Robotics

# Deshret

Team Name: ***Deshret***

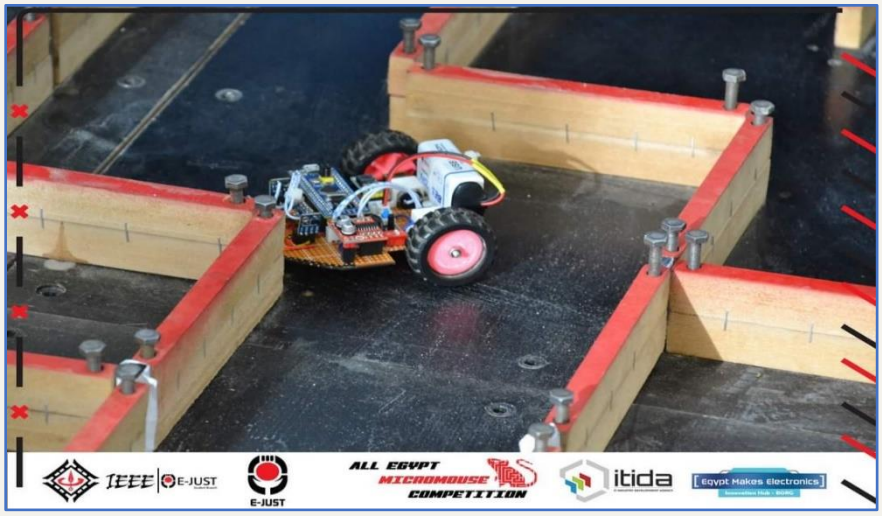
Team Members:

1- Mohamed Ehab Orfy
2- Ziyad Eslam
3- Nour ElDeen Shalaby
4- Ali El Bishbishy
5- Samaa Alaa

# Contents

# Introduction:

Micromouse is a competition where a small robot solves a maze in the fastest possible time. This competition contains different challenges whether mechanical, software and hardware challenges. We have participated the past year in AEMC (All Egypt Micromouse Competition) 2021 which was held in E-JUST and won the first place. We have acquired a massive experience from the past year which gave us lots of ideas to implement this year. This year we added new members to the team to benefit from the fresh ideas and the experience of the old members.



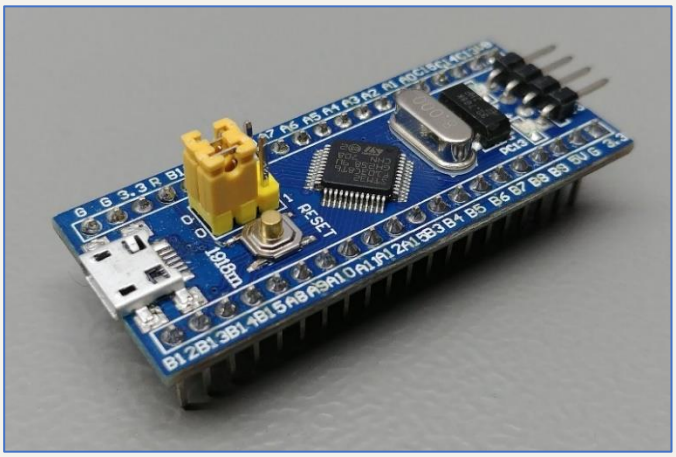

*Our past year design*

# Hardware

## Microcontroller

The microcontroller is the most important part of our robot. It controls the motors, sensors, and processes the whole algorithm. So, our choice was very tricky here. There was a large variety of MCUs but, at the end we settled on the STM32 blue pill. The Blue Pill is a development board based on ST Microelectronics STM32F103C8T6A microcontroller that has an ARM Cortex-M3 core that runs at 72MHz max. It characterized by its small form factor, ease of use and its large flash memory.

It was powerful enough for our robot compared to widely used microcontrollers such as the ( Arduino Nano ) as shown in the comparison below.



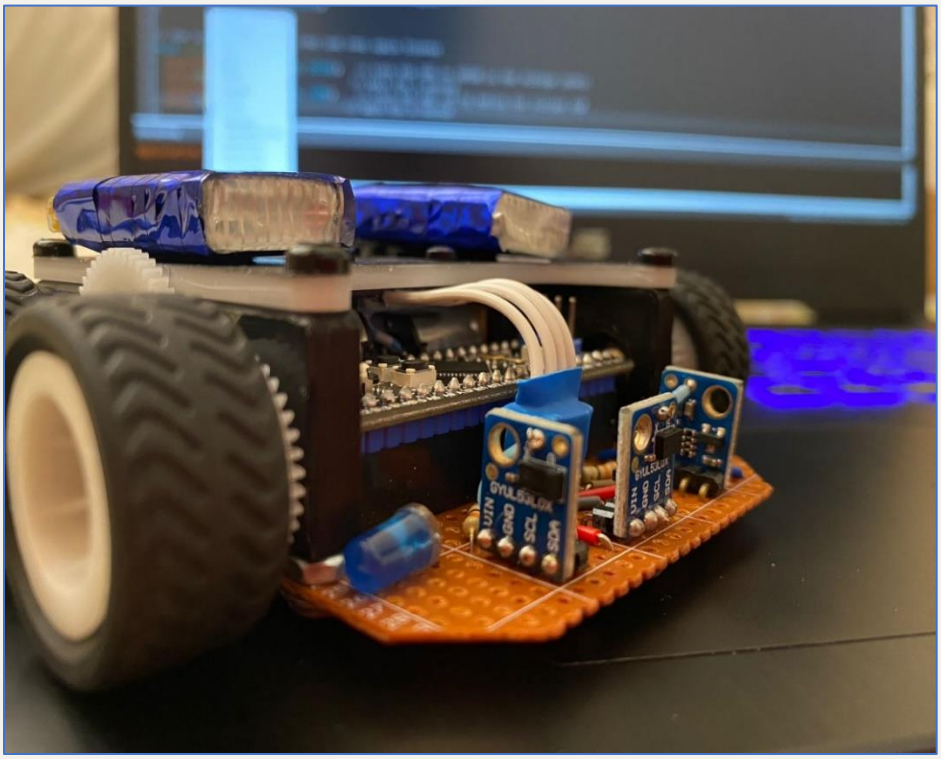
*STM32 blue pill*

| PRPOERTY | STM32F103C8T6 | ARDUINO NANO |
|---|---|---|
| | | |
| Clock Frequency | 72Mhz | 16Mhz |
| No.of I2C Buses | 2 | 1 |
| No.of SPI Buses | 2 | 1 |
| CAN BUS | Yes | No |
| No. of analog Channels | 10 | 8 |
| No. of PWM Channels | 15 | 6 |
| No. of USART Buses | 3 | 1 |
| No.of I/O's | 32 | 24 |
| On Board RTC | Yes | No |
| Architecture | ARM Cortex M3 32bit | AVR RISC 8bit |
| ADC Resolution | 12 bit | 10bit |
| Quantisation Levels | 4096 | 1024 |
| Flash Memory | 64KB | 32KB |
| SRAM | 20KB | 2KB |
| Debugging | Serial, JTAG | Serial |
| PWM resolution | 16bit | 10bit |

## Distance sensors:

It was a great challenge to find an accurate sensor with a small form factor to make the robot as small as possible. We decided to use the VL53L0X TOF sensor which has a range of measurements from 30 -2000 mm. This sensor has a great advantage beside its accuracy and range, it is not affected by external light sources.


*VL53L0X Sensor*



*Sensor alignment in our robot*

## Motors

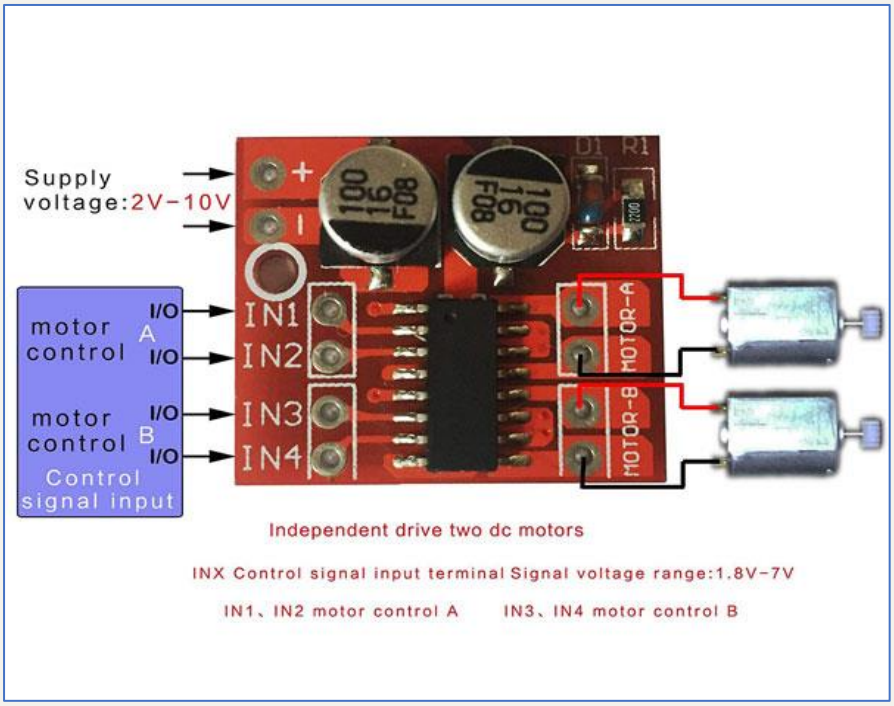
Our target was to choose a low power consumption, cheap and relatively fast motors. The choice was N20 micro metal gear lead screw motor. This motor draws stall current 160 mA which is suitable for our power consumption target. Also, the dimensions of this motor are very small 25 x 12 mm.

*N20 lead screw motor*

# Motor driver

We faced a challenge finding a small motor driver which is suitable with our motors. We chose the MX1616. Its specs:

- Continuous current 1.5A, max. 2.5A.
- Very small module only 25 x 21 mm.



