information has to go back and forth while doing tasks.

The ESP32 mounted in the front with a integrated OV2640 camera has the same specs as the SparkFun-module, the only difference being the OV2640 camera that is mounted on the board.

The camera mounted is capable of giving pictures and streams in a resolution up to 1600x1200px (ArduCam, 2019). At full resolution it can do up to 15fps. This is not very relevant for the project, as there will mostly be still pictures returned from the front-facing camera.

#### Batteries

The batteries used are relatively standard Lithium-Polymer rechargeable batteries. They have a nominal capacity of 2000mAh, and a nominal voltage of 3.7V (RS Pro, 2019).

The reason for using these exact batteries are their size and capacity. The size is 62x43.3x7mm, which make them ideal for fitting inside the modules for the robot, which have a dimension of 64x64mm.

The capacity makes us able to run the robot for a long time, without having to recharge the batteries. This is discussed further up in the report, under power consumption.

#### Boost Converter

The boost converters used for this project are converters that can do 2V-24V to 5V-28V, with an output of 2A (BangGood, 2019).

With the total consumption of this robot being around 2000mA, there was a need for boost converters that could handle around that output. So with two of them installed, there was no fear of getting brownout-errors from the ESP32 (Brownout-error is an error given when there was not enough power available).

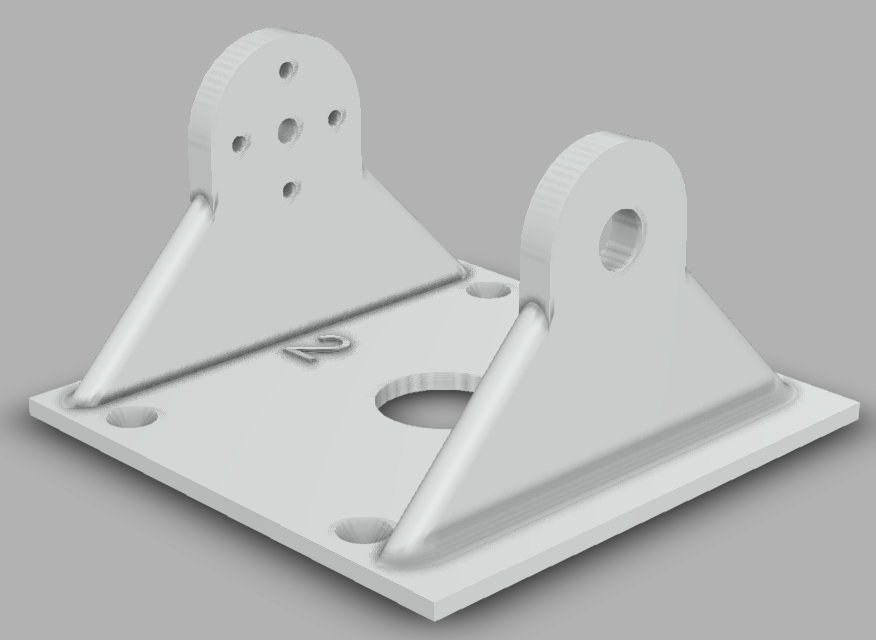
#### USB Battery Charger

For the charging of a battery, a TP4056 LiPo Battery Charger with USB connection was used. This has a charging current of 1A and a full charge voltage of 4.2V (BangGood, 2019).

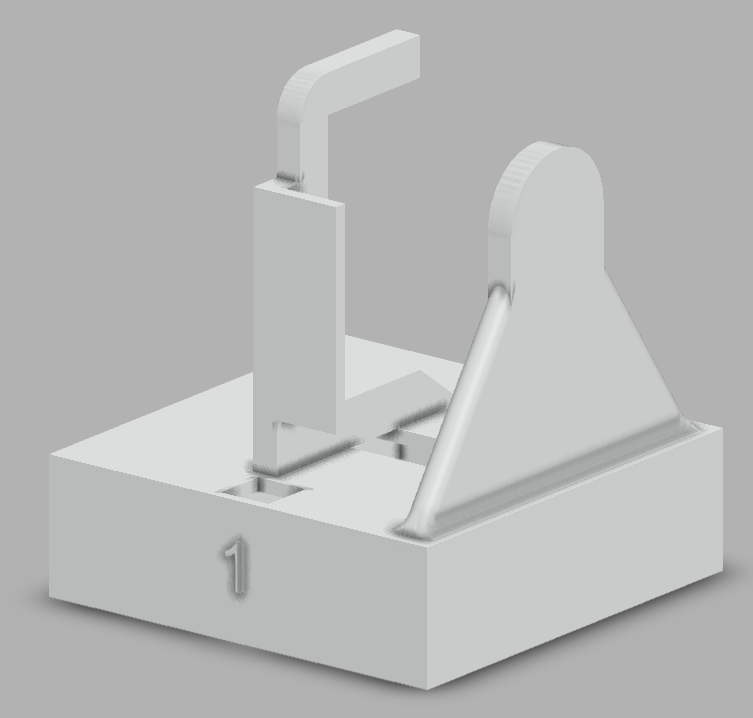
This makes the charging not the fastest, but at least gives a consistent way to charge the batteries easily by using a micro USB-cable.

