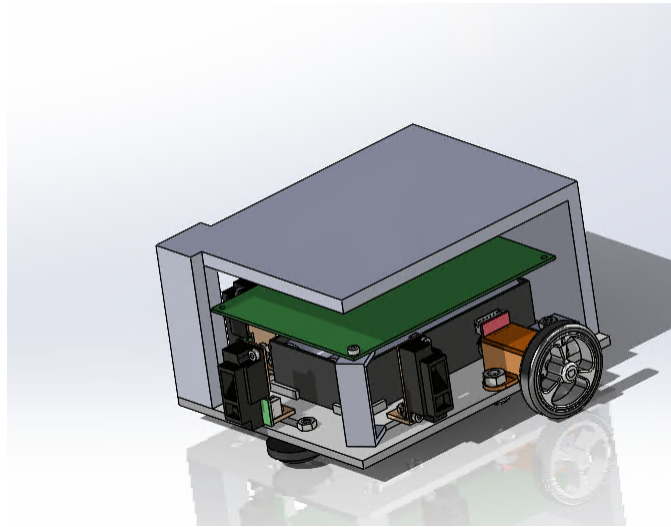


# Engineering Project - Micromouse

Mike Rotfort - 316399013

November 9, 2024



## Introduction

Micromouse is a robotics competition that challenges participants to design and build a small, autonomous robot capable of navigating through a maze in the shortest possible time. The goal is for the robot to explore the maze, locate its center, and then optimize its path to reach the center as quickly as possible. Taken at face value the task seems simple but in reality it is far from it. The project entails a multitude of challenges ranging from the design and optimization of the robot's chassis and sensors to the development of sophisticated algorithms for maze exploration and path planning. Balancing speed, precision, and computational efficiency becomes a delicate engineering task.

# Relevant competition rules (Design requirements)

1. Size Restrictions: The dimensions of a Micromouse that changes (or not) its geometry during a run shall not be greater than 25 cm  $\times$  25 cm. There are no restrictions on the height of a Micromouse.
2. Operation: A Micromouse must operate without any external control once the competition begins. It shall not use an energy source employing a combustion process. It also can't leave any part of its body behind while navigating the maze and it can't fly over, climb, scratch, cut, burn, mark, damage, or destroy the walls of the maze.
3. Starting Position: The start of the maze is located at one of the four corners. The start square is bounded on three sides by walls.

Maze Structure (in competition):

1. The maze is composed of multiples of an 18 cm  $\times$  18 cm unit square. The maze comprises of 16  $\times$  16 unit squares. The walls of the maze are 5 cm high and 1.2 cm thick.
2. The start of the maze is located at one of the four corners and is bounded on three sides by walls. The start line is located between the first and second squares. That is, as the mouse exits the corner square, the time starts. The destination goal is the four cells at the center of the maze. At the center of this zone is a post, 20 cm high and each side 2.5 cm. (This post may be removed if requested.) The destination square has only one entrance.
3. Small square zones (posts), each 1.2 cm  $\times$  1.2 cm, at the four corners of each unit square are called lattice points. The maze is so constituted that there is at least one wall at each lattice point.
4. Multiple paths to the destination square are allowed and are to be expected. The destination square will be positioned so that a wall-hugging mouse will NOT be able to find it.
5. Time Constraints: Each contesting Micromouse is allocated a total of 10 minutes of access to the maze from the moment the contest administrator acknowledges the contestant(s) and grants access to the maze. Any time used to adjust a mouse between runs is included in the 10 minutes. Each run (from the start cell to the center zone) in which a mouse successfully reaches the destination square is given a run time. The minimum run time shall be the mouse's official time. First prize goes to the mouse with the shortest official time.
6. Scoring: Scoring is based on factors such as the time taken to reach the goal, successful completion of the maze, and possibly penalties for touching walls or other rule violations [7]. Fast and accurate navigation is typically rewarded.

# Design outline

1. Size cannot be over 25×25 cm. Reasoning: competition rules (design requirement 1)
2. Able to complete the maze at least once in a 10 minute time frame. Reasoning: competition rules [6].
3. IR sensors for wall detection. Reasoning: To detect walls, determine distances and make navigational decisions (design requirement 2).
4. Needs to have either 2 wheels (with a third dummy wheel) or 4 wheels. Reasoning: provides enough stability and the winning mice in the latest competitions had 4 wheels so this is preferable.
5. Onboard micro controller. Reasoning: competition rules (design requirement 2)
6. Onboard power source that will last the entire length of the competition. Reasoning: Compliance with competition rules (design requirement 2)
7. Need to have an efficient maze exploration strategy. Reasoning: To save time on each run and ensure finding the fastest path.

## Possible exploration (search) strategies

All the strategies guarantee finding the shortest time but they differ in their efficiency of achieving this goal.

1. **Depth first search algorithm:** With this strategy, whenever the mouse encounters a dead end or a loop it goes back to the last intersection and tries a different route (i.e. turning left instead of right). The main drawback of this technique is that even though it will find its path to the goal it may miss a shorter one.
2. **Breadth first search algorithm:** This strategy is similar to the Depth first search in the sense that we use intersections as checkpoints only in a different way. Instead of going until reaching a dead end we turn back at the next intersection and try the path we skipped, before moving on to the next layer of intersections. Here, the main drawback is that we backtrack a lot and some mazes can take a very long time to explore, which means a possibly wasted run.
3. **Meticulous mouse:** Not much of searching strategy as the mouse just goes over the whole maze to make sure it doesn't miss the shortest path. Obviously this can take a lot of time, meaning we waste a run.
4. **FloodFill:** This technique strives to mimic the movement of water filling an area, covering every part of the maze. According to the rules of the competition we know that the goal point is in the center of the maze. Using this knowledge the mouse divides the maze into cells and assigns a number to each cell (the farther from the center the higher the number, just like water from a hill). The mouse assumes no walls and draws the shortest path it can find to the goal and tries to reach it. When it encounters a wall he marks it by changing the values of the surrounding cells and

draws a new shortest path based on the new cells. This technique guarantees finding the shortest path of the maze without having to explore all parts of the maze.

In summary, when comparing between these techniques it is obvious that using FloodFill is the preferred method of maze exploration both in terms of ease of coding and efficiency in finding the shortest path.

## The different parts making up the micromouse

### Communication - UART

1. The files that control the UART module are `uart.c` and `uart.h`. These files were written based on the UART manual provided by Microchip to configure the UART module in its microcontrollers. Special care must be taken into account when selecting the baud-rate in `uart.h`, this baud-rate needs to match the baud-rate of the HC-05 (the chosen UART Bluetooth module), the data-sheet of the Bluetooth module provides the instructions to configure its baud-rate.
2. Sending - The essential part of the UART is to send data to another device. This especially helps to understand more complex behaviors as the concrete values for certain distances returned from the sensors or the misbehavior of the encoders, which sometimes occurred, as well as just enables an external visualization of internal data. This can be done in three ways. The easiest and most basic approach is to just send char by char to the register `U1TXREG` of the micro-controller, which is inconvenient. A better way is to implement a method to send Strings. This can be done by repeatedly send a character to the same register as before, but with checking the `UTXBF` register, which tells if the sending buffer is full, this can be done automatically in a loop. The third and most convenient way is to just use the `printf` method provided by C, which writes to the standard output and as this is configured to the UART it works like just prompting an output on a normal console.

Receiving - Receiving is done in a similar manner. The data has to be read from `U1RXREG` in the corresponding interrupt method. It is important to write that buffer explicitly to a variable otherwise it won't be freed.

The receiving of data was very important in the beginning of the driving phase to easily change speeds, stop the motors or just test, whether we can drive completely straight, move for 90 degrees etc or if we have to improve slightly.

### QEI - Quadratic Encoder interface

The encoders are suitable for monitoring and regulating the speed, position and direction of the rotation of the drive shaft. The DC-motors have an optical encoder with two output channels, A and B. A code wheel on the shafts is

optically captured and further processed. At the encoder outputs two 90° phase shifted rectangular signals are available with 7 impulses per motor revolution.

It has two channels at channel A channel B on the motor connector. The encoders also need a ground and Udd reference of 3.3V in order to work. The 3.3V is to make the digital signal readable for the micro controller. Our micro controller dsPIC33FJ64GS608 has a flexible trigger for ADC conversions and configurations of the PWM for motor control. The micro controller does most of the work for determining the directions, whether channel A or B has its edge first. There is a 7 lines and impulses per revolution, which means that the controller counts 7x4 (two edges for two channels) for one revolution.