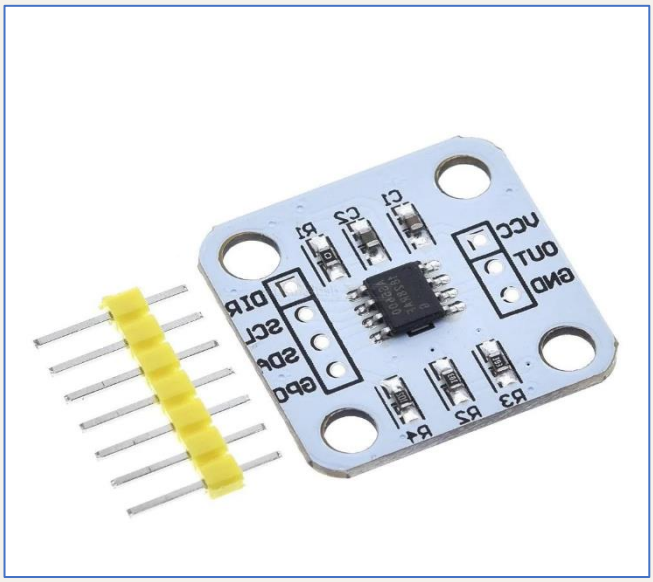

*MX1616*

# Encoder

**One of the biggest problems that we faced the last year was that we struggled to find encoders to mount on our motors. This year we found a suitable one which is the AS5600 which is a programmable 12-bit high-resolution contactless magnetic rotary position sensor. AS5600 can work as a magnetic potentiometer or a magnetic encoder with excellent reliability and durability. Compared with the traditional encoder, AS5600 has significant advantages: high precision, non-contact, no limits on rotation angle. All those advantages making it suitable for non-contact angle measurement applications, such as the robot arm, tripod head, motor closed-loop control, machine tool axis positioning.**

*Features:*

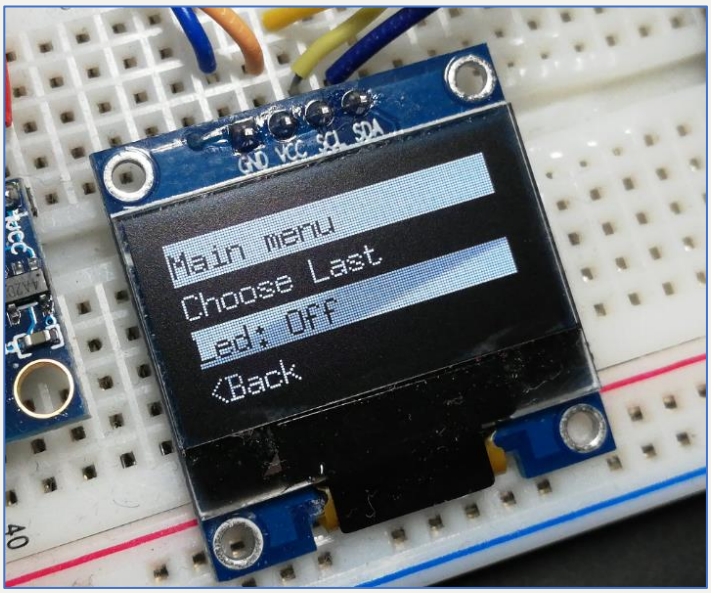
- Non-contact, no rotation angle limitation
- 12-bit high-resolution, 4096 positions per round
- I2C (digital) or PWM (Analog) Output




*AS5600*

# Debugging features

- OLED Screen:
  - An 0.96-inch OLED screen is used to debug our code and it is as an addition to our design. A simple menu is made which will be explained in the software part.




*0.96-inch OLED*

- Buzzer:
    - A piezoelectric passive buzzer is used to debug on our code and determine whether the functions is executed properly or not.
- Bluetooth module:
    - A Bluetooth module (HM-10) is used for debugging, monitoring sensor readings and tuning PID constants that would be explained below.

# Power

- Battery: After comparing and testing out the N20 motors with the Beston 9v rechargeable battery (That we used last year) & custom *Lipo* Battery cells Total (7.4v - 550 mAH) and measuring the motors RPM.

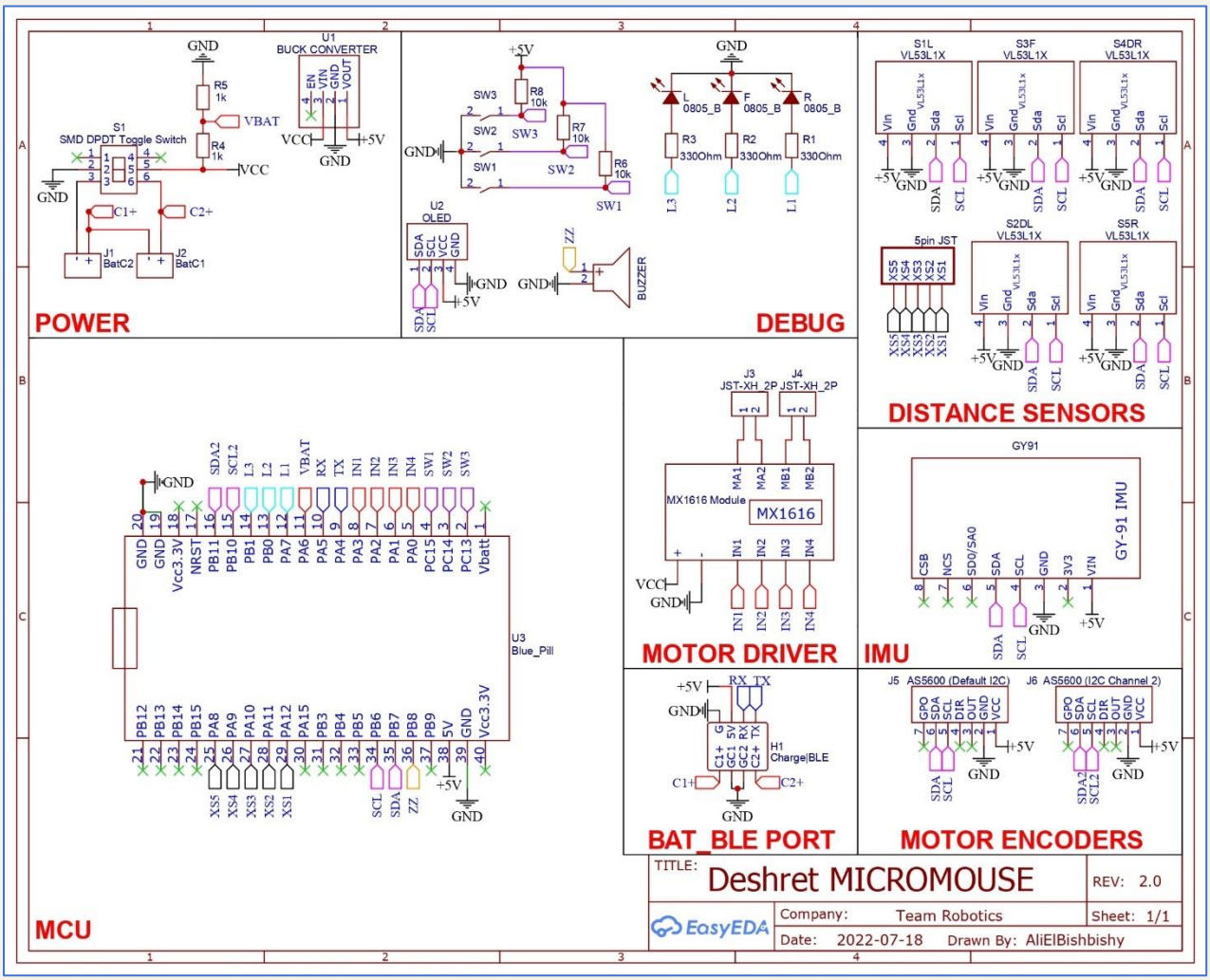
We decided to use the custom *Lipo* Battery cells. Due to its compact size and its weight and has the proper voltage and current ratings for our robot and runs the N20 motors with higher RPM.

*Custom LIPO Cells*

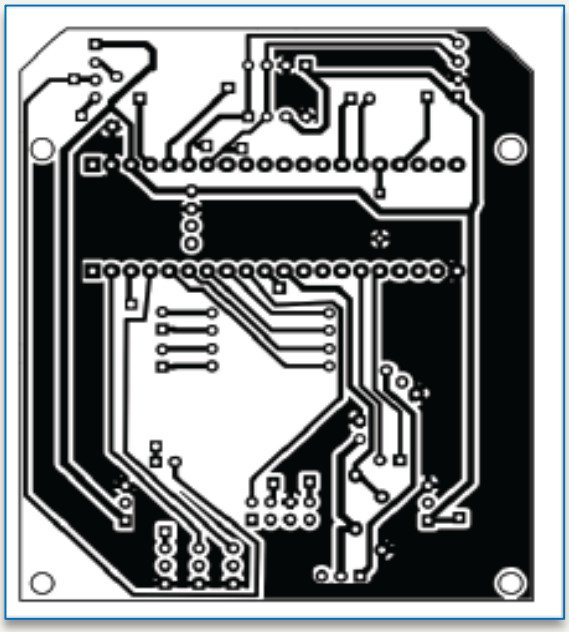

*Beston 9v rechargeable battery*

# PCB Design:

- Schematic design:




- Track-outlines



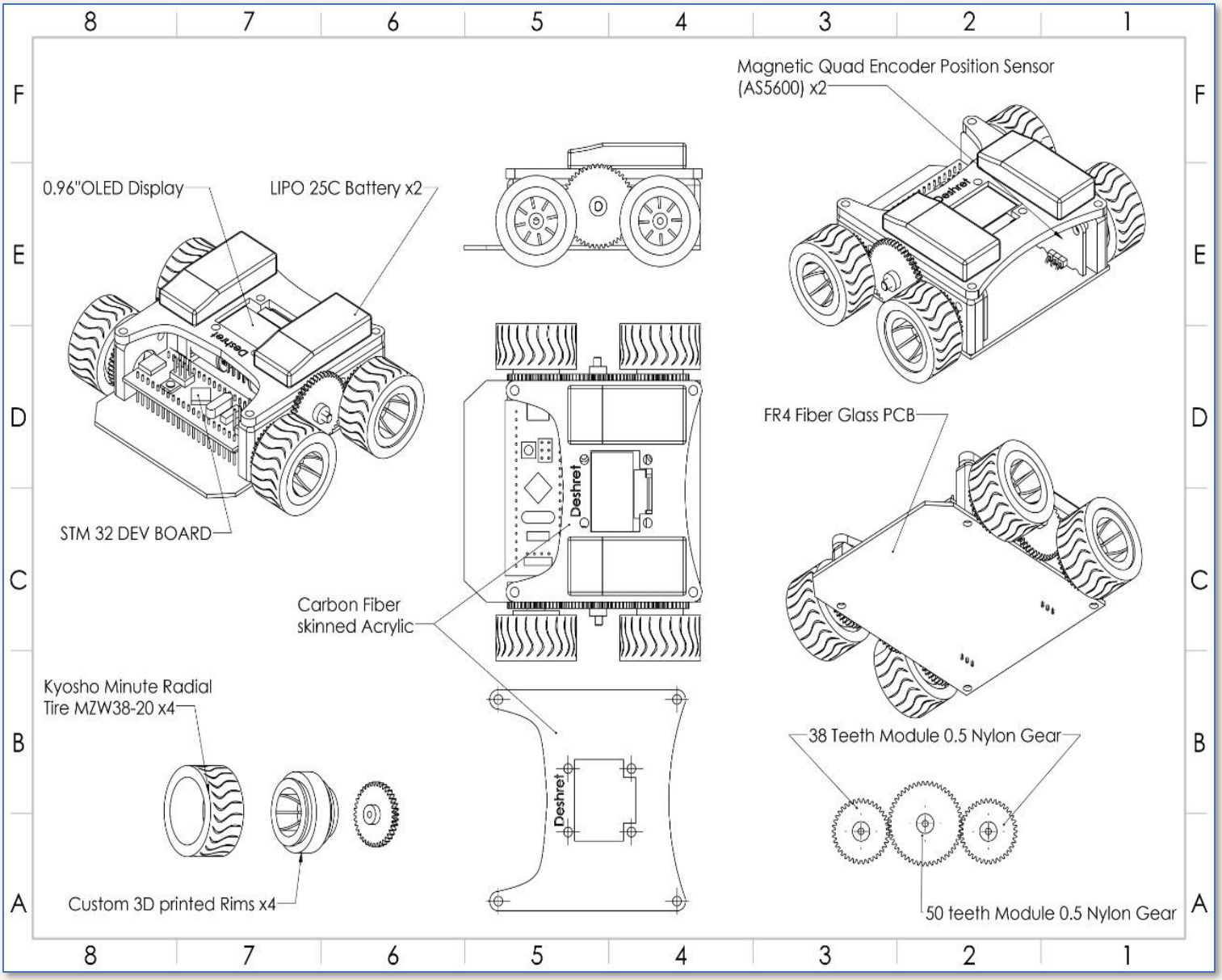
# Mechanical design:

## Robot Measurements:

Robot dimensions LxWxH: *82 x 103 x 39 mm.*

Weight: *170 grams.*

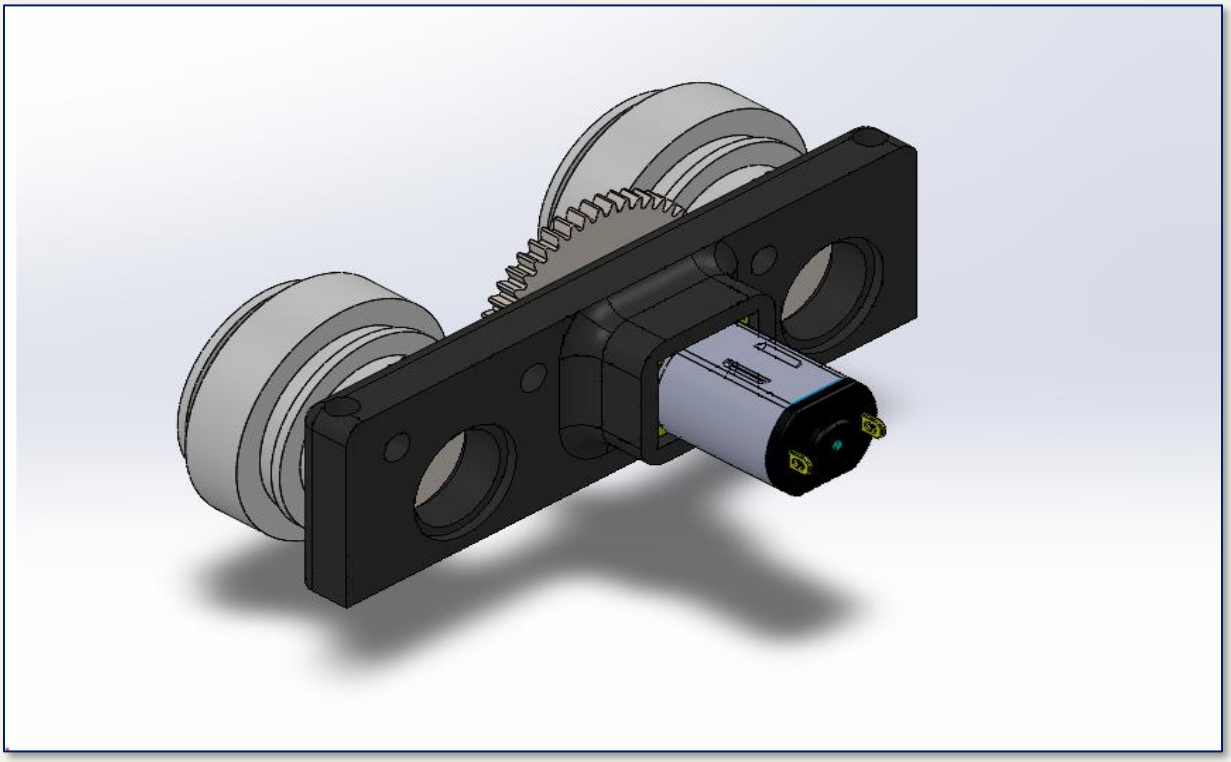
## Robot body:




*The Whole Robot Design*

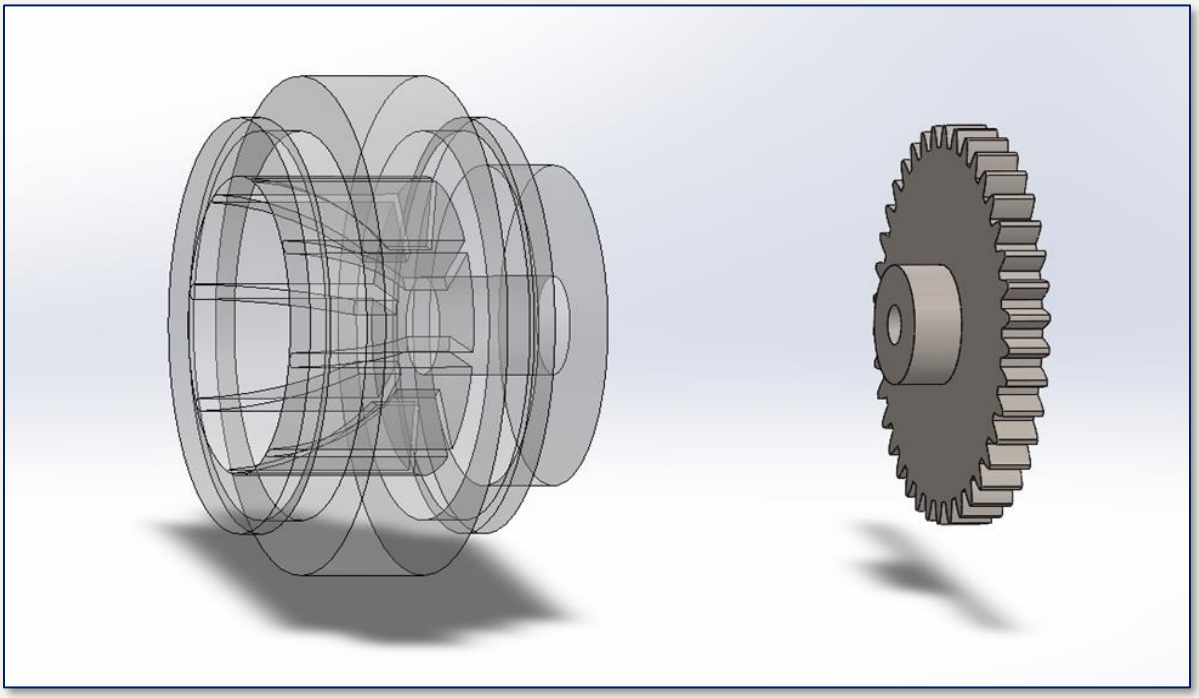
***We started by collecting as much data as possible from previous competitions***