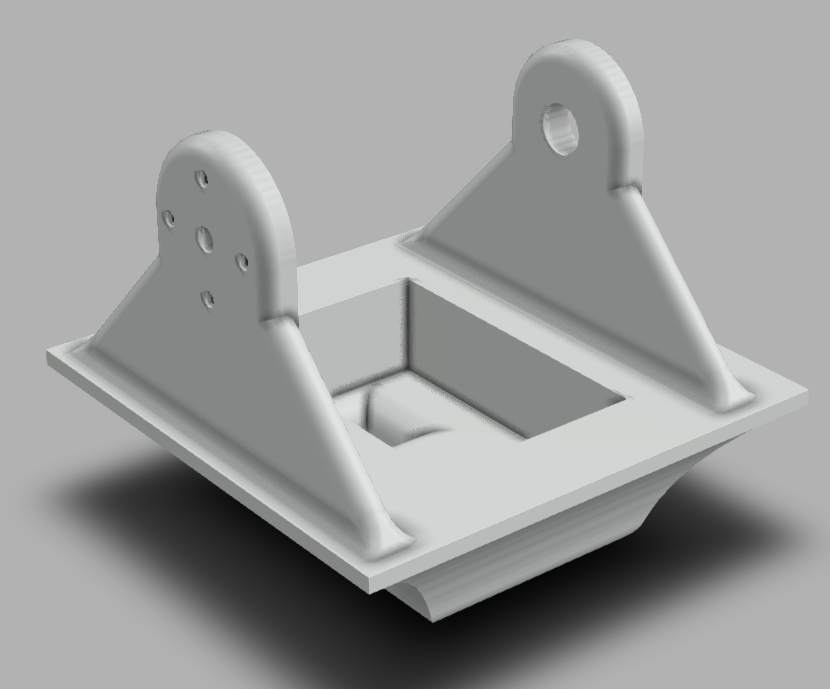
### 3D Models

All mechanical parts for the snake-like robot are modelled using Fusion 360, and are made so it can be expanded with relative ease. Under are shown the basic parts, with also some unique parts, and the complete assembly.

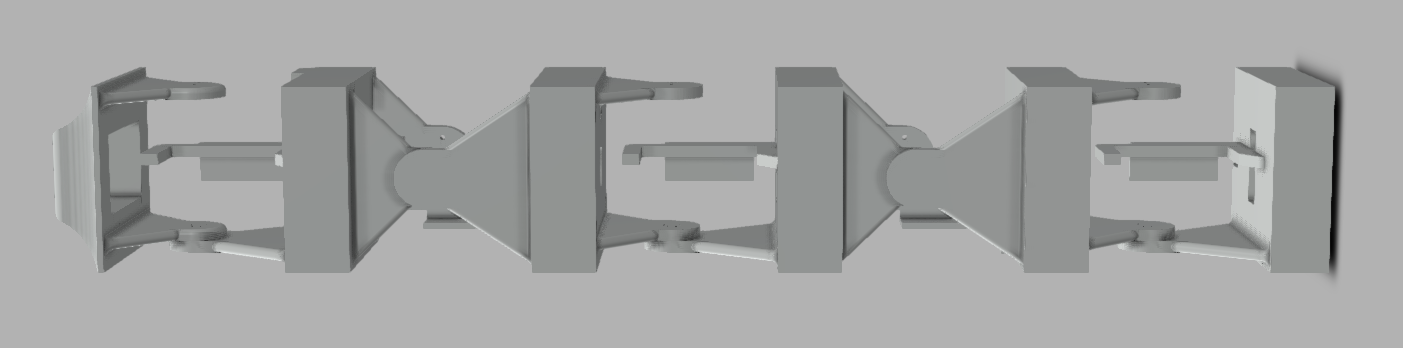


Over one of the parts are shown, this is a basic part that is connected to the other piece which will have the servo mounted on it.

This part is connected to the first part shown. This is the one that has the servo mounted on it, it also has a thick base which has room for batteries, boost-converters or similar parts.



This is the front of the snake-like robot. This has space in it for the front-facing camera. This part will be painted in a distinct color, which makes it possible to find the front of the snake-like robot with ease.