1. Mapping the pins - In the micro controller data sheet the two channels are called QEA and QEB for the two channels and respectively QEA/B1 for the two motors. The encoder outputs are directly connected with the micro controller through this schematic:

- QEA1= RD11
- QEB1 = RD0
- QEA2 =RD12
- QEB2 = RG1

**Initialization and interrupts** - Depending on the operating mode, the QEI module generates interrupts for the following events:

- In Reset On Match mode ( $QEIM<2:0> = 111$  and  $101$ ), an interrupt occurs on position counter rollover/underflow.
- In Reset On Index mode ( $QEIM<2:0> = 110$  and  $100$ ), an interrupt occurs on detection of index pulse and optionally when the CNTERR bit ( $QEIXCON<15>$ ) is set
- When operating as a timer/counter ( $QEIM<2:0> = 001$ ), an interrupt occurs on a period match event or a timer-gate falling-edge event when  $TQGATE = 1$

The interrupt priority level bits ( $QEIXIP<2:0>$ ) was set to 6. As the encoders are necessary for the mouse to function properly. We decided to go for a Match mode 111 where the interrupt occurs on a position counter overflow or underflow. This is defined in the QEI interrupt initialization function.

**Calculating Position and speed** - The encoders allows the actual velocity of the shaft (speed and direction of rotation) to be compared to the desired velocity and then used in a controller. Facts about the encoders: 7 pulses per motor rotations x 2 channels + 2 ups/downs (positive and negative edge) x30motor rounds per wheel turn. This gives a high resolution, and is needed to be able to drive slowly. To get the most out of the counter position count variable needs to be a long. For developing the position and speed calculation we followed an example from reference document by microchip. Instead of calculating the angular position, we converted it

into a linear position by taking the diameter of the wheels into account. This was only possible as we know which wheels we were using, but is a risk when the wheels might be changed. As one motor is opposite the other, we also had to make sure that the signs for forward and backward was correct. To increase the speed of the calculation and to reduce the risk of errors, we introduced macros such as GETENCODERVALUE1 in both functions and defined them in the header file.

**Using the PWM** - The PWM module increments its counter register by 1 every clock cycle, until it reaches its period value (PTPER), which causes a reset to 0. A dutycycle for a specific output channel (PWMx) is defined by the corresponding duty cycle register (PDCx). At the start of every period, all output channels are set active. The channels remain so until the counter reaches the value stored in the duty cycle register, when the respective channel is switched inactive. The whole procedure is illustrated in figure 1.

PDCx = 1000; // Setting duty cycle register x of module X to 1000.

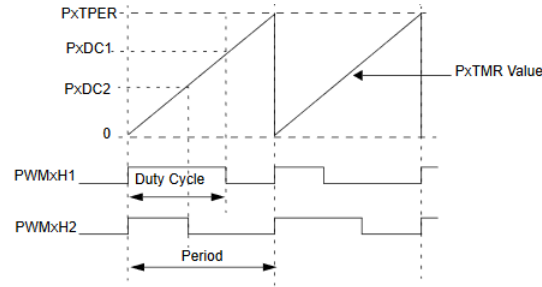


Figure 1: Edge-aligned PWM signal generation in free-running mode. PxDC1 and PxDC2 represent two duty cycle register values and PWMxH1/PWMxH2 their corresponding output channels. The top part of the image (above the 0 line) depicts the PWM counter values, counting up until being reset when reaching PxTPER. The bottom part of the image depicts the resulting channel values and duty cycles. Here, the x stands for the number of a PWM module. In this particular example the channels are high channels. An active phase of the dutycycle would therefore output 0s to the channel.

In this case, PWMx will be active until the counter reaches 1000, when it becomes inactive. It is reactivated when the counter is reset upon reaching the period value PTPER. The timer period register is a 15 bit register, while the duty cycle register has 16 bits. The duty cycle register consequently has a half-cycle resolution, allowing a finer duty cycle adjustment. The correct formula to calculate the duty cycle register value needed for 100% dutycycle is:  $DC = 2 \cdot (PTPER + 1)$ . The "+ 1" results from the counter register starting its count at 0. For any wanted duty cycle, the number to be written to the duty cycle register is the corresponding fraction of the calculated 100% DC-number.

## Control

To complete our main task of finding the shortest path to the goal of the maze, the mouse would need to perform quick and precise movements while traversing the maze. In our control methodology we implement a cascade of PID controllers, the outside loop is responsible for precise position control while the inner loop is responsible for the motor velocity control.

### Initialization

A default controller was defined as a struct describing a PID controller. Each struct contains several variables: Firstly, there are the variables necessary for the PID computation itself. An example use for these variables is to store the

last error or the integrated past errors. They are vital for the computation, but only describe a basic PID procedure and are thus not further discussed here.

Secondly, there are variables that define the PID controller itself:

$k_p$ : The multiplier of the proportional portion of the error

$k_i$ : The multiplier of the integral portion of the error

$k_d$ : The multiplier of the derivative portion of the error

$top_{lim}$ : An upper bound to the PID controllers output

$bot_{lim}$ : A lower bound to the PID controllers output

$int_{lim}$ : An upper bound to the PID controllers integral portion of the error

The upper and lower bounds on the PID controllers outputs were added to ensure sane and safe output values. Especially in the testing phase, it was vital to ensure a programming error could not lead to a motor speed that might harm the robot. The upper bound on the integral of the PID is necessary to restrain the integral from growing to extremely high values, for example when the wheels are blocked. In such a case, the boundless growth of the integral portion would lead to the motors running uncontrollably at full speed for some time after the blocking is resolved, as the massive integral portion overpowers any other control mechanism. The choice to standardize controllers as a struct was made for two main reasons: Firstly, the evaluation and calculation of a PID controller can be standardized and covered by a single function for any and all controllers as well. The second reason is that the creation of additional controllers, should the need arise at some point, is simple and straightforward.

Additionally, the struct allows to use only part of the PID controller without alteration of the surrounding process. Simply setting any of the multipliers to 0 effectively removes them from the calculation.

The final control cascade consists of a PI-controller for position control and a PID-controller for velocity control. The position controller gets a target in form of a reachable position of the controlled wheel. This target is compared to the wheels position counter provided by the QEI module to calculate the position error. The position controllers configuration is as follows:

$$k_p = 1 \left[ \frac{1}{t} \right]$$

$$k_i = 0.1 \left[ \frac{1}{t} \right]$$

$$k_d = 0 \left[ \frac{1}{t} \right]$$

$$top_{lim} = 30 \left[ \frac{d}{t} \right]$$

$$bot_{lim} = -30 \left[ \frac{d}{t} \right]$$

Here,  $t$  is the regular time passing between two consecutive position measurements (in our case 2ms) and  $d$  is the position difference (distance) measured in ticks of the QEI module. The output of the position controller becomes the

velocity controllers input. Velocity control is carried out by a PID-controller with following configuration:

$$k_p = 16 \left[ \frac{t \cdot DC}{d} \right]$$

$$k_i = 0.02 \left[ \frac{t \cdot DC}{d} \right]$$

$$k_d = 4 \left[ \frac{t \cdot DC}{d} \right]$$

$$\begin{aligned}
top_{lim} &= 2000[DC] \\
bot_{lim} &= -2000[DC] \\
int_{lim} &= 100 \left[ \frac{t \cdot DC}{d} \right]
\end{aligned}$$

where  $t$  and  $d$  are defined as before and  $DC$  is the duty cycle relative to the PWM period cycle value. The values for  $k_p, k_i, k_d$  were found by series of tests with different values until we got a satisfying performance. The methodology of selecting the values in testing was as follows; First we only apply the outer loop, we start with a small  $k_p$  ramping it up periodically until we have a small steady state error. Once that is achieved, we add a small  $k_i$  to combat the steady state. When the desired performance is achieved (no osculation and steady state error) we add the inner loop and repeat the process for the inner PID values.

## Movement calculation

Exploring the maze boils down to a repetition of basic movement patterns: Going one cell forward, turn right  $90^\circ$  and turn left  $90^\circ$ . All of these movements are implemented as functions that calculate position targets for both wheels and feed them into the control cascade. To calculate the values for a specific movement, several measurements were taken and calculated:

- wheel diameter  $d_w$ ,
- body axis diameter  $d_a$ ,
- encoder signals per wheel turn (full revolution)  $s$ ,
- wheel circumference  $c_w$ ,
- body rotation axis circumference  $c_a$ .