- Our 260 RPM Motors are press fit (friction fit) into place



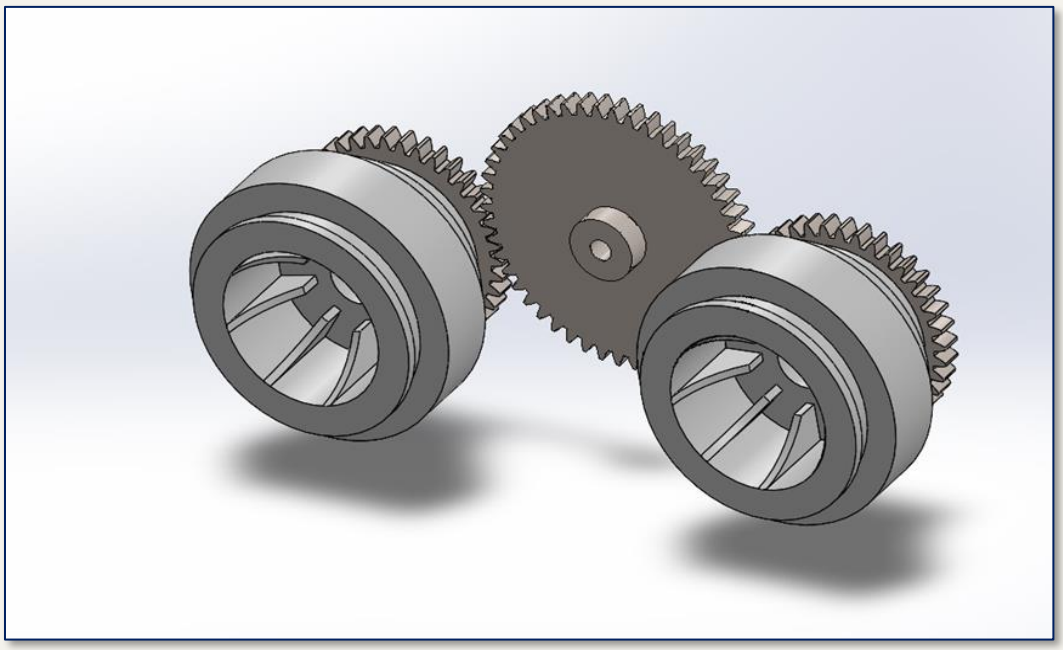
*Mini Z 28Ø mm tires*

- Both the Rims and the Motor mounts are 3D printed PLA plastic with 20% infill to keep it as strong and light as possible.

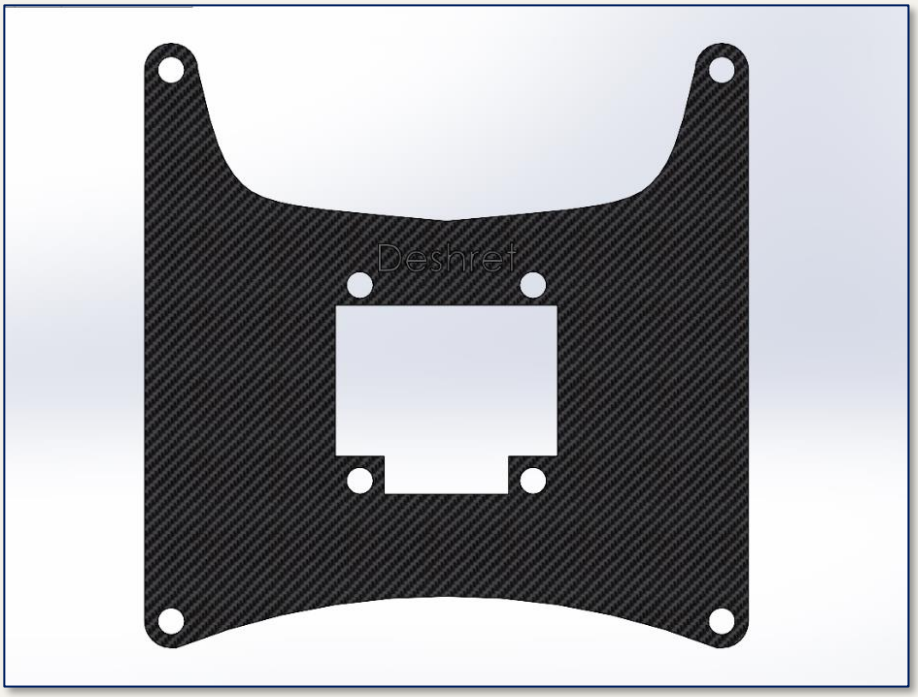


- The wheel gears' 6mm hub is press fit (friction fit) in the Rims
- We used off the shelf plastic gears with a 50:38 gear teeth ratio

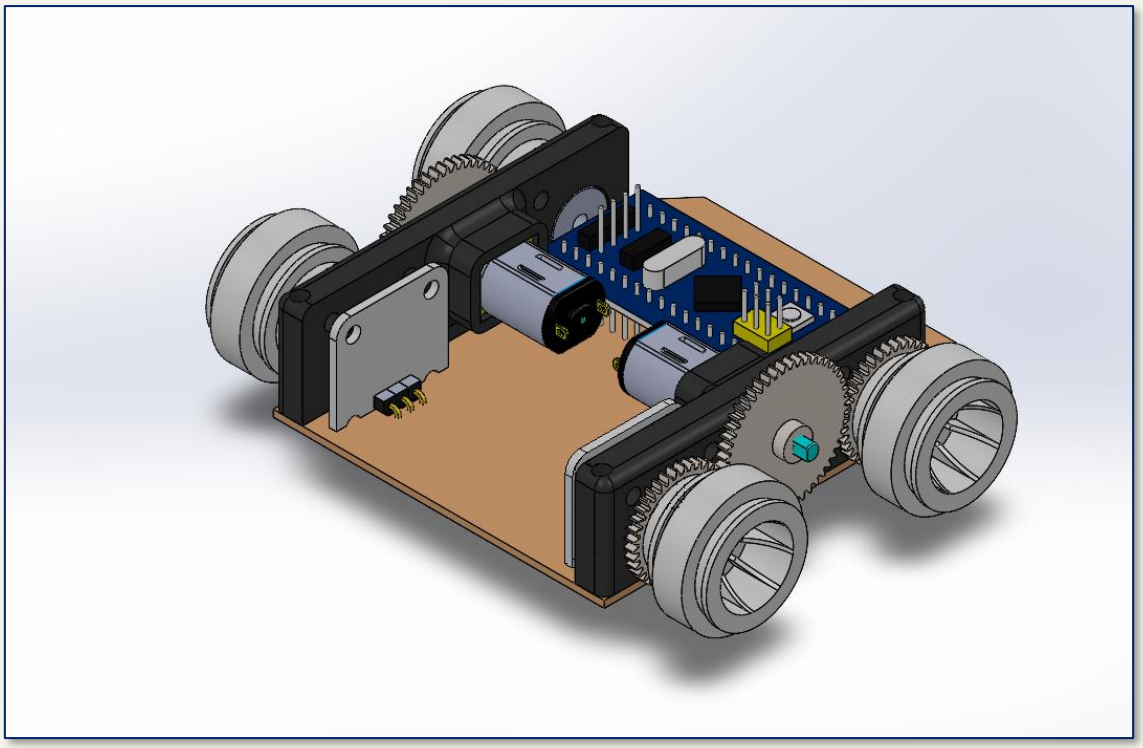
(Module 0.5 Ansi Metric/20 pressure angle /Face Width 2mm).



- X-shaped white acrylic structural support from the top that doubles as a holder for the OLED and reduces the load on the fiberglass PCB

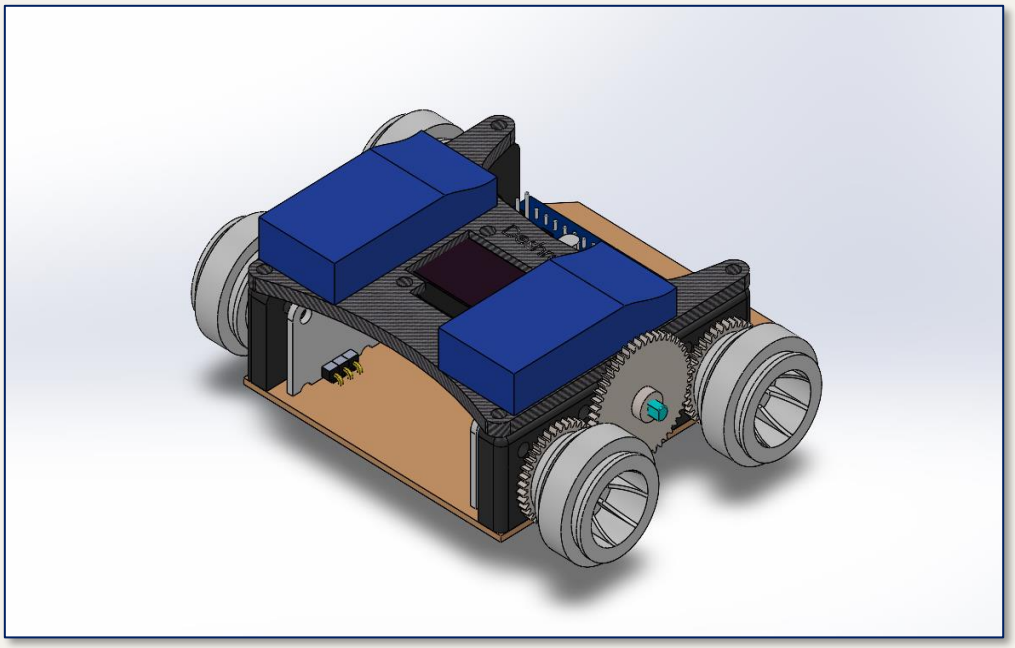


- All the structural members are held together by 4 M3 screws on opposing corners of the robot
- We used fzz64 flanged bearing on the wheels' axle to minimize friction
- The motor Mount keeps the encoder sensor concentric with the wheel shaft