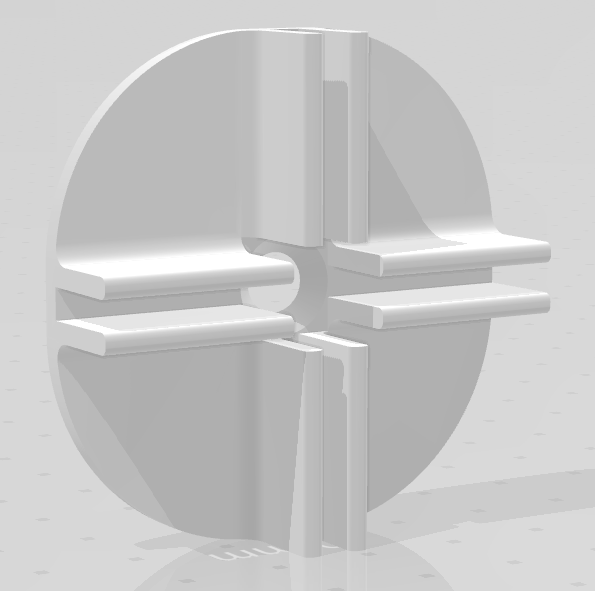
The snake-like robot assembled completely is shown above. As mentioned earlier, it consists of 5 modules, with a designated front, and a designated back. All the parts in the middle between the two modules mounted together are made for storing the electronic parts needed for the project.

### The Maze

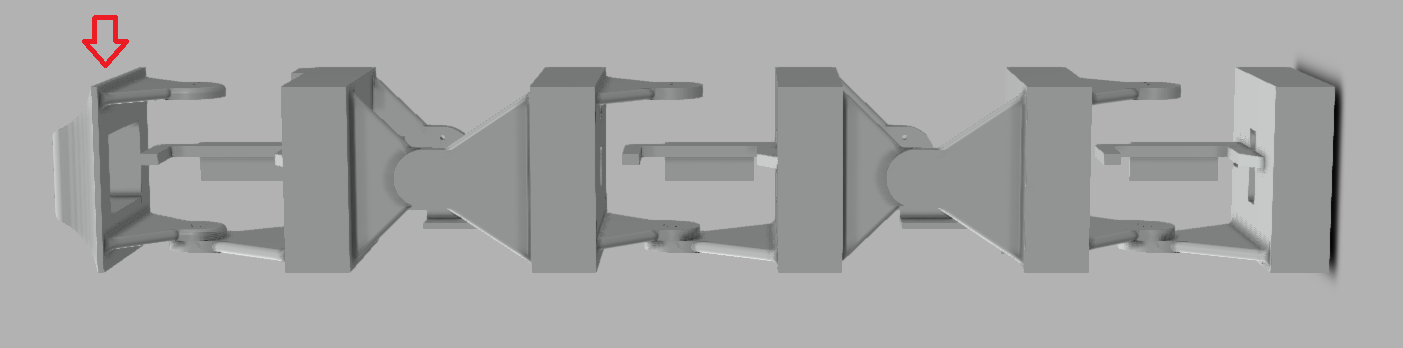
For the maze included in this project it was decided to make it modular, so there was an easy way to change the layout to really be able to test both the snakes movement as well as the pathfinding algorithm easily.

It consists of 8 walls that are 70x15cm big, with a thickness of 3mm. These are to build the walls around the maze. For the inside walls 30 walls with a dimension of 20x15cm, also with a thickness of 3mm.

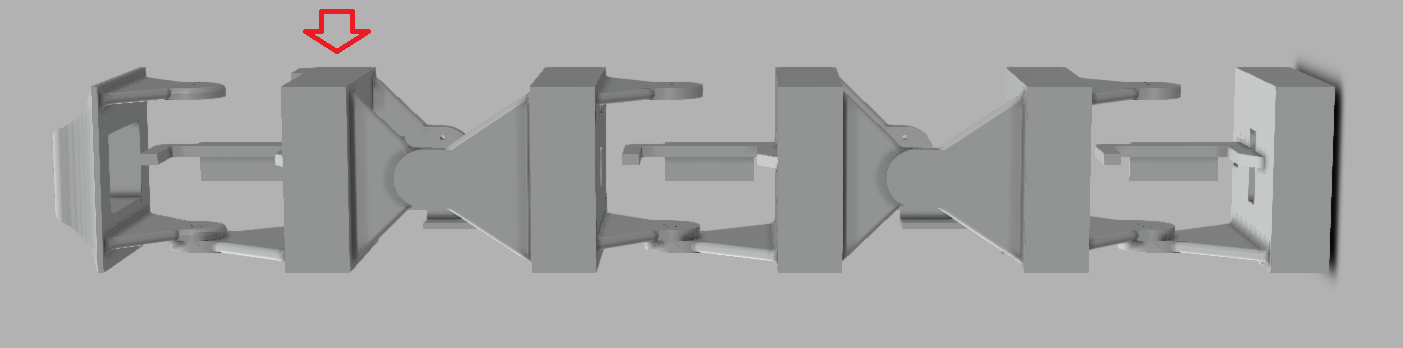
To be able to mount this easily together a small holder was designed and printed. This makes it easy to mount the maze together in different layouts without any big extra trouble.