To move forward a distance of  $x$  mm, each wheel has turn  $n$  signals forward:  $n = \frac{x}{c_w} \cdot s$

To move forward a cell, all one has to do is measure the length of said cell and insert it into the formula above. An additional function to move any number of cells forward was implemented to avoid stopping in between every cell and thus allow for potentially smoother and faster driving after exploration. To achieve an  $y$  turn, the number of signals each wheel has to count if both wheels turn in opposite directions calculates as:  $n = \frac{c_a}{c_w} \cdot \frac{y}{360} \cdot s$ . For a  $90^\circ$  turn, this gives 840.2 encoder signals.

parameter	value
$d_w$	32[mm]
$d_a$	128[mm]
$c_w$	100.5[mm]
$c_a$	402.1[mm]
$s$	840[signals]

Table 1: measurements and calculations results

## Sensor based movement



While the robot moves, movement and odometry errors are accumulating and adding up over time. To account for this and correct the errors, sensor measurements are included into the control cascade to enhance robot positioning.

1. **Centering between walls:** When the robot moves forward between two walls, it is desirable to correct its position to keep the robot in the center between the walls. For this purpose, the distance measurement of the right sensor is subtracted from the left sensors measurement. The resulting value is added to (right wheel) or subtracted from (left wheel) the velocity controllers target. This leads to the robot turning away from walls more the closer it gets to them. To this end the value of the measurement is multiplied by a correction factor before passing the value to the controller. The correction factor modifies how aggressively the motors correct for the difference.
2. **Approaching walls:** To account for small accumulated errors in the traveled distance and to avoid crashes, the robot elicits a different behavior when moving forward and sensing a wall in front of it. Instead of just continuing the planned movement, the robot approaches the wall until it nearly touches it. The reason for this behavior is the size and axis of rotation of the robot which is located closer to the back of the robot (for stability, figure 12). By getting close to the wall we make sure that the axis of rotation is close to the center of the cell which makes the movement to the next cell easier and removes the need for aggressive corrections that can impede the movement and solving process.
3. **Wall following:** When moving forward and only one wall on one side is sensed, the robot tries to follow that wall in a distance of roughly 4cm. During movement the distance from the wall is calculated via the IR sensors and translated to cm. Then a difference is calculated between the measured distance and the desired 4cm distance from the wall. Finally, in a similar fashion to the centering process, this difference is multiplied by a correction factor and passed to the motor velocity controller.

We should note that the correction factors described in this method were found by experimentation until found the factor that gave a satisfying result.

## Sensors

For our robot, we are using the analog distance sensor GP2Y0A51SK0F by Sharp [3] to sense the walls of the maze. This sensor has a range from 2 to 15cm, which is sufficient for its task.

## Setup

Our setup consists of three of those. One on each side and a third one in the front (all three are mounted via custom made brackets to the frame of the mouse). This structure leads to an optimal coverage of the surroundings, with a minimal amount of hardware used.

## Analog to digital conversion (ADC)

We connected the sensors directly to the ADC interface of the micro-controller via analog inputs 1, 2 and 3. The result of the ADC is a 10bit integer.

## Calibration

We placed the sensor at known distances with 5mm intervals in order to create a function that translates the ADC values from the sensors to cm. Our function converts the ADC values up to 9cm, beyond which it determines that the wall is too far.

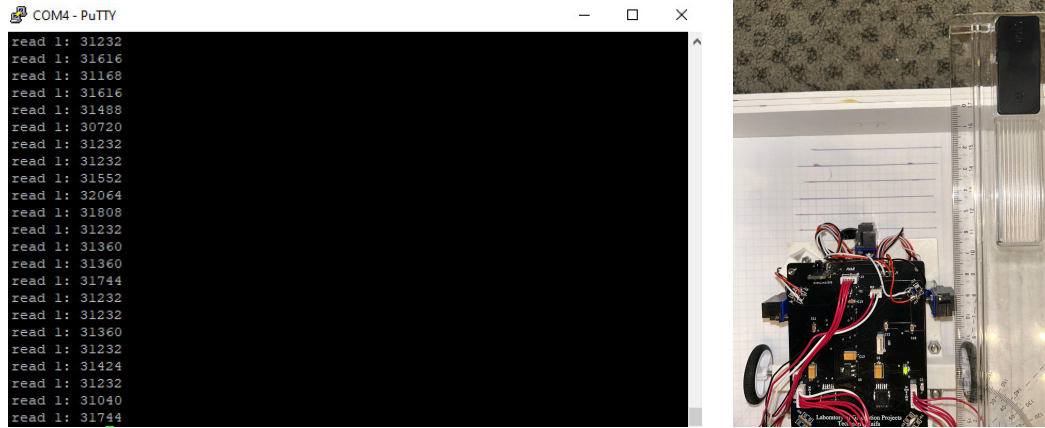


Figure 2: Left - example sensor readings from the calibration process. Right - example positioning of the sensor at a known distance in the calibration process. During calibration we noticed that as we get farther away from the wall the reading decreases and we leveraged this behavior in our unit conversion function.

## Maze exploration and finding the shortest path

For completing the main objective of this project, solving the maze by finding the shortest route, we selected the flood-fill algorithm. There are several maze sizes, we decided to solve a 4x4 maze because we are limited in room size and portability (built at home and transported to the lab at a later stage). The current mouse configuration would require a significant amount of time to solve large mazes due to slow movement ability. After finding the shortest path the mouse is then able to quickly complete the maze.

The algorithm works by having a 2D array of the maze in memory that has multiple characteristics: wall existence at each side and the value of the cell which signifies the Manhattan distance from the goal. Here is a high level outline of the algorithm:

1. Set all cell values to “blank state” (we chose 255 as the blank state value).
2. Set goal cell value to 0 and add it to the queue.
3. While the queue is not empty:
  - (a) Take front cell in queue “out of line” for consideration.
  - (b) Set the values of all blank<sup>1</sup> and accessible<sup>2</sup> neighboring cells to the front cell’s value +1
  - (c) Add the modified cells to the queue
  - (d) Else,continue

The 2D array of the maze is recalculated after every step the mouse makes. It is also worth mentioning that the starting position of the mouse is always in the top left corner of the maze.

<sup>1</sup>cell value is set to 255.

<sup>2</sup>accessible - there is no wall between the neighbor and currently considered cell.

Additionally, at the start, each direction is assigned a cardinal direction (east, west, north, south). By default, at the start, the front direction of the robot is assigned to north and the rest of the directions are assigned accordingly. When the mouse discovers a wall, it memorizes the direction of the wall in that cell according to the cardinal directions. Furthermore, to optimize the search, whenever the mouse discovers a wall it knows that this wall (inner wall) belongs to two adjacent cells and therefore if, for example, it discovers an east wall in cell (0,1), it knows that it is also the west wall of cell (1,1). This way, it is easy to track the inner walls positions of each cell. For this to operate properly, the mouse updates the cardinal direction of its front after every step.

Figures 3-10 demonstrate an example of a maze solving process. In the demonstration the goal cell was chosen to be coordinate (1,2) but due to our implementation of the solving algorithm and its helper functions we can choose any cell of the maze as the goal cell:



(0,0) 255 	(0,1) 255	(0,2) 255	(0,3) 255
(1,0) 255	(1,1) 255	(1,2) 255	(1,3) 255
(2,0) 255	(2,1) 255 	(2,2) 255	(2,3) 255
(3,0) 255	(3,1) 255	(3,2) 255	(3,3) 255

Figure 3: Starting position of the mouse. The mouse holds in memory blank cells, location of maze border walls and the inner walls (In red) of the maze in the starting position. Each cell is given a coordinate (x,y) starting from (0,0) where x grows downward and y grows to the right. The mouse only knows about the existence of the maze border walls the inner walls that are painted in red (the black inner walls are unknown to the mouse).

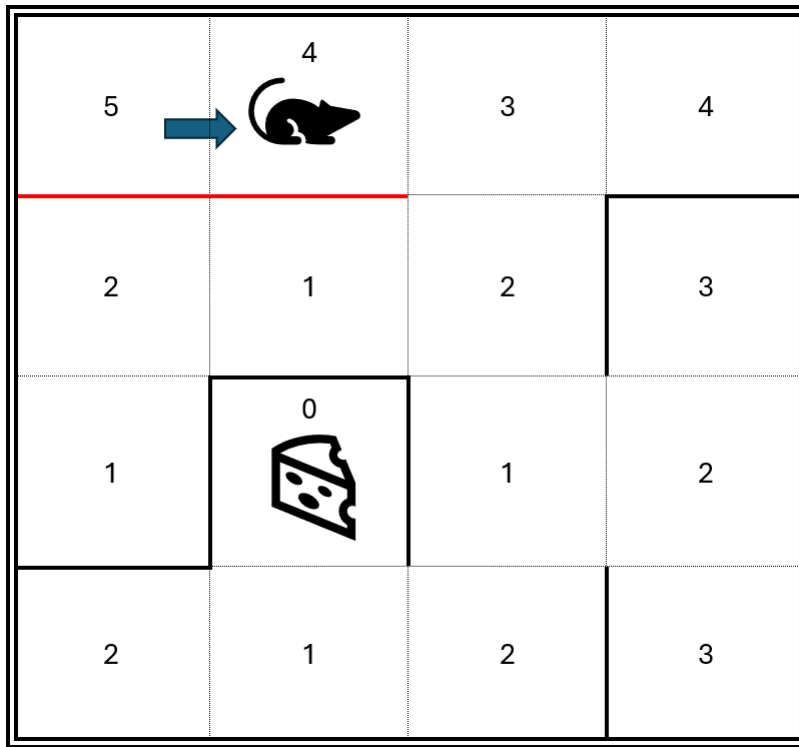


Figure 4: The maze is now mapped based on the information the mouse has on the inner walls. Each cell is assigned a value corresponding to its manhattan distance to the goal cell based on current information.