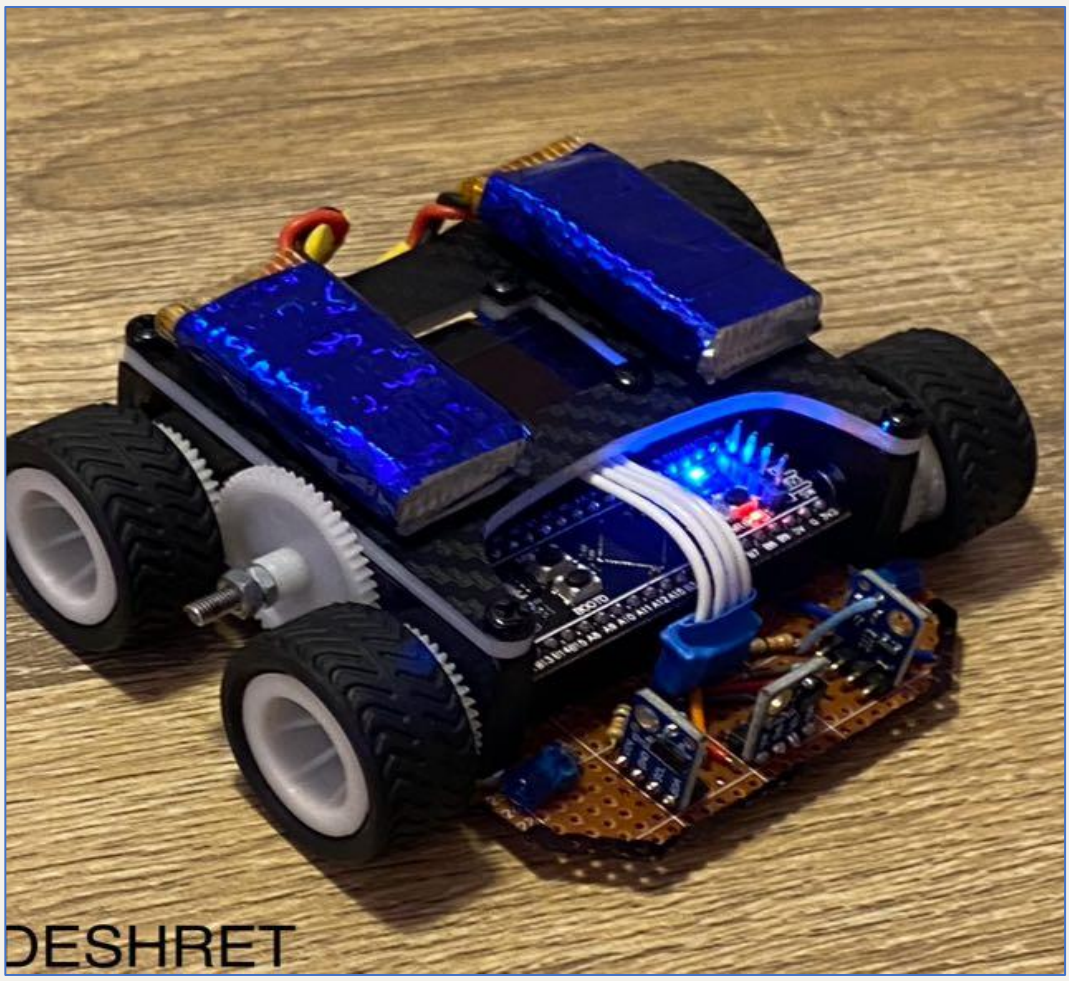


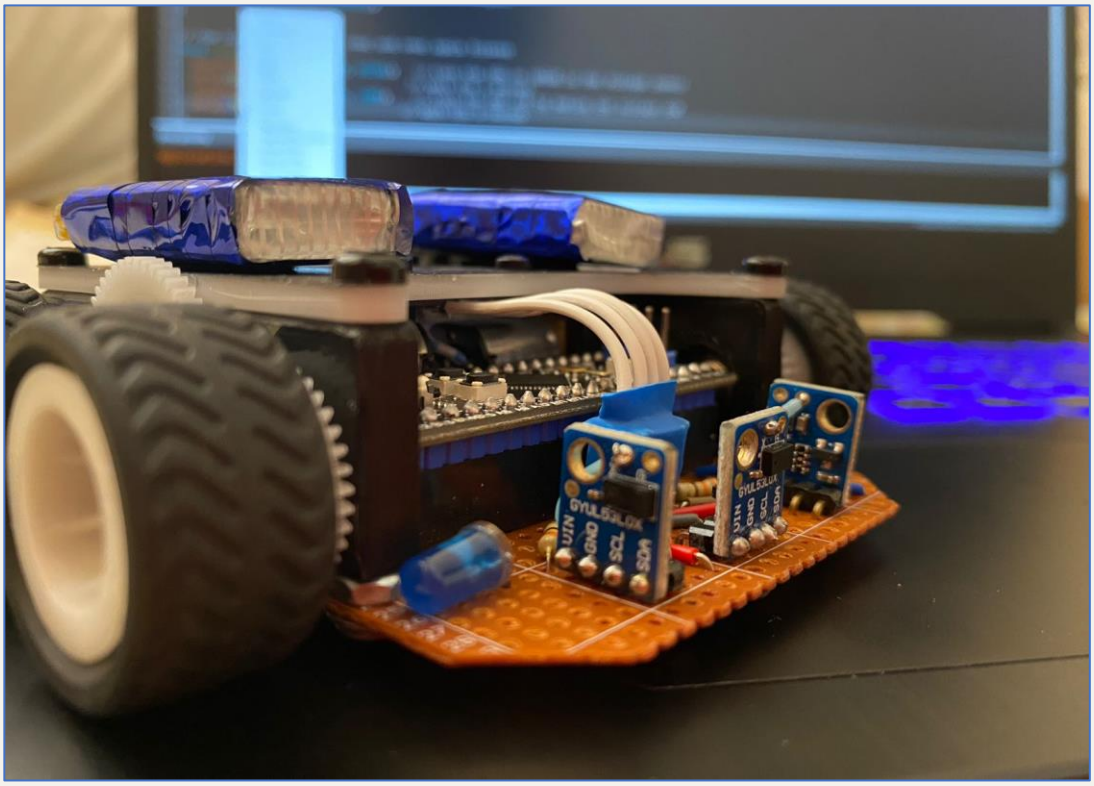
- We had to cut a part of the encoder module to add more clearance for the robot off the ground.

…Passing the design outline to the electrical hardware team



# Real-Live Photos:

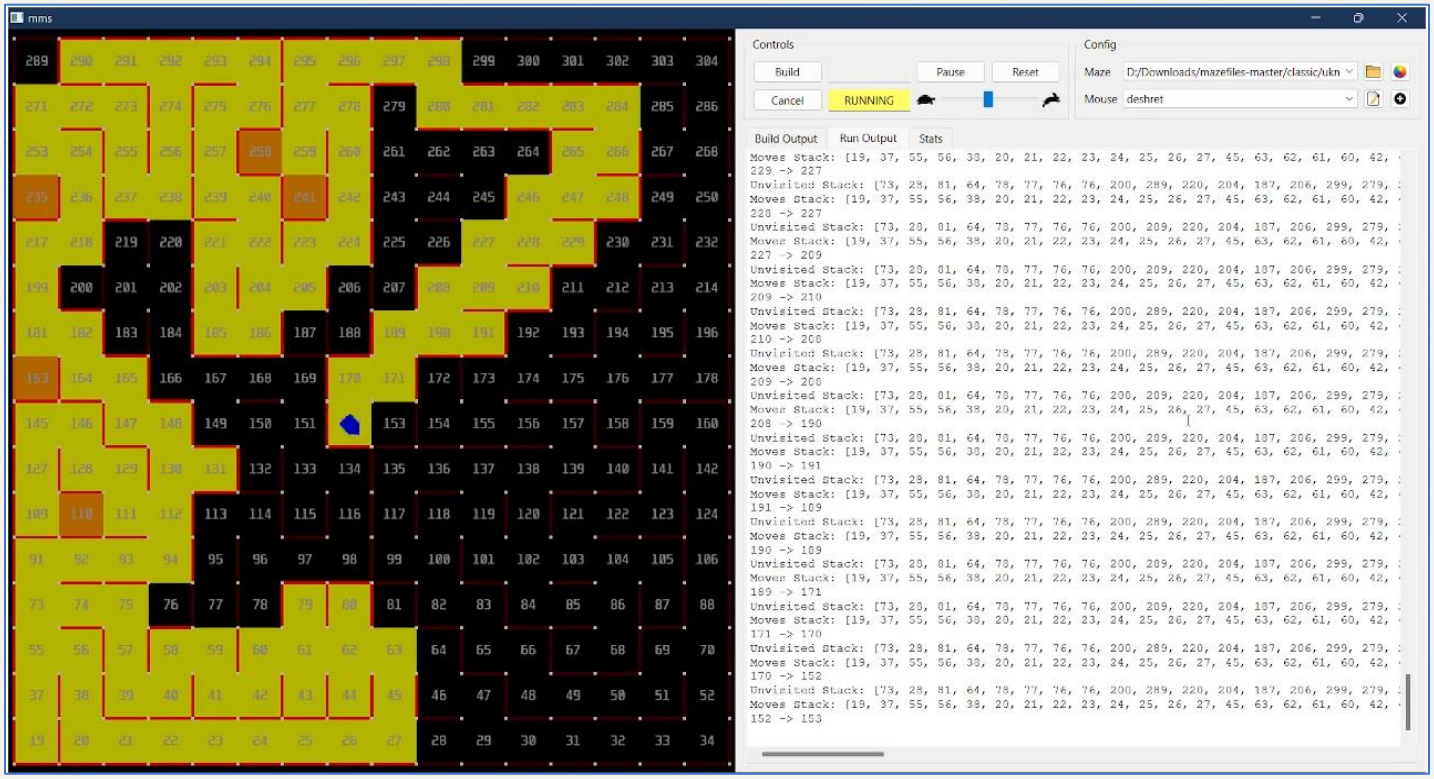


# Code and Algorithms:

- The robot first starts with the mapping phase. In this phase, the robot targets to discover all the cells of the maze. The mapping is done using Depth First Search approach. In this approach, the robot gets all adjacent unvisited cells accessible from the current cell, and adds them to a stack data structure. Then the robot takes the highest cell in the stack and goes to it, then it repeats the process again. Whenever the robot moves, it records its movement and adds it to another stack. When the robot finds no adjacent unvisited cells, it undoes its last moves from the moves stack until it gets adjacent to an unvisited cell. To save on mapping time, when the robot finds an unvisited cell that is adjacent to 4 discovered cells, the robot marks it as discovered immediately as it already knows its 4 sides.
- When the robot discovers all the cells, it returns to the start then starts the fastest route phase. The robot calculates the fastest route using Breadth First Search algorithm.