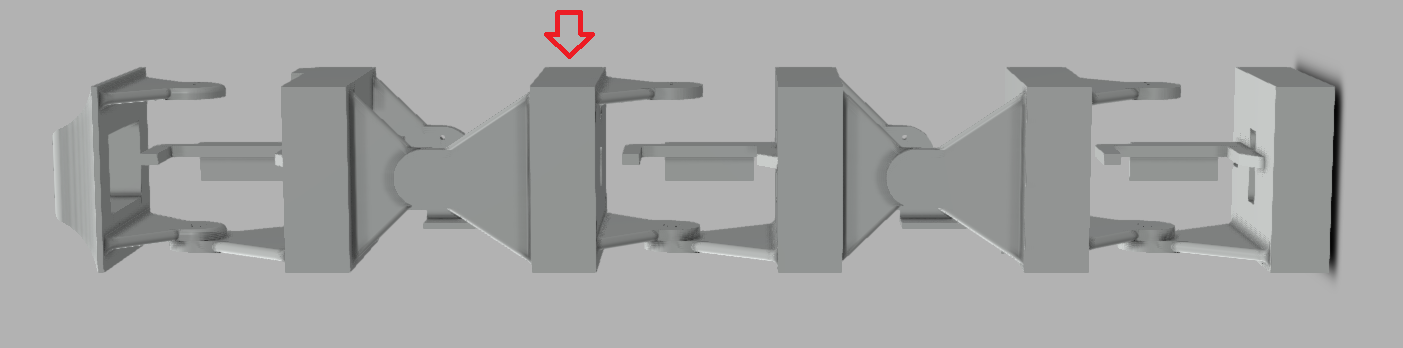
### Assembly



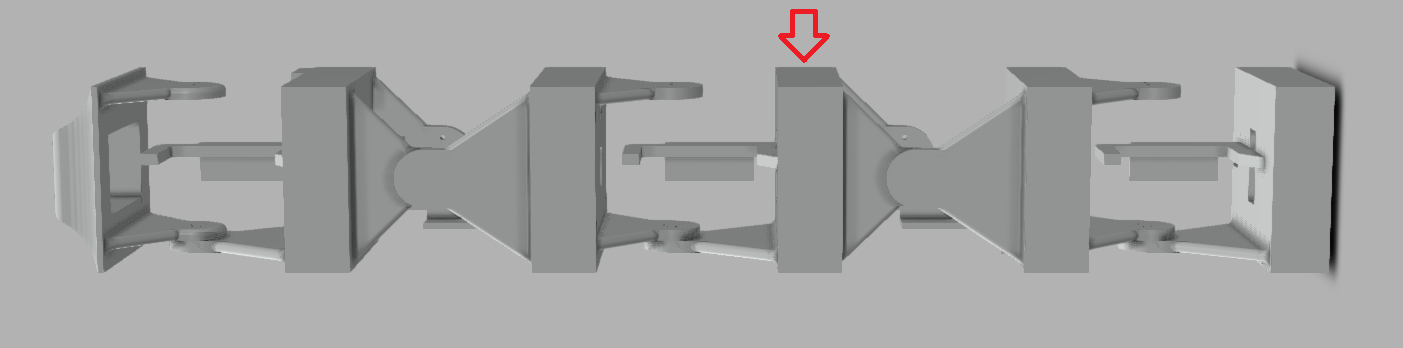
In the first piece there is only the ESP32-CAM which is mounted. This part is only connected to power, and sets up a Web Server on its own IP-address, port 80. Which is accessed by Python to retrieve a video stream or a picture.



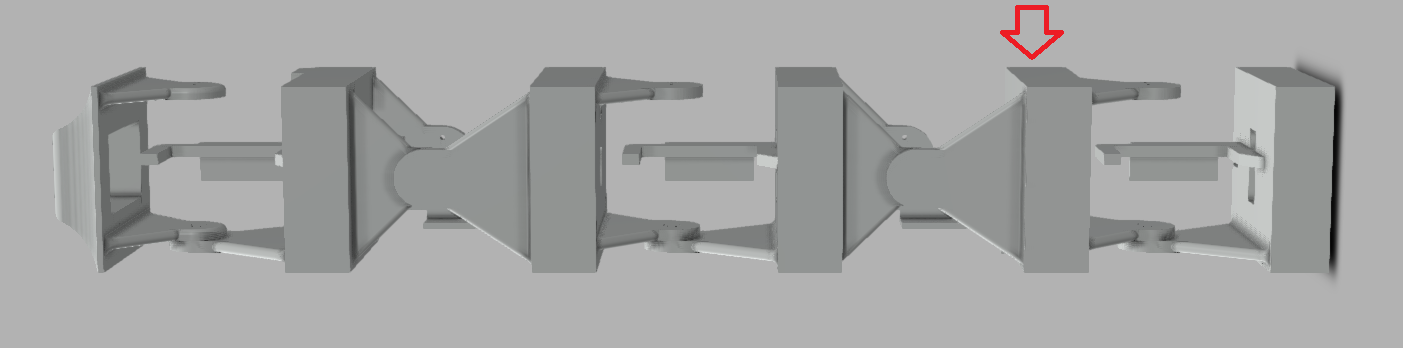
In the second piece there is mostly just circuitry, a connection point for ground and +5V. This is just a junction for the cables basically.