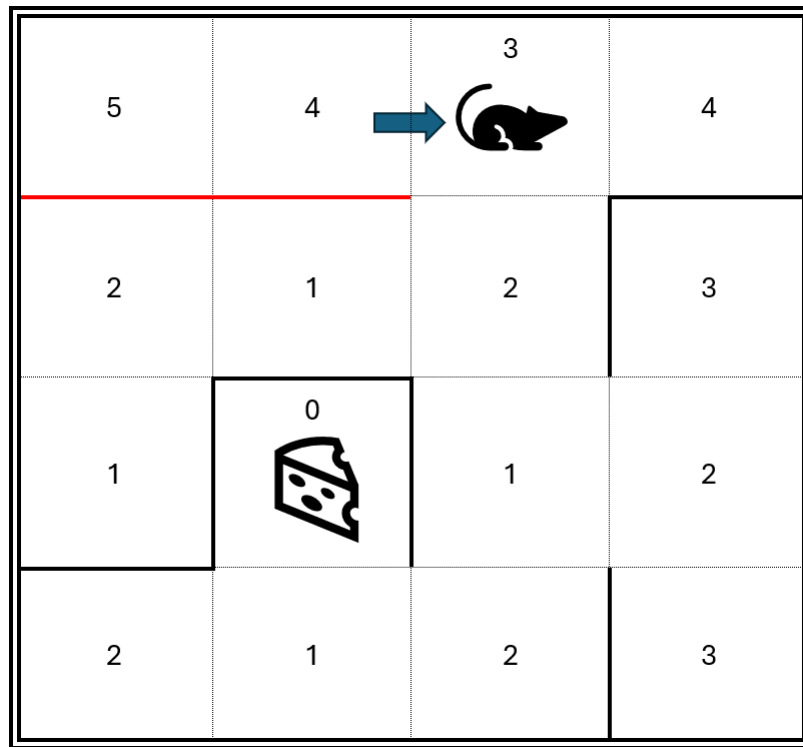


Figure 5: Second step, going in descending cell value order. This step didn't add new information and therefore the mapping of the maze remains unchanged compared to the previous step in figure 4.

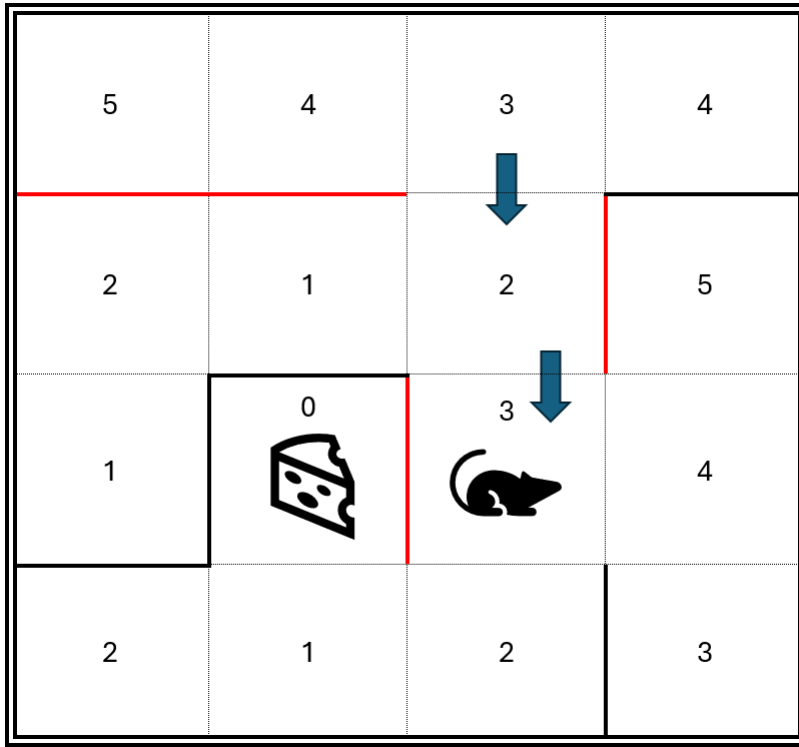


Figure 6: Third and fourth steps. Two new walls are discovered and the mapping is updated accordingly

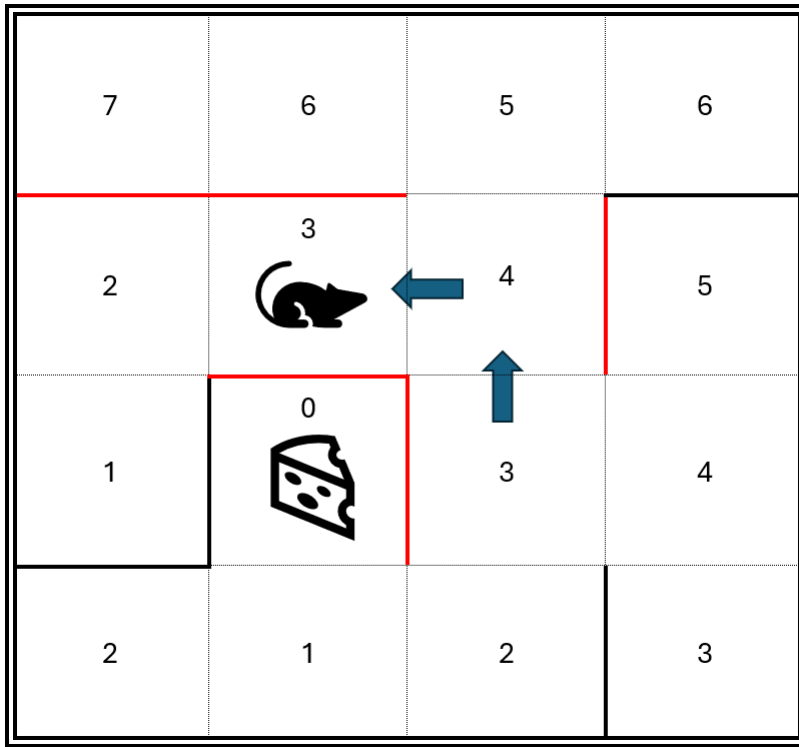


Figure 7: Due to built in bias, the mouse turns around and tries to reach the goal from another path instead of continuing on its course downward and toward the goal. It discovers another wall and updates the map accordingly.

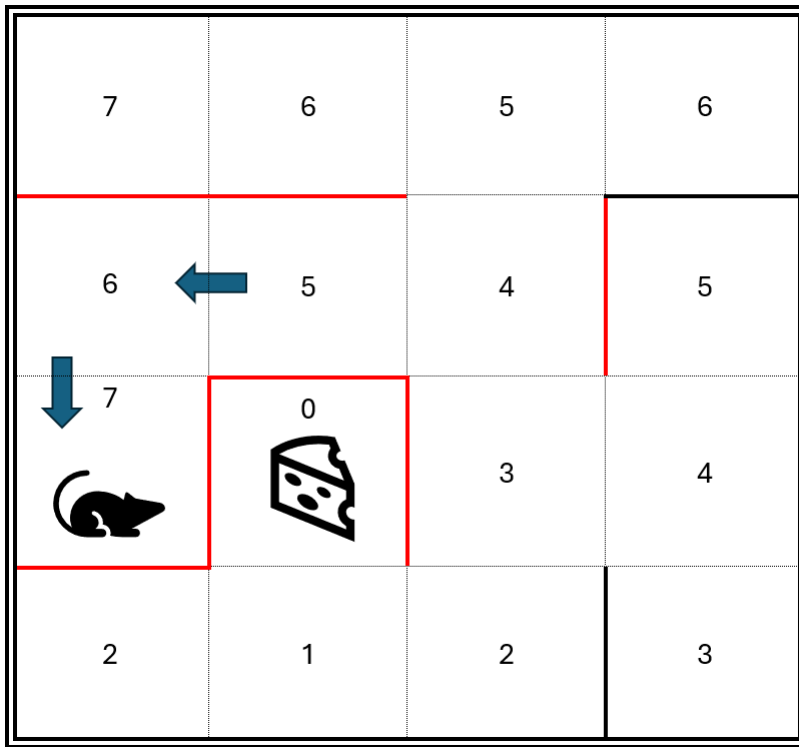


Figure 8: The mouse continues on course towards the goal, discovers two more wall and updates the map accordingly.