- The algorithm was tested and simulated by mms simulator as it is shown below:




*Robot mapping the maze*


*The Shortest path is figured out*


*Maze Solved*

- A simple menu is displayed on the OLED with various options to start the robot motion or resume the mapping or running the shortest path:


*OLED Display*

## 1- The algorithm and OlED display:

- Deshret-arduino.ino

```
#include <Arduino.h>
#include <U8g2lib.h>
#include <Wire.h>

U8G2_SSD1306_128X64_NONAME_F_SW_I2C u8g2(U8G2_R2, /* clock=*/ PB6, /* data=*/ PB7, /*
reset=*/ U8X8_PIN_NONE);
int batteryLevel = 99;

const char *string_list =
  "Start Over\n"
  "Resume Mapping\n"
  "Run Shortest Path";

uint8_t current_selection;

// variable initialization
int unvisited[324] = {0};
int unvisited_i = 0;
bool maze[324][4] = {false};
bool visited[324] = {false};
int moves[324] = {0};
int moves_i = 0;
int path[324] = {0};
int path_i = 0;
```

```
int path_j = 0;
int current = 19;
int orient = 0;
char directions[] = {'n', 'e', 's', 'w'};
int dir[] = {18, 1, -18, -1};

bool ex = false;
int ocurrent;

int p[324] = {0};

//functions
void loop2()
{

}

void setup()
{
  Serial.begin(115200);
  u8g2.begin(PC13, PC15, U8X8_PIN_NONE, U8X8_PIN_NONE, U8X8_PIN_NONE, U8X8_PIN_NONE);
  u8g2.setFont(u8g2_font_7x14_tr);
  pinMode(PC13, INPUT_PULLUP);
  pinMode(PC15, INPUT_PULLUP);

  // Setup LED Timer
  Timer2.setMode(TIMER_CH1, TIMER_OUTPUTCOMPARE);
  Timer2.setPeriod(500000);
  Timer2.setCompare(TIMER_CH1, 1);
  Timer2.attachInterrupt(TIMER_CH1, loop2);
}

void loop()
{
  if (ex)
  {
    ex = false;
    ackReset();
    ocurrent = current;
    current = 19;
    orient = 0;
  }
  String t = String(batteryLevel) + "%";
  if (t.length() == 2)
    t = "0" + t;
  String title = "DESHRET MK2    " + t;
  current_selection = u8g2.userInterfaceSelectionList(
                        title.c_str(),
                        current_selection,
```

```
                      string_list);
if (u8g2.userInterfaceMessage(
      "Are you sure?",
      u8x8_GetStringLineStart(current_selection - 1, string_list ),
      "",
      " Yes \n No ") != 1)
  return;
u8g2.clearBuffer();
u8g2.drawStr(0, 10, "Starting in 3...");
u8g2.sendBuffer();
delay(1000);
u8g2.clearBuffer();
u8g2.drawStr(0, 10, "Starting in 2...");
u8g2.sendBuffer();
delay(1000);
u8g2.clearBuffer();
u8g2.drawStr(0, 10, "Starting in 1...");
u8g2.sendBuffer();
delay(1000);
u8g2.clearBuffer();
u8g2.drawStr(0, 10, "Hold PC15 Button");
u8g2.drawStr(0, 24, "To Reset");
u8g2.sendBuffer();
if (current_selection == 1)
{
  for (int i = 0; i < 18; i++)
    for (int j = 0; j < 18; j++)
    {
      setColor(j - 1, i - 1, 'k');
      for(int k = 4; k--;)
        clearWall(j - 1, i - 1, directions[k]);
    }
  memset(unvisited, 0, sizeof(unvisited));
  unvisited_i = 0;
  memset(maze, false, sizeof(maze));
  memset(visited, false, sizeof(visited));
  memset(moves, 0, sizeof(moves));
  moves_i = 0;
  memset(path, 0, sizeof(path));
  path_i = 0;
  path_j = 0;
  current = 19;
  orient = 0;
  memset(p, 0, sizeof(p));
  phase();
}
else if (current_selection == 2)
{
  bfs(19);
```

```
    int v = ocurrent;
    path_i = 0, path_j = 0;
    while (v != -1)
    {
      path[path_i] = v;
      path_i++;
      v = p[v];
    }

    path_i--;
    sprint(path, path_j, path_i);

    phase();
  }
  else if (current_selection == 3)
  {
    phase2();
  }
}
```

- algorithm.ino

```
// function initialization
void bfs(int s)
{
  int q[324] = {0};
  int q_i = 0;
  int q_j = 0;
  bool used[324] = {false};
  int d[324] = {0};
  q[q_i] = s;
  q_i++;
  used[s] = true;
  p[s] = -1;
  while (q_i > q_j)
  {
    int v = q[q_j];
    q_j++;
    for (int i = 0; i < 4; i++)
    {
      if (!maze[v][i])
        continue;
      int u = v + dir[i];
      if (!used[u])
      {
        used[u] = true;
```

```
        q[q_i] = u;
        q_i++;
        d[u] = d[v];
        p[u] = v;
      }
    }
  }
}

void wall(int shift)
{
  int x = (4 + orient + shift) % 4;
  int y = current + (x < 2 ? 1 : -1) * (1 + ((x + 1) % 2) * 17);
  if (!visited[y])
  {
    unvisited[unvisited_i] = y;
    unvisited_i++;
  }
  maze[current][x] = true;
}

void go(int a, int l)
{
  if (!digitalRead(PC15))
  {
    ex = true;
    return;
  }
  if ((4 + orient - 1) % 4 == a)
  {
    turnLeft();
    orient = a;
  }
  while (orient != a)
  {
    turnRight();
    orient = (4 + orient + 1) % 4;
  }
  if (!digitalRead(PC15))
  {
    ex = true;
    return;
  }
  if (l)
    moveForward(l);
}

void sprint(int arr[], int start, int end)
{
```

```
  while (end > start)
  {
    for (int i = 0; i < 4; i++)
    {
      if (arr[end - 1] - current == dir[i])
      {
        int n = arr[end - 1];
        end--;
        int c = 1;

        while (end - start > 1 && arr[end - 1] - n == dir[i])
        {
          n = arr[end - 1];
          end--;
          c++;
        }

        go(i, c);
        current = n;
        if (ex == true)
          return;
        break;
      }
    }
  }
}

void phase()
{
  setWall(0, 0, 's');

  // marking off-border cells as visited
  for (int i = 0; i < 18; i++)
  {
    visited[i] = true;
    visited[i * 18] = true;
    visited[i + 306] = true;
    visited[i * 18 + 17] = true;
  }

  // numbering cells
  for (int i = 0; i < 18; i++)
    for (int j = 0; j < 18; j++)
      setText(j - 1, i - 1, String(j + i * 18));

  while (true)
  {
    // coloring visited cell yellow
    setColor(current % 18 - 1, current / 18 - 1, 'y');
```

```
    // marking discovered walls
    if (wallLeft())
    {
      int w = (4 + orient - 1) % 4;
      setWall(current % 18 - 1, current / 18 - 1, directions[w]);
    }
    if (wallRight())
    {
      int w = (4 + orient + 1) % 4;
      setWall(current % 18 - 1, current / 18 - 1, directions[w]);
    }
    if (wallFront())
    {
      int w = orient;
      setWall(current % 18 - 1, current / 18 - 1, directions[w]);
    }

    // marking cell as visited
    visited[current] = true;

    // storing discovered walls
    if (!wallFront())
      wall(0);
    if (!wallRight())
      wall(1);
    if (!wallLeft())
      wall(-1);

    if (!unvisited_i)
      break;

    // getting nearest unvisited cell
    int z = unvisited[unvisited_i - 1];
    unvisited_i--;
    while (true)
    {
      if (visited[z])
      {
      }
      else if (visited[z + 1] && visited[z - 1] && visited[z + 18] && visited[z - 18])
      {
        visited[z] = true;
        setColor(z % 18 - 1, z / 18 - 1, 'o');
        for (int i = 0; i < 4; i++)
          maze[z][i] = maze[z + dir[i]][(i + 2) % 4];
      }
      else
        break;
      if (unvisited_i)
```

```
    {
      z = unvisited[unvisited_i - 1];
      unvisited_i--;
    }
    else
    {
      z = -1;
      break;
    }
  }

  if (z == -1)
    break;

  int c = 0;
  int tmp = current;
  while (true)
  {
    bool b = false;
    for (int j = 0; j < 4; j++)
    {
      if (tmp + dir[j] == z && maze[tmp][j])
      {
        sprint(moves, moves_i - c, moves_i);
        moves_i -= c;

        if (!ex)
          go(j, 1);

        moves[moves_i] = current;
        moves_i++;

        current = z;
        int x = (orient + 2) % 4;
        maze[current][x] = true;
        b = true;

        if (ex == true)
          return;
        break;
      }
    }

    if (b)
      break;

    c++;
    tmp = moves[moves_i - c];
  }
```

```
  }

  //return to start
  bfs(19);

  int tmp = current;
  int i = 0;
  while (p[tmp] != -1)
  {
    i++;
    tmp = p[tmp];
  }
  moves_i = i;
  tmp = current;
  while (p[tmp] != -1)
  {
    moves[i - 1] = p[tmp];
    i--;
    tmp = p[tmp];
  }
  sprint(moves, 0, moves_i);
  go(0, 0);
}

void phase2()
{
  // calculate best path
  bfs(19);

  int v = 170;
  path_i = 0, path_j = 0;
  while (v != -1)
  {
    path[path_i] = v;
    path_i++;
    v = p[v];
  }
  while ((path[path_j] == 170 || path[path_j] == 171 || path[path_j] == 152 || path[path_j]
== 153) &&
         ((path[path_j + 1] == 170 || path[path_j + 1] == 171 || path[path_j + 1] == 152 ||
path[path_j + 1] == 153)))
  {
    path_j++;
  }
  for (int i = path_j; i < path_i; i++)
    setColor(path[i] % 18 - 1, path[i] / 18 - 1, 'g');

  path_i--;
  sprint(path, path_j, path_i);
```

```
  //return again to start
  for (moves_i = 0; moves_i < path_i; moves_i++)
    moves[moves_i] = path[path_i - moves_i];
  sprint(moves, 0, moves_i);
  go(0, 0);
}
```