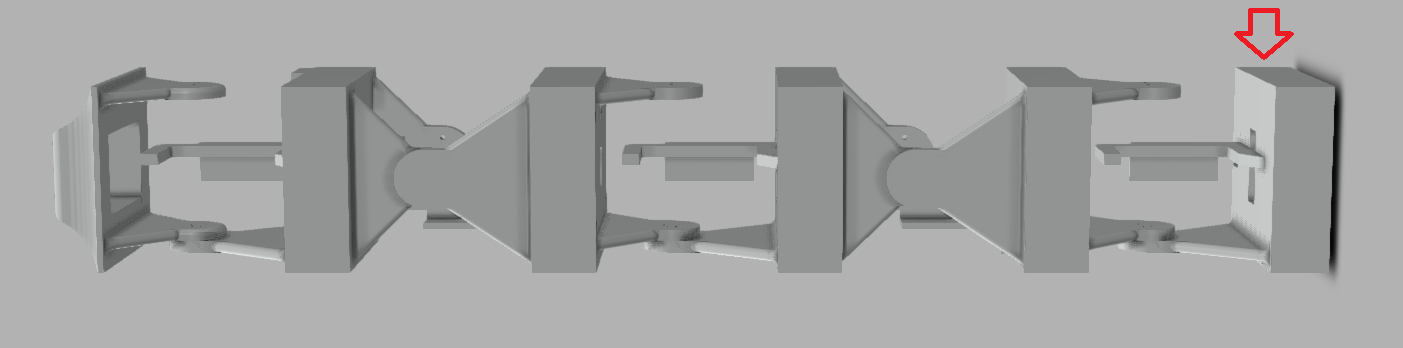
In this part the ESP32 is mounted, as well as a button which is mounted to be able to reprogram the ESP32.



‘In this part the whole power circuit is mounted, which consists of the two boost converters, external power port as well as a power button to be able to remove power from the servos and the ESP32.



This part consists of two batteries connected in parallel. These just supply the rest of the robot.



This part consists of two batteries, as well as the USB recharging board, which makes it possible to recharge the batteries slowly.

## Software

The software for the robot is done with two programming languages;

1. Arduino
2. Python

The ESP32 and ESP32-Cam both run on Arduino, and the rest of the programming (image analysis, pathfinding etc.) run on Python.

### Movement

The movement is done in the ESP32, and is done with Arduino. The movement is based on a phase difference between the different modules to actually make it move efficiently.

This is all based on the lectures from Houxiang Zhang, where he details what his own experiments when working with these kind of modular robots resulted in.

For the forward and backward movement, the formula used is;

This is then sent to the vertical modules of the robot. The phase shift between the modules for forward movement is 120°.

When it comes to turning, this is done by just adjusting the angle of the two horizontal modules of the robot by giving them an offset adjustment. This is done by giving them a direction (left or right), then giving them an angle to turn by.

### Commands

The commands given to the ESP32 is done via UDP, this is detailed further into the report under Communication. The PC sends single letter commands to the ESP32, some examples are “f” for forward, “b” for backwards etc.

From this the snake will do the tasks programmed to these commands, and send responses to the PC about whether it is received and done.

### Image Analysis

The image analysis for this project is done in Python, by using the OpenCV library. This supplies a wide array of tools to do almost anything with incoming streams or pictures.

#### Overhead Camera for Maze

For the overhead camera the basic gist of the operations are;

1. Take picture
2. Blur picture to smooth out uneven colours
3. Use Canny Edge detect to find the maze
4. Use Hough Transform to find lines
5. Store these lines for later use

When blurring the picture the Gaussian Blur-method is used. It was decided to use the Gaussian Blur because this is one of the most efficient ways of blurring a picture, as well as because the Canny Edge Detect works best if a Gaussian Blur is applied before running it.

By then applying a Hough Transform to find the lines. This works by it detecting possible lines and voting over what it will determine as lines. This makes it possible to narrow in what it actually detects.

The function used in this project is a Probabilistic Hough Transform. This returns the (x1,y1) and (x2,y2) of the lines it finds. This makes it easy to use in the later functions for producing the maze in a coordinate system for the other functions to work with.

#### Overhead Camera for the Snake

To be able to find the snake easier in the maze, the decision was made to color some of the different modules in separate easily distinguishable colors. This was to make it relatively easy to threshold based on their color, and be able to remove the maze and its surroundings easily.

The first step of the process is to make masks for each of the colors. This is done by making a lower and higher threshold, and look for everything that is in between these limits. After this is done a Gaussian Blur is applied to the picture to smooth out the colors.

Then the image is converted to the HSV-spectrum before actually applying the threshold. After thresholding the image through the different masks, dilation and erodation is applied to the result of the differing masks. This is to remove small unwanted noise from the pictures, as there can be some parts of the pictures which has these small dots inside the different color ranges.

From this a contour is extracted from the end result. This is used to find the center of the mass, and therefore also the coordinates for the center of these colored modules. The function returns the result of the processing in a picture, as well as the coordinates for the separate modules. This is to produce a line later in the process, to be able to see which direction the snake is facing, and its angle.