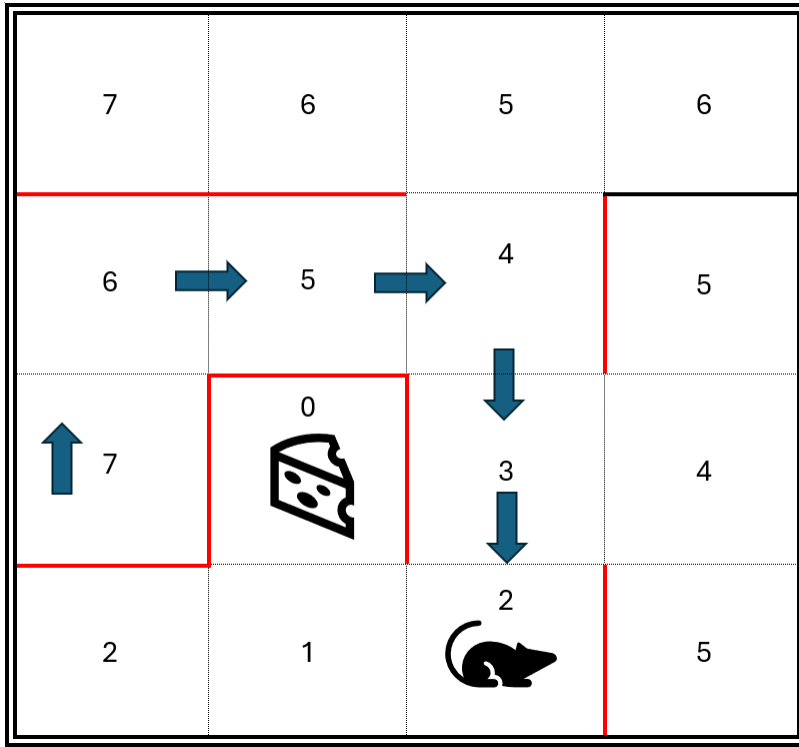


Figure 9: The mouse finally realizes the correct path and retraces its steps. On its way to the goal it discovers another wall and updates the map accordingly.

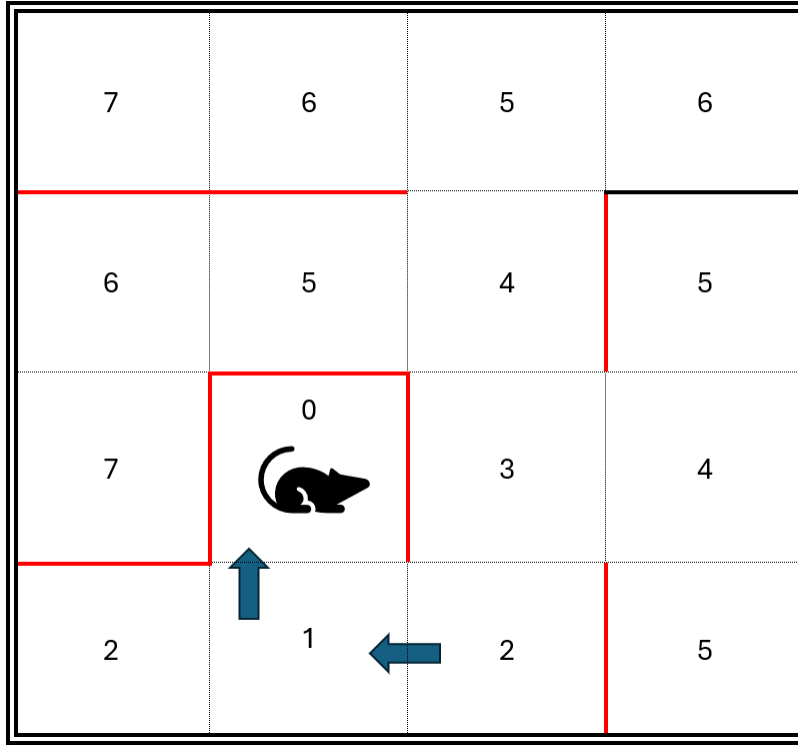


Figure 10: The has reached its goal and found the shortest path to it as a consequence if the mapping process. You may notice that it never discovered the wall in (0,3). This is the of the algorithm's efficiency. In larger mazes, even though the mouse has reached the goal it might still think there is a shorter path. Therefore, for maximal efficiency we would switch the start and goal cells and have the mouse find the shortest path to the start. By doing this we can ensure we find the absolute shortest path between the goal and the start.

Since we built a 4x4 maze we are guaranteed to find the shortest path on the first pass. On larger mazes it is better to make the mouse go back to the start and keep searching for the shortest path. The necessary changes larger mazes are noted in the code as a comment, we are confident that applying those changes will work smoothly on a larger maze though we never got to test the mouse on a larger maze. Once the shortest path is found, we apply changes to the movement functions to have the mouse travel to the destination as fast as possible on the next run.

## Communication

According to the competition rules, the mouse is not allowed to be remote controlled. Unfortunately we do not fully meet this requirement as we have not installed buttons on our micromouse that would allow us to initiate the maze solving algorithm. As it stands, the mouse has to receive the solving command by UART from a computer via Bluetooth. From the moment of receiving the command, the mouse works on its own and is not dependent on the computer for the solution process.

## Chassis and parts

Most of the parts used in the project are COTS. The only exception being the PCB that was designed and given to us by the project instructor. The entire chassis is 3d printed as it was fast, easy and readily available making the testing of the fit and function of the parts quick and mostly painless. The full BOM is as described in table 2.

Item NO.	Part[#reference]	Manufacturer	QTY
1	Back plate	3D printed	1
2	PCB bracket	3D printed	2
3	Cover	3D printed	1
4	Sensor bracket	3D printed	3
5	motor bracket	3D printed	2
6	IR sensor GP2Y0A51SK0F [3]	Sharp	3
7	DC motor N20 6V 500RPM [4]	-	2
8	Wheel-32x7mm	pololu	2
9	Dummy Wheel	-	1
10	Battery pack	-	1
11	PCB	Technion	1
12	Microcontroller - dsPIC33FJ64GS608 [1]	Microchip	1
13	H-Bridge - TLE9201SG [2]	Infenion	2
14	UART module - HC-05 [5]	ITEAD	1
15	M3 Hex head screw	-	3
16	M3 Philips head screw	-	3
17	M3 Flat head screw	-	4
18	M4 Philips head screw	-	4
19	M2 Hex head screw	-	8
20	Rechargeable batteries	-	2

Table 2: BOM

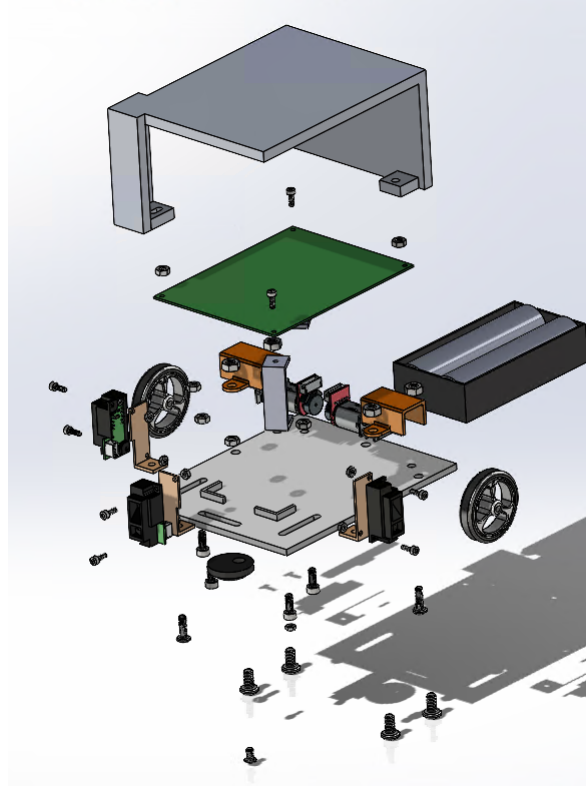


Figure 11: Exploded view of the final assembly

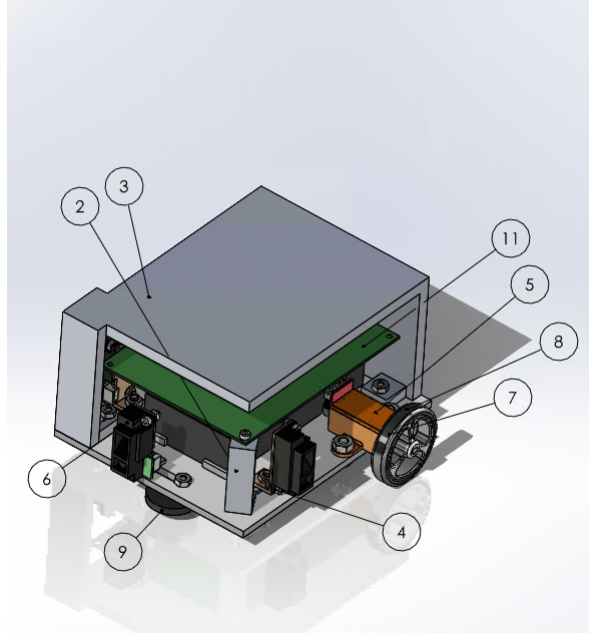


Figure 12: Fully assembled and numbered parts according to table 2

## Validation and Verification

In order to test the mouse’s ability to solve the maze (design requirement 2 and 3) we built a 4x4 (cells) demonstration maze. As we stated before, the size is small due to the need for the maze to be portable. In order to comply with competition rules, each unit cell is 18x18cm, the walls of the maze are 5cm in height and approximately 1cm thick.