- Simulator-API.ino

```
// ----- API -----

void log(String message) {
    Serial.print("log " + message + "\n");
}

int mazeWidth() {
    return getInteger("mazeWidth");
}

int mazeHeight() {
    return getInteger("mazeHeight");
}

bool wallFront() {
    return getBoolean("wallFront");
}

bool wallRight() {
    return getBoolean("wallRight");
}

bool wallLeft() {
    return getBoolean("wallLeft");
}

bool moveForward(int distance) {
    if(distance == 1)
        return getAck("moveForward");
     return getAck("moveForward " + String(distance));
}

void turnRight() {
    getAck("turnRight");
}

void turnLeft() {
    getAck("turnLeft");
```

```
}

void setWall(int x, int y, char direction) {
    Serial.print(
        "setWall "
        + String(x) + " "
        + String(y) + " "
        + String(direction) + "\n"
    );
}

void clearWall(int x, int y, char direction) {
    Serial.print(
        "clearWall "
        + String(x) + " "
        + String(y) + " "
        + String(direction) + "\n"
    );
}

void setColor(int x, int y, char color) {
    Serial.print(
        "setColor "
        + String(x) + " "
        + String(y) + " "
        + String(color) + "\n"
    );
}

void clearColor(int x, int y) {
    Serial.print(
        "clearColor "
        + String(x) + " "
        + String(y) + "\n"
    );
}

void clearAllColor() {
    Serial.print("clearAllColor\n");
}

void setText(int x, int y, String text) {
    Serial.print(
        "setText "
        + String(x) + " "
        + String(y) + " "
        + text + "\n"
    );
}
```

```
void clearText(int x, int y) {
    Serial.print(
        "clearText "
        + String(x) + " "
        + String(y) + "\n"
    );
}

void clearAllText() {
    Serial.print("clearAllText\n");
}

bool wasReset() {
    return getBoolean("wasReset");
}

void ackReset() {
    getAck("ackReset");
}

// ----- Helpers -----

String readline() {
    String response = "";
    while (response == "") {
        response = Serial.readStringUntil('\n');
    }
    return response;
}

String communicate(String command) {
    Serial.print(command + "\n");
    return readline();
}

bool getAck(String command) {
    String response = communicate(command);
    return response == "ack";
}

bool getBoolean(String command) {
    String response = communicate(command);
    return response == "true";
}

int getInteger(String command) {
    String response = communicate(command);
    return response.toInt();}
```

## 2-Motion & Sensors

- deshret-motion.ino

```
#include <Wire.h> //This is for i2C
#include <PID_v1.h>
#include "Adafruit_VL53L0X.h"

#define TURN 69.0
#define CELL 300.0

TwoWire Wire2 (2, I2C_FAST_MODE);

double Kp = 1, Ki = 0, Kd = 0;
double Setpoint1 = 0, Input1, Output1;
double Setpoint2 = 0, Input2, Output2;
PID PID1(&Input1, &Output1, &Setpoint1, Kp, Ki, Kd, DIRECT);
PID PID2(&Input2, &Output2, &Setpoint2, Kp, Ki, Kd, DIRECT);

int magnetStatus1 = 0; //value of the status register (MD, ML, MH)
int magnetStatus2 = 0;
int lowbyte1; //raw angle 7:0
float pi = 3.14285714;
word highbyte1; //raw angle 7:0 and 11:8
int rawAngle1; //final raw angle
float degAngle1; //raw angle in degrees (360/4096 * [value between 0-4095])

int lowbyte2; //raw angle 7:0
word highbyte2; //raw angle 7:0 and 11:8
int rawAngle2; //final raw angle
float degAngle2; //raw angle in degrees (360/4096 * [value between 0-4095])

int quadrantNumber1, previousquadrantNumber1; //quadrant IDs
float numberofTurns1 = 0; //number of turns
float correctedAngle1 = 0; //tared angle - based on the startup value
float startAngle1 = 0; //starting angle
float totalAngle1 = 0; //total absolute angular displacement
float previoustotalAngle1 = 0; //for the display printing

int quadrantNumber2, previousquadrantNumber2; //quadrant IDs
float numberofTurns2 = 0; //number of turns
float correctedAngle2 = 0; //tared angle - based on the startup value
float startAngle2 = 0; //starting angle
float totalAngle2 = 0; //total absolute angular displacement
float previoustotalAngle2 = 0; //for the display printing

float offsetL = 0, offsetR = 0;

// objects for the vl53l0x
Adafruit_VL53L0X lox1 = Adafruit_VL53L0X();
```

```
Adafruit_VL53L0X lox2 = Adafruit_VL53L0X();
Adafruit_VL53L0X lox3 = Adafruit_VL53L0X();

// this holds the measurement
VL53L0X_RangingMeasurementData_t measure1;
VL53L0X_RangingMeasurementData_t measure2;
VL53L0X_RangingMeasurementData_t measure3;

void setup()
{
  Serial.begin(115200); //start serial - tip: don't use serial if you don't need it (speed
considerations)
  Wire.begin(); //start i2C
  Wire.setClock(800000L); //fast clock
  pinMode(PC15, INPUT_PULLUP);
  Wire2.begin(); //start i2C
  Wire2.setClock(800000L); //fast clock

  checkMagnetPresence1(); //check the magnet (blocks until magnet is found)
  checkMagnetPresence2();
  ReadRawAngle1(); //make a reading so the degAngle gets updated
  ReadRawAngle2();
  startAngle1 = degAngle1; //update startAngle with degAngle - for taring
  startAngle2 = degAngle2;

  PID1.SetMode(AUTOMATIC);
  PID2.SetMode(AUTOMATIC);
  PID1.SetOutputLimits(-255, 255);
  PID2.SetOutputLimits(-255, 255);

  while (digitalRead(PC15))
  {
    updateEncoder();
  }
  int timer = millis();
  while (millis() < timer + 1000)
  {
    updateEncoder();
  }
  offsetL = calculate_distanceL(totalAngle1);
  offsetR = calculate_distanceR(totalAngle2);
  //turnRight();
}

void loop()
{

}
```

```
bool wallFront()
{
  lox1.rangingTest(&measure1, false); // pass in 'true' to get debug data printout!
  if (measure1.RangeStatus != 4)
    return measure1.RangeMilliMeter < CELL;
  return false;
}

bool wallRight()
{
  lox2.rangingTest(&measure2, false); // pass in 'true' to get debug data printout!
  if (measure2.RangeStatus != 4)
    return measure2.RangeMilliMeter < CELL;
  return false;
}

bool wallLeft()
{
  lox3.rangingTest(&measure3, false); // pass in 'true' to get debug data printout!
  if (measure3.RangeStatus != 4)
    return measure3.RangeMilliMeter < CELL;
  return false;
}

bool moveForward(int distance)
{
  Setpoint1 += CELL * distance;
  Setpoint2 += CELL * distance;

  do
  {
    updateEncoder();

    float distL = calculate_distanceL(totalAngle1) - offsetL;
    float distR = calculate_distanceR(totalAngle2) - offsetR;

    Input1 = distL;
    PID1.Compute();
    Input2 = distR;
    PID2.Compute();

    Serial.print("Distance 1: ");
    Serial.print(distL);
    Serial.print(", Distance 2: ");
    Serial.print(distR);
    Serial.print(", Offset 1: ");
    Serial.print(offsetL);
    Serial.print(", Offset 2: ");
    Serial.print(offsetR);
```

```
    Serial.print(", Output: ");
    Serial.print(Output1);
    Serial.print(" / ");
    Serial.println(Output2);

    analogWrite(PA0, Output1); //Left wheels forward
    analogWrite(PA1, 0); //Left wheels backward
    analogWrite(PA2, 0); //Right wheels backward
    analogWrite(PA3, Output2); //Right wheels forward
  }
  while (Output1 > 0 && Output2 > 0);
}

void turnRight()
{
  Setpoint1 += TURN;
  Setpoint2 -= TURN;

  do
  {
    updateEncoder();

    float distL = calculate_distanceL(totalAngle1) - offsetL;
    float distR = calculate_distanceR(totalAngle2) - offsetR;

    Input1 = distL;
    PID1.Compute();
    Input2 = distR;
    PID2.Compute();

    Serial.print("Distance 1: ");
    Serial.print(distL);
    Serial.print(", Distance 2: ");
    Serial.print(distR);
    Serial.print(", Offset 1: ");
    Serial.print(offsetL);
    Serial.print(", Offset 2: ");
    Serial.print(offsetR);
    Serial.print(", Output: ");
    Serial.print(Output1);
    Serial.print(" / ");
    Serial.println(Output2);

    analogWrite(PA0, Output1); //Left wheels forward
    analogWrite(PA1, 0); //Left wheels backward
    analogWrite(PA2, -Output2); //Right wheels backward
    analogWrite(PA3, 0); //Right wheels forward
  }
  while (Output1 > 0 && -Output2 > 0);
```

```
}

void turnLeft()
{
  Setpoint1 -= TURN;
  Setpoint2 += TURN;

  do
  {
    updateEncoder();

    float distL = calculate_distanceL(totalAngle1) - offsetL;
    float distR = calculate_distanceR(totalAngle2) - offsetR;

    Input1 = distL;
    PID1.Compute();
    Input2 = distR;
    PID2.Compute();

    Serial.print("Distance 1: ");
    Serial.print(distL);
    Serial.print(", Distance 2: ");
    Serial.print(distR);
    Serial.print(", Offset 1: ");
    Serial.print(offsetL);
    Serial.print(", Offset 2: ");
    Serial.print(offsetR);
    Serial.print(", Output: ");
    Serial.print(Output1);
    Serial.print(" / ");
    Serial.println(Output2);

    analogWrite(PA0, 0); //Left wheels forward
    analogWrite(PA1, -Output1); //Left wheels backward
    analogWrite(PA2, 0); //Right wheels backward
    analogWrite(PA3, Output2); //Right wheels forward
  }
  while (-Output1 > 0 && Output2 > 0);
}
```