It was also decided to implement a running average filter on the coordinates, this is because there can be some noise/uneven measuring on the center of the mass, and this is something that can create a lot of jitter in the movement.

#### Front-facing Camera

Skriv om hva man gjør for front-facing camera

### Graphical User Interface

Snakk om hva som skjer med GUI’en

### Dead End Detection

The way templating is used in this project is that there are taken dozens of pictures, and the different types of dead ends are extracted into their own small pictures. These are then compared to the pictures taken by the overhead camera, and possible dead ends are then made from the templating.

These are then weighted against a threshold, before they are thrown in a loop to compare the positioning of these different possible dead ends. This is to eliminate having several dead ends at almost the same point.

When it has run its course, it will produce a set of dead ends in a list, which can be shown as they are on the picture under.

Et bilde som inneholder innendørs, vegg, baderom, liten

Automatisk generert beskrivelse

### Pathfinding

The pathfinding for the project is done with the RRT\*-method. This method takes in the coordinates for all the lines found from the image analysis of the maze, thus creating a maze in a coordinate system.

It then gets the information about the start point for the maze (the snakes position), as well as all the “goals” in the maze, which will be the different dead ends.

It then iterates over these goals, to try every possible way to each goal, and then to the separate goals after. After it has computed all this, it compares the different paths to find which path will be the most efficient path to take for the snake to search the whole maze.

When the final path is chosen, it will send this path to the GUI to show to the end-user, and then send all information to the algorithm that sends the commands to the snake and monitors its position.

### Logging

The logger runs in the background at all times, and creates a document with everything that happens during the programs run. It details all commands sent to the snake, how many iterations the pathfinding-algorithm ran for, as well as all other communication between parts of the program and between the PC and the ESP32.

This makes it possible to recreate a run for the snake, by having all information available about what happened, what it did and what commands were sent during a run.

## Test Setup and Plan

### Initial tests

After getting the groundwork of the program up and running, the first tests that were done were basic movement from point A to point B, with a path given by the RRT\*-algorithm. Extracting the start-point from the snakes position, and setting a goal-point some distance away inside the maze.

This was to test the general movement of the snake, and what kind of controller that was needed to regulate the snakes movement.

### Testing of RRT\*

To test the RRT\*-method, the maze had to be set up and a lot of different types of mazes had to be constructed to ensure that it was possible to find a path through them. This was done manually by taking time to create the different mazes.

### Testing of “Go to Target”

After getting everything set up to automatically find the mazes edges, finding the snake and the target and their coordinates, the testing of the whole program started.

This consisted of just building simple mazes and letting the program as a whole do everything. Here different types of problems were detected that needed clearing up. The movement as a whole was jittery and unstable, where the snake had a chance of flipping over during turning movements.

This was solved by reducing the amplitude and increasing the period time for the snakes movement. This makes it slower, but more stable, which is a price one has to pay to ensure the quality of the end-product.

## Computer Analysis Programs

## Communication

There is only communication going into two different branches in this project. One branch is between the PC and the ESP32 controlling the snake, the other one is between the PC and the ESP32-CAM. The communication goes both ways in these two branches.

### Communication between ESP32 and PC

The communication between the ESP32 and the PC uses UDP. This is a very simple protocol to communicate via, as it works with a “fire and forget”-concept. There is no response built into the protocol on whether the package was received, if it was corrupted or if the receiver was not there to accept the package. By using this protocol, there had to be some redundancies built into the communication manually, to ensure that the package was received, and the task was done.

#### Communication Protocol

The protocol between the PC and ESP32 is quite a simple one. What has been done is basically create a system where the PC sends commands, and the ESP32 responds with “a” or “d” depending on the task it has been tasked with doing. The command is sent from the PC in the form of bytes. The ESP32 then reads this as a byte array, and interprets this as ASCII-characters. Seen under is a spreadsheet which shows all possible commands.

|  |  |  |
| --- | --- | --- |
| **Command sent** | **Response** | **Snake’s task** |
| “f” | “a” after activating flag  “d” after performing task | Move forward one cycle |
| “b” | “a” after activating flag  “d” after performing task | Moves backward one cycle |
| “v” | “a” after activating flag  “d” after performing task | Lateral shift left one cycle |
| “h” | “a” after activating flag  “d” after performing task | Lateral shift right one cycle |
| “m” | “a” after activating flag  “d” after performing task | Rotates clockwise one cycle |
| “n” | “a” after activating flag  “d” after performing task | Rotates counter-clockwise one cycle |
| “s” | “a” after deactivating flags | Stops all movement |
| “r” | “a” after deactivating flags  “d” after performing task | Stops movement, resets positions |
| “tXXX” | “a” before activating  “d” after performing task | Starts turning, XXX is the degrees to turn where 090 would be straight |
| “pXXX” | “a” after performing task | Changes the period-parameter for the movement. XXX is the sum to change to, this is then multiplied by 1000. |
| “aXX” | “a” after performing task | Changes the amplitude-parameter for the movement. XX is the sum to change to. |
| Anything else | “x” after receiving command | Respons with “x” to tell the PC that the command sent was not understood. |

This makes up the communication-protocol between the ESP32 and the PC. The decision was made to make the ESP32 respond to the PC to know that the commands made it through, and in some of the tasks, respond with that the task was done.