This mostly complies with the rules, as we stated them at the start of this report, the only specification that isn’t with compliance to the rules is the thickness of the walls as we chose to not be too strict with this requirement. The reason of this slight non compliance is the availability of the material we chose to work with for the walls. We made the walls out of foamed PVC boards which are available in 3mm or 5mm thickness. The walls were built in a “sandwich” formation and therefore we chose to work with the 3mm variant. This led to each wall being ~1cm in thickness after accounting for some extra thickness from the gluing process. Each wall “sandwich” is slotted from the sides to the mazes poles (figure 13) to create a side of the maze’s cell. The poles themselves are bolted to a wooden platform 80x80 cm which allows some slack on all sides to make transportation easier.

Item NO.	Part	Manufacturer	QTY
1	Wooden Platfrom	Carpentry shop	1
2	Maze poles	3D printed	25
3	Wall “sandwiches”	Hand cut and glued	24
4	M3 Philips head screws	-	25

Table 3: Maze BOM

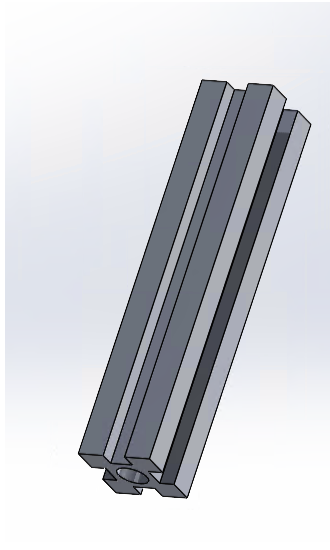


Figure 13: Maze pole

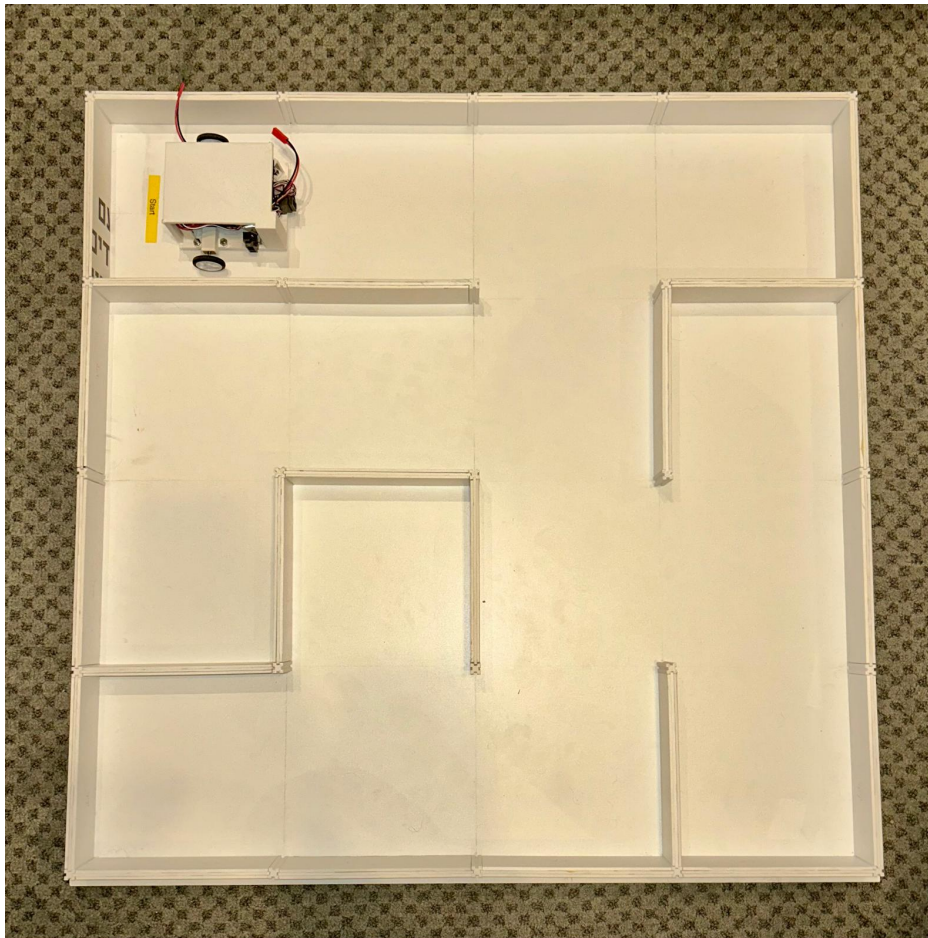


Figure 14: Maze final product

A successful result would be if the mouse can find the shortest path to the goal under various

configurations of the maze.

In addition verification of the mouses size (design requirement 1) is done with a ruler.

## Results and Discussion

During testing, the mouse was able to successfully traverse the maze and find the goal cell under various configurations of the maze as long as it was within the limitations that were discovered in testing. It correctly evaluated the Manhattan distance of all the cells to the goal and most importantly it was able to identify the shortest path to it. In other words, the mouse performs the exploration of the maze as intended. However some difficulties (the aforementioned limitations) arise when the mouse tries to take a turn to a cell that doesn't have at least one wall on one of it's sides. In some of the tests if the mouse's turning axis isn't perfectly centered with the cell's center, after turning to such a cell (no side walls), it may over correct to one of the sides and stop shortly afterwards because it sees a wall of an adjacent cell and can't move forward. This is the result of the way we use walls to keep the mouse centered and we have not found a good way to resolve this problem. Therefore the limitation of our mouse is that all the maze's cells have to have at least one wall on the side in order for it to function properly. Furthermore the final product proved to be 13cm wide and 90.6cm long which complies with our design requirements. In addition, while the mouse does need to communicate with the computer via Bluetooth to start it's operation this largely does comply with the rules but probably won't be allowed in competition regardless. In summary the mouse's performance satisfies the design requirement with some limitations [7]. As such, it is not competition ready and serves as a prototype to learn from and build upon. A potential solution can be the addition of a gyroscope to help with the mouse alignment instead of relying on the IR sensors. Other improvements to performance can be achieved by redesigning the PCB such that the motors can be attached directly to it which will save space and make turning easier, removal of the heavy rechargeable battery in favor of a light LiPo battery will remove a lot of weight and, in combination with the redesigning of the PCB, will remove the need for a chassis completely thereby reducing overall size.

## Part drawings and references

[1] Microchip DSPIC33FJ64GS608 Micro-controller.

[online] <https://www.microchip.com/en-us/product/dsPIC33FJ64GS608#document-table>

[2] 6A H-Bridge TLE9201SG.

[online] [https://www.infineon.com/dgdl/Infineon-TLE9201SG-DS-v01\\_00-en.pdf?fileId=db3a304345087709014518190f48](https://www.infineon.com/dgdl/Infineon-TLE9201SG-DS-v01_00-en.pdf?fileId=db3a304345087709014518190f48)

[3] Sharp GP2Y0A51SK0F IR distance sensor.

[online] [https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a51sk\\_e.pdf](https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a51sk_e.pdf)

[4] DC motors 6V 500RPM.

[online] [https://www.aliexpress.com/item/1005004999529855.html?spm=a2g0o.order\\_list.order\\_list\\_main.53.304e180291](https://www.aliexpress.com/item/1005004999529855.html?spm=a2g0o.order_list.order_list_main.53.304e180291)

[5] Bluetooth UART module HC-05.