- encoder.ino

```
void ReadRawAngle1()
{
  //7:0 - bits
  Wire.beginTransmission(0x36); //connect to the sensor
  Wire.write(0x0D); //figure 21 - register map: Raw angle (7:0)
  Wire.endTransmission(); //end transmission
```

```
  Wire.requestFrom(0x36, 1); //request from the sensor

  while (Wire.available() == 0); //wait until it becomes available
  lowbyte1 = Wire.read(); //Reading the data after the request

  //11:8 - 4 bits
  Wire.beginTransmission(0x36);
  Wire.write(0x0C); //figure 21 - register map: Raw angle (11:8)
  Wire.endTransmission();
  Wire.requestFrom(0x36, 1);

  while (Wire.available() == 0);
  highbyte1 = Wire.read();

  //4 bits have to be shifted to its proper place as we want to build a 12-bit number
  highbyte1 = highbyte1 << 8; //shifting to left

  rawAngle1 = highbyte1 | lowbyte1; //int is 16 bits (as well as the word)

  degAngle1 = rawAngle1 * 0.087890625;

  //Serial.print("Deg angle1: ");
  //Serial.println(degAngle1, 2); //absolute position of the encoder within the 0-360 circle

}
void ReadRawAngle2()
{
  //7:0 - bits
  Wire2.beginTransmission(0x36); //connect to the sensor
  Wire2.write(0x0D); //figure 21 - register map: Raw angle (7:0)
  Wire2.endTransmission(); //end transmission
  Wire2.requestFrom(0x36, 1); //request from the sensor

  while (Wire2.available() == 0); //wait until it becomes available
  lowbyte2 = Wire2.read(); //Reading the data after the request

  //11:8 - 4 bits
  Wire2.beginTransmission(0x36);
  Wire2.write(0x0C); //figure 21 - register map: Raw angle (11:8)
  Wire2.endTransmission();
  Wire2.requestFrom(0x36, 1);

  while (Wire2.available() == 0);
  highbyte2 = Wire2.read();

  //4 bits have to be shifted to its proper place as we want to build a 12-bit number
  highbyte2 = highbyte2 << 8; //shifting to left

  rawAngle2 = highbyte2 | lowbyte2; //int is 16 bits (as well as the word)
```

```
  degAngle2 = rawAngle2 * 0.087890625;

  //Serial.print("Deg angle: ");
  //Serial.println(degAngle2, 2); //absolute position of the encoder within the 0-360 circle

}
void correctAngle1()
{
  //recalculate angle
  correctedAngle1 = degAngle1 - startAngle1; //this tares the position

  if (correctedAngle1 < 0) //if the calculated angle is negative, we need to "normalize" it
  {
    correctedAngle1 = correctedAngle1 + 360; //correction for negative numbers (i.e. -15
becomes +345)
  }
  else
  {
    //do nothing
  }
  //Serial.print("Corrected angle: ");
  //Serial.println(correctedAngle1, 2); //print the corrected/tared angle
}
void correctAngle2()
{
  //recalculate angle
  correctedAngle2 = degAngle2 - startAngle2; //this tares the position

  if (correctedAngle2 < 0) //if the calculated angle is negative, we need to "normalize" it
  {
    correctedAngle2 = correctedAngle2 + 360; //correction for negative numbers (i.e. -15
becomes +345)
  }
  else
  {
    //do nothing
  }
  //Serial.print("Corrected angle: ");
  //Serial.println(correctedAngle2, 2); //print the corrected/tared angle
}
void read_total()
{
  /*
    //Quadrants:
    4  |  1
    ---|---
    3  |  2
  */
```

```
//Quadrant 1
if (correctedAngle1 >= 0 && correctedAngle1 <= 90)
{
  quadrantNumber1 = 1;
}

//Quadrant 2
if (correctedAngle1 > 90 && correctedAngle1 <= 180)
{
  quadrantNumber1 = 2;
}

//Quadrant 3
if (correctedAngle1 > 180 && correctedAngle1 <= 270)
{
  quadrantNumber1 = 3;
}

//Quadrant 4
if (correctedAngle1 > 270 && correctedAngle1 < 360)
{
  quadrantNumber1 = 4;
}
//Serial.print("Quadrant: ");
//Serial.println(quadrantNumber); //print our position "quadrant-wise"

if (quadrantNumber1 != previousquadrantNumber1) //if we changed quadrant
{
  if (quadrantNumber1 == 1 && previousquadrantNumber1 == 4)
  {
    numberofTurns1++; // 4 --> 1 transition: CW rotation
  }

  if (quadrantNumber1 == 4 && previousquadrantNumber1 == 1)
  {
    numberofTurns1--; // 1 --> 4 transition: CCW rotation
  }
  //this could be done between every quadrants so one can count every 1/4th of transition

  previousquadrantNumber1 = quadrantNumber1;  //update to the current quadrant

}
if (correctedAngle2 >= 0 && correctedAngle2 <= 90)
{
  quadrantNumber2 = 1;
}

//Quadrant 2
```

```
  if (correctedAngle2 > 90 && correctedAngle2 <= 180)
  {
    quadrantNumber2 = 2;
  }

  //Quadrant 3
  if (correctedAngle2 > 180 && correctedAngle2 <= 270)
  {
    quadrantNumber2 = 3;
  }

  //Quadrant 4
  if (correctedAngle2 > 270 && correctedAngle2 < 360)
  {
    quadrantNumber2 = 4;
  }
  //Serial.print("Quadrant: ");
  //Serial.println(quadrantNumber); //print our position "quadrant-wise"

  if (quadrantNumber2 != previousquadrantNumber2) //if we changed quadrant
  {
    if (quadrantNumber2 == 1 && previousquadrantNumber2 == 4)
    {
      numberofTurns2++; // 4 --> 1 transition: CW rotation
    }

    if (quadrantNumber2 == 4 && previousquadrantNumber2 == 1)
    {
      numberofTurns2--; // 1 --> 4 transition: CCW rotation
    }
    //this could be done between every quadrants so one can count every 1/4th of transition

    previousquadrantNumber2 = quadrantNumber2;  //update to the current quadrant

  }
  totalAngle1 = (numberofTurns1 * 360) + correctedAngle1; //number of turns (+/-) plus the
actual angle within the 0-360 range
  //Serial.print("Total angle1: ");
  //Serial.println(totalAngle1, 2);

  totalAngle2 = (numberofTurns2 * 360) + correctedAngle2; //number of turns (+/-) plus the
actual angle within the 0-360 range
  // Serial.print("Total angle2: ");
  // Serial.println(totalAngle2, 2);
  //error =(totalAngle1*(-1))-totalAngle2;
  // Serial.println("Error inside main fun");
  //Serial.println(error);
}
```

```
void checkMagnetPresence1()
{
  //This function runs in the setup() and it locks the MCU until the magnet is not positioned properly

  while ((magnetStatus1 & 32) != 32) //while the magnet is not adjusted to the proper distance - 32: MD = 1
  {
    magnetStatus1 = 0; //reset reading

    Wire.beginTransmission(0x36); //connect to the sensor
    Wire.write(0x0B); //figure 21 - register map: Status: MD ML MH
    Wire.endTransmission(); //end transmission
    Wire.requestFrom(0x36, 1); //request from the sensor

    while (Wire.available() == 0); //wait until it becomes available
    magnetStatus1 = Wire.read(); //Reading the data after the request

    //Serial.print("Magnet status: ");
    //Serial.println(magnetStatus, BIN); //print it in binary so you can compare it to the table (fig 21)
  }
}
void checkMagnetPresence2()
{
  //This function runs in the setup() and it locks the MCU until the magnet is not positioned properly

  while ((magnetStatus2 & 32) != 32) //while the magnet is not adjusted to the proper distance - 32: MD = 1
  {
    magnetStatus2 = 0; //reset reading

    Wire2.beginTransmission(0x36); //connect to the sensor
    Wire2.write(0x0B); //figure 21 - register map: Status: MD ML MH
    Wire2.endTransmission(); //end transmission
    Wire2.requestFrom(0x36, 1); //request from the sensor

    while (Wire2.available() == 0); //wait until it becomes available
    magnetStatus2 = Wire2.read(); //Reading the data after the request

    //Serial.print("Magnet status: ");
    //Serial.println(magnetStatus, BIN); //print it in binary so you can compare it to the table (fig 21)
  }

  //Status register output: 0 0 MD ML MH 0 0 0
  //MH: Too strong magnet - 100111 - DEC: 39
  //ML: Too weak magnet - 10111 - DEC: 23
```

```
  //MD: OK magnet - 110111 - DEC: 55

  //Serial.println("Magnet found!");
  //delay(1000);
}
float calculate_distanceR(float totalAngle2)
{
  float distanceR = (totalAngle2 * 28 * pi) / 360;
  return distanceR