### Communication between ESP32-CAM and PC

The communication between the ESP32-CAM and the PC happens via a Web Server that the ESP32 sets up on its own IP, on Port 80. Via Python we request the picture by sending a RestAPI-request to “xxx.xxx.xxx.xxx/capture” (where the x’s are the IP of the Web Server).

## Program Structure

The structure of the program is based on Python being the master, and the ESP32’s being slaves. Python is running all the computationally heavy program, while the ESP32’s are just waiting for commands.

This means that all the heavy lifting is done on the computer. The general structure of the program itself can be seen under in the flowchart.

# results

## Theoretical Analysis

## Experimental Results

## Design Alternatives

### Aluminium Frame

One alternative to the design would be to use aluminium frames for the modules instead of the 3d-printed parts. This would make the frame quite a lot more solid and robust compared to the design today.

The big advantage of having that as a frame instead would be better mounting possibilities for all components that are in the snake. The downside would of course be that if some problems would arise with parts not fitting, or other parts needed to be used, it wouldn’t be easy to produce new parts to compensate for this.

One of the problems using 3d-printed parts have been that they can become quite weak if there are some problems during the print that are not visible when the part is finished printing. Mounting parts with screws have also been a problem where the threads of the screws destroy part of the modules, this would not be a problem with an aluminium frame.

### More sensors

One other possibility for this project would be to make the snake with more sensors in mind. For example mounting sonic sensors or IR-sensors to be able to detect collisions more dynamically than the current design.

To make this happen the modules would have to be designed differently to compensate for the space these sensors would need, as well as to ensure that the sensors would not impact the movement of the snake.

The sensors would make it possible to dynamically detect collisions without having to redetect the lines of the maze to see them. This would make it computationally less heavy than the current design. It would also make it able to more easily detect collisions on every single module instead of having to threshold to separate each module via image processing.

### 

## Challenges and Problems

### Communication via TCP vs. UDP

The initial plan for communication between the ESP32 and the computer was to use WiFi and send information via TCP. This is because TCP sends a message back when receiving packets to give notice if the package does not arrive or there is any other problem (receiver not connected to the internet etc.). But during the testing phase we saw that sending just a 100x200 array could take up to 9s one way. This would take way too much time to be able to actively send pictures and get information back to the snake to tell it if it has found the object.

It was quickly decided to try out UDP to check the time for sending information via this protocol instead. The downside to using UDP is of course that it is “fire-and-forget”. It does not care if the receiver is not online, it send the packet and is quite happy with the result regardless of what happens with the packet.

During testing here it was found that sending the same array as earlier took less than 1ms. This is a drastic improvement and it was quickly decided that this is the protocol to use. This does not create a huge risk, as the PC and the snake will not be far from each other, and there is also implemented a method where the ESP32 sends alive-messages as well as done-messages after receiving commands.

### Communication challenges

During the programming of the Arduino the program was sliced into separate modules to easily be able to test each module to see that everything worked as expected. During the merging of the programs there came some challenges that were not expected.

Specifically during the merging of the movement and the communication modules. When initializing the Arduino program during the testing here, everything went as expected. The module initialized and attached the servos to the right pins, and then connected to the WiFi and sent the test-package to a given IP and Port. But during the parsing of an incoming packet the ESP32 raised a Guru Meditation Error (the ESP32’s variation of a BSOD), saying “LoadProhibited” and gave a dump of information.

From what we could deduce from documentation found online this was because the application attempted to access a member of a structure, but the pointer to the structure was NULL (ESPRESSIF, 2019). After a lot of testing and debugging we found that if we changed the pins the servo attached to, everything went fine. After this worked, we realized that one of the pins we attached the servo to was pin 16, which is an RX-pin, which probably is what caused our problem.

### Detection of Dead Ends

During the building of different mazes there were several problems with the templating not having enough templates to compare to, when looking at a new maze. This was caused by such simple things as the maze being a bit more crooked than other times.

It was decided when doing the comparisons of the templates, that the templates would not be rotated, to save computational time. This makes it harder to find dead ends when the maze could be crooked, or some of the plates being used became slightly curved.

The only fix that was found for this problem was to just increase the amount of pictures for the templates, so it could be ensured that all possible dead ends were found.

# Discussion

## Predicted Time vs. Used Time

## Result vs. Plan

## Improvements

### Design Improvements

For the design there are several improvements that could be made. One of the big ones is a better modular design so if one module breaks, its easier to remove it and replace it with a fresh one. This could have been done by using plugs between the modules (or a similar concept), so they could easily be dismounted and replaced.

The design could also have handled all the cables that run through them better, so there were less chances of the cables being hooked into moving parts and damaged during movement.