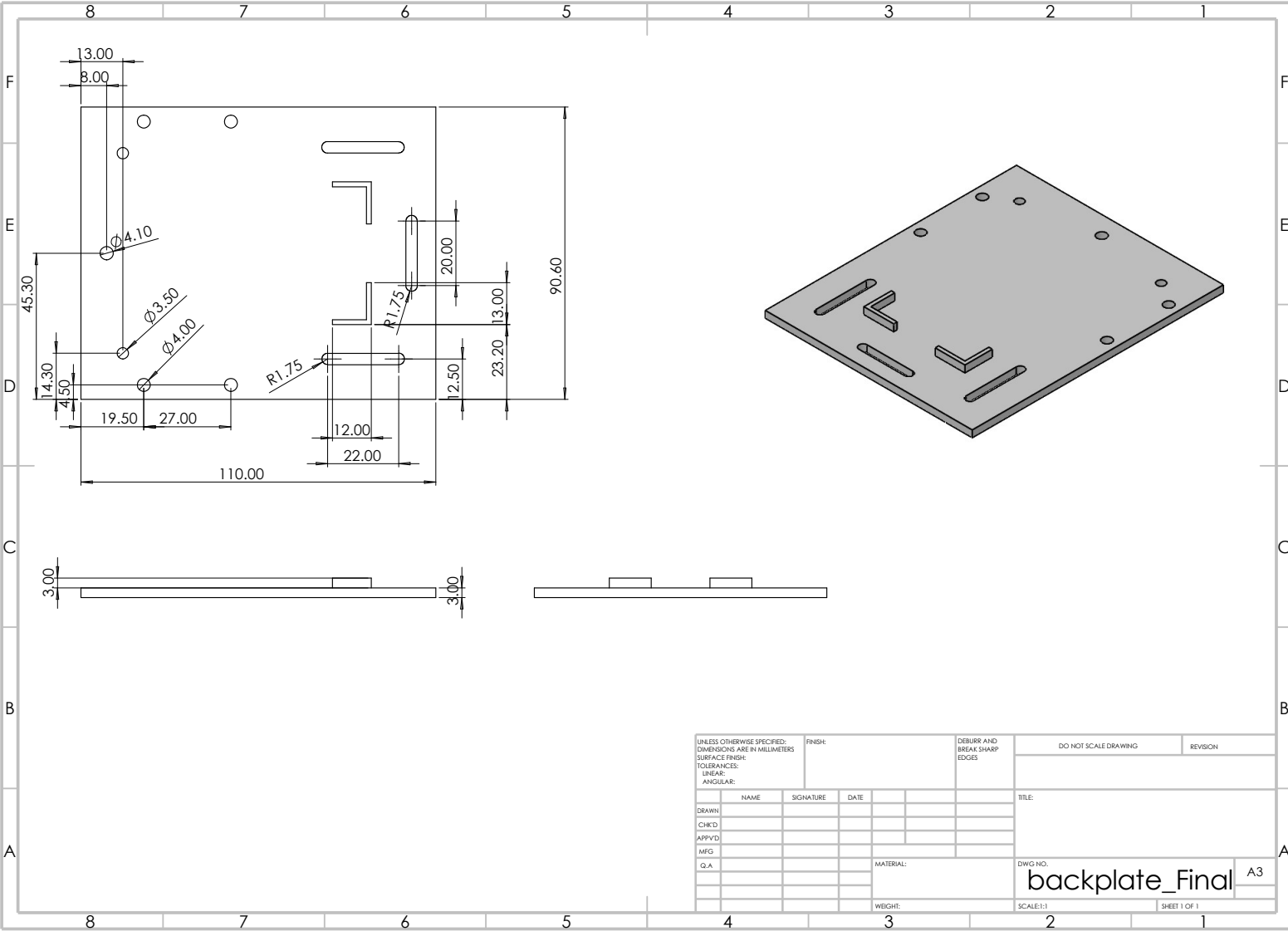
[online] [https://components101.com/sites/default/files/component\\_datasheet/HC-05%20Datasheet.pdf](https://components101.com/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf)

[6] Competition rule book.

[online] <https://r1.ieee.org/wp-content/uploads/2020/02/Micromouse-Guideline-2020.pdf>

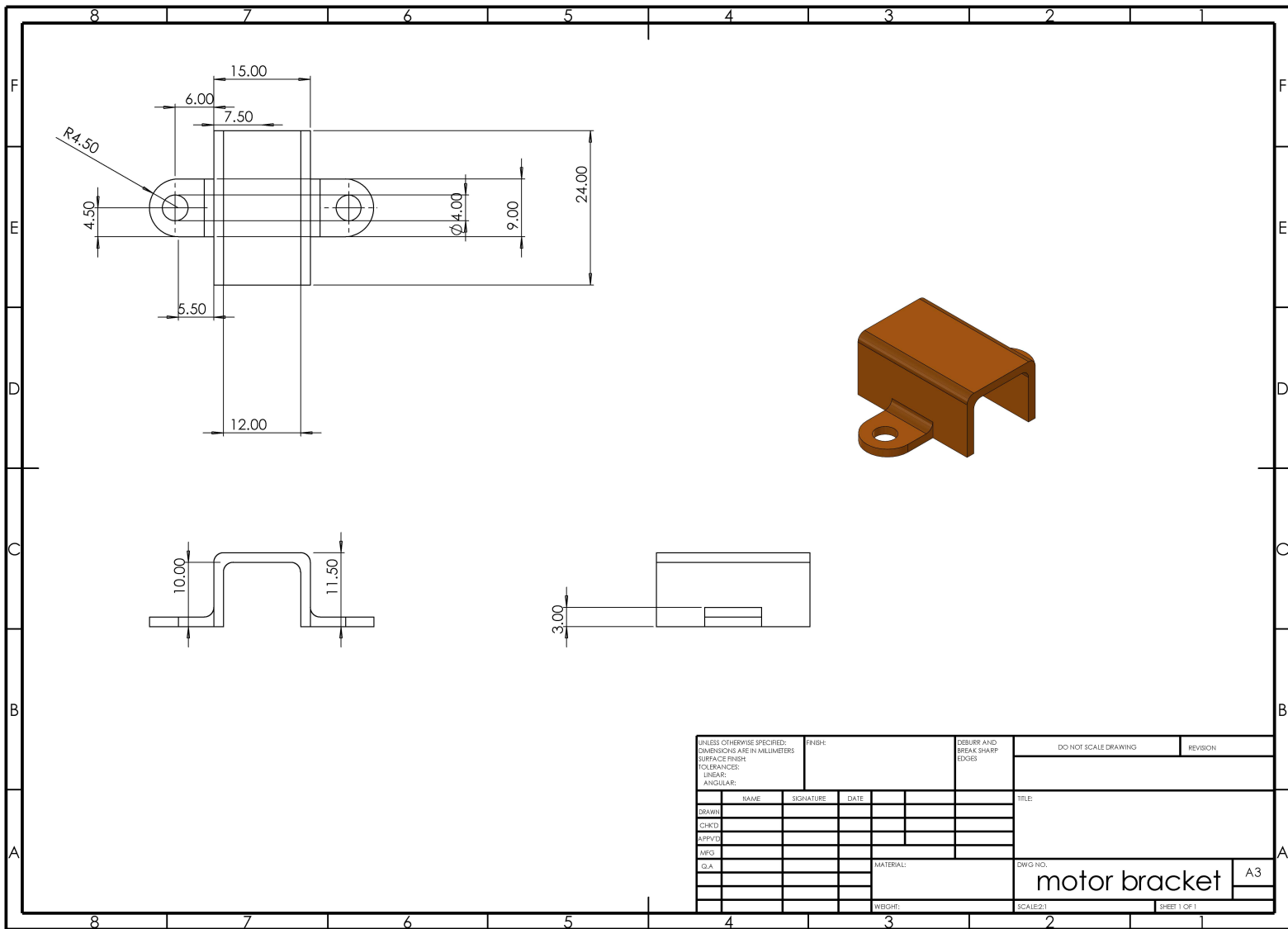
[7] First successful run.

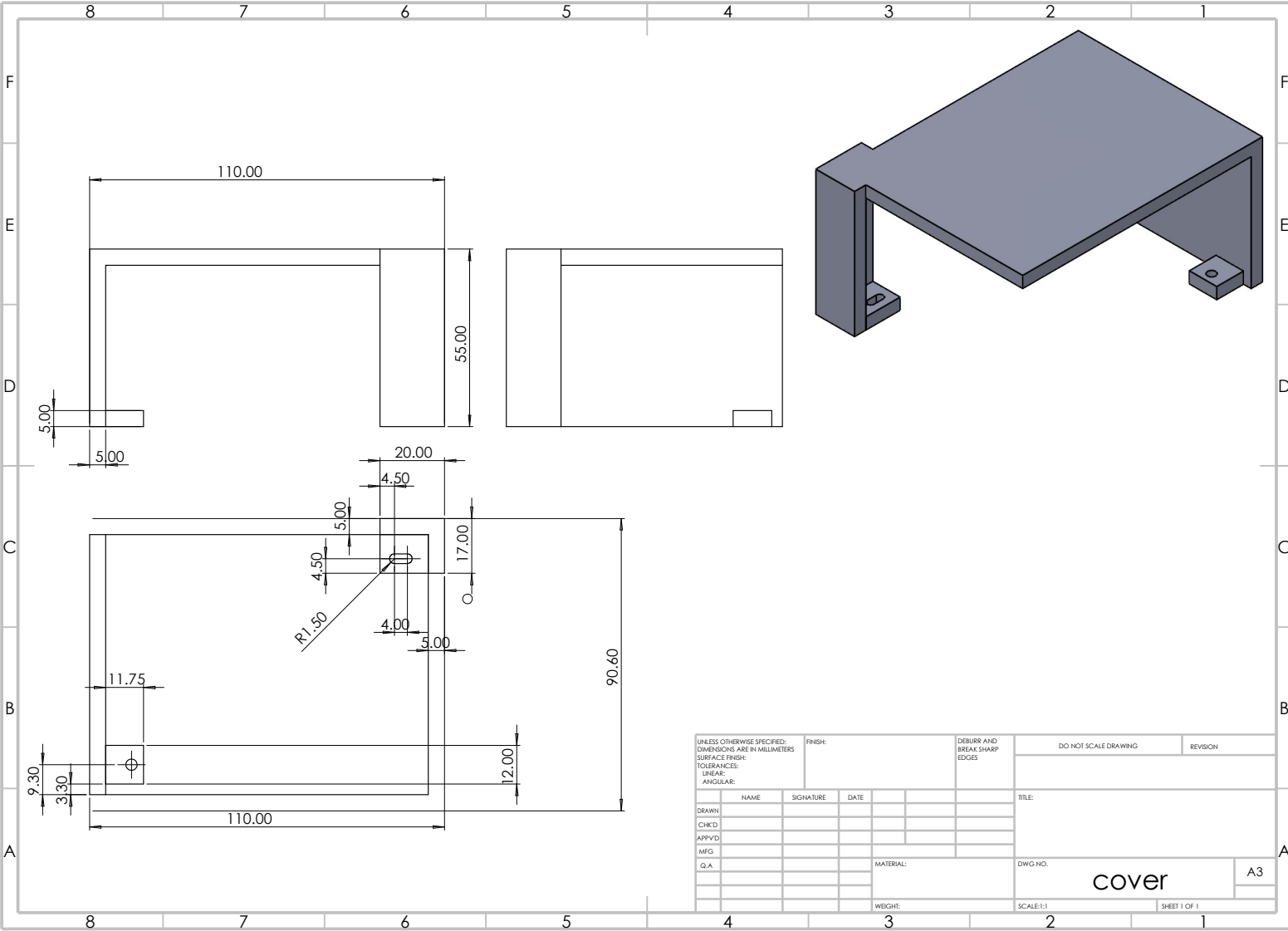
[online] <https://www.youtube.com/shorts/OQ9yuvwSb1w>





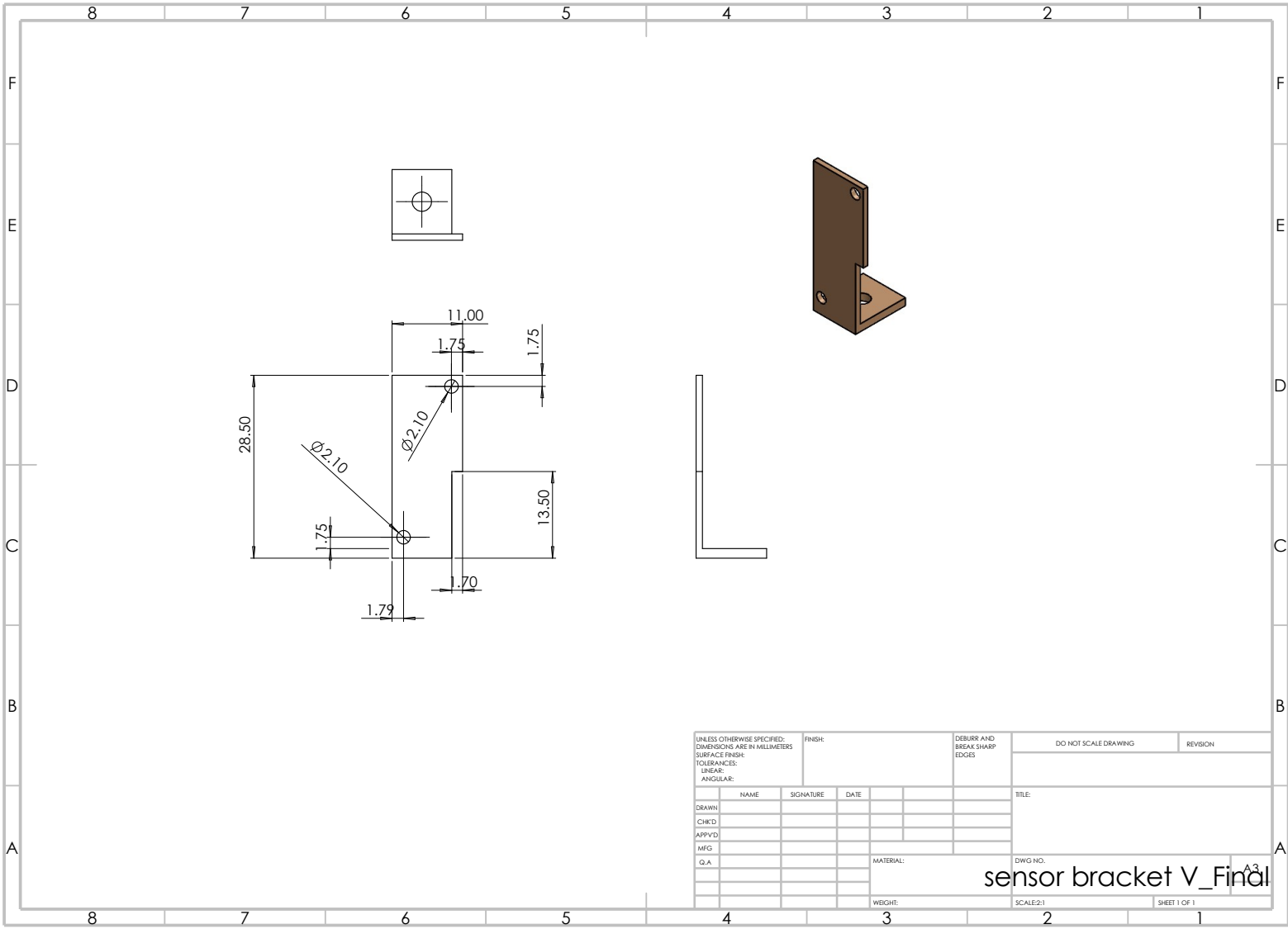






UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:				FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
DRAWN	NAME	SIGNATURE	DATE					TITLE:	
CHWD								DWG NO. <b>cover</b> <span>A3</span>	
APPRD									
MFG									
Q.A.									
						MATERIAL:		SCALE: 1:1	SHEET 1 OF 1
						WEIGHT:			

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS				FINISH:		DO NOT SCALE DRAWING		REVISION	
TOLERANCES: LINEAR: ANGULAR:				DEBURR AND BREAK SHARP EDGES		<div style="border: 1px solid black; padding: 10px; min-height: 150px;"> <p>TITLE:</p> <p style="font-size: 2em; margin-top: 20px;">pcb bracket_V3</p> </div>			
NAME		SIGNATURE		DATE					
DRAWN									
CHECKED									
APPROVED									
MFG						<div style="border: 1px solid black; padding: 10px;"> <p>DWG NO.</p> <p style="font-size: 1.5em; margin-top: 10px;">A3</p> </div>			
Q.A									
				MATERIAL:		<div style="border: 1px solid black; padding: 10px;"> <p>SCALE:2:1</p> <p>SHEET 1 OF 1</p> </div>			
				WEIGHT:					



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS				FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
SURFACE FINISH:									
TOLERANCES:									
LINEAR:									
ANGULAR:									
DRWN:	NAME	SIGNATURE	DATE					TITLE:	
CH'D:									
APP'D:									
MFG:									
Q.A.									
				MATERIAL:		DWG NO.		A3	
				WEIGHT:		SCALE:2:1		SHEET 1 OF 1	

sensor bracket V\_Final