### Program Improvements

For the program that runs there are ways to improve it. The biggest one being the

### kkkkk

## What worked vs. What did not work

### Choice of how to iterate through dead ends

Elimination vs. Iterating through every possibility

Elimination: , where N = amount of dead ends

Iterating through everything: , where N = amount of dead ends

So in a case of 6 dead ends, the difference would be:

With a difference of 699 computations needed, the elimination method is by far the best choice.

### How to detect dead ends

The first concept for detecting the dead ends was based on corner detection. The thought was quite simple:

1. Find all corners inside the maze
2. Check the distance between the corners
3. Check for collision with the walls of the maze
4. Give a range where the two corners will be recognized as a dead-end

While the idea was simple enough, there was a lot of trouble with either a lot of false positives for the corners, or not detecting enough of them.

It was therefore quickly decided to scrap this idea and try out other possibilities after churning through a lot of different corner detection methods. The final product uses templating, which in our experience gave the best and most accurate result.

The downside to using templating is that it is not the most efficient method, as it does not use any form of AI, its just a simple comparison. As the corners in the maze may come in a lot of different variations, the amount of templates needed for the comparison is quite huge. This makes the process a bit slower, but it is still manageable and does not use up to much processing power or slow down the process too much.

## Challenges in usage

## Comparison with real snake

## Comparison with other robots with image processing

# Conclusion

*[Here you should present the main results of the work together with the experience you've gained in the process. Here you will summarize the most important chapter discussions, and arrive at a conclusion. Did you solve the problem as required or expected with the chosen methods? Was the result according to the mission stated bye the employer? What did we learn from this project, both scientifically, and not least in relation to the work process of a project? ]*

# ReferENCES

AI-Thinker. (2019, 10 22). *Loboris WebServer.* Hentet fra ESP32-CAM Module: https://loboris.eu/ESP32/ESP32-CAM%20Product%20Specification.pdf

ArduCam. (2019, 10 29). *Docs: 2MP OV2640*. Hentet fra Camera Breakout Board: https://www.arducam.com/docs/camera-breakout-board/2mp-ov2640/

BangGood. (2019, 10 29). *Battery & Charger: TP4056 5V 1A LiPo Charger*. Hentet fra BangGood: https://www.banggood.com/TP4056-1A-Lipo-Battery-Charging-Board-Charger-Module-Mini-USB-Interface-p-1027027.html?rmmds=buy&cur\_warehouse=CN

BangGood. (2019, 10 29). *RC Airplane Parts: DC Boost Converter 2A*. Hentet fra BangGood: https://www.banggood.com/DC-Boost-Converter-2A-Power-Supply-Module-2V-24V-To-5V-28V-Adjustable-Regulator-Board-p-1100757.html?akmClientCountry=NO&rmmds=search&cur\_warehouse=CN

ESPRESSIF. (2019, 10 22). *API Guides: Fatal Errors*. Hentet fra ESP-IDF Programming Guide: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/fatal-errors.html

Last Minute Engineers. (2018, December 23). *Insight Into ESP32 Sleep Modes & Their Power Consumption*. Retrieved from https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/

RS Pro. (2019, 10 29). *Webdocs: 3.7V Li-Po Rechargable Battery.* Hentet fra Webdocs: https://docs-emea.rs-online.com/webdocs/163b/0900766b8163bee5.pdf

SparkFun. (2019, 10 29). *Products: SparkFun ESP32 Thing*. Hentet fra Sparkfun: https://www.sparkfun.com/products/13907

Texas Instruments. (2019, 10 16). *LOW INPUT VOLTAGE SYNCHRONOUS BOOST CONVERTER.* Hentet fra https://www.sparkfun.com/datasheets/Prototyping/tps61200.pdf

Tower Pro. (2019, 10 16). *Products: MG995*. Hentet fra Tower Pro Online Shop: https://www.towerpro.com.tw/product/mg995/

*[Authors, title of book or article, name of journal or publisher / publisher, or no date for the journal, year, place as referred to in the report. Course lectures can also be referred to, as with the title on the subject and the name of the presenter. Internet pages must also be included. Even oral discussion partners can be included in the reference list, when this is a source of important or detailed information used in the report*

*see example below]*

1. *Design of Propulsion and Electric Power Generation systems* – H.K. Woud and D.Stapersma
2. Matlab. Wavelet Toolbox. Users Guide. The Math Works Inc., 1997
3. Pugh, D T., Tides, Surges and Mean Sea-Level. John Wiley & Sons. New York, 1996
4. http://dieselmarine.com/

Appendix

*[Material and data prepared or collected in connection with the project, but not natural to include in the main part of the report. The reason cam be the level of details, volume or format.*

*Typical examples are: detailed calculations or analysis, set of design drawings, supporting information, computer code etc…. ]*

Appendix A Project Timeplan

Appendix B FEM analysis of cylinder liner

Appendix C Design drawings

Appendix D Material properties

Appendix E etc….etc…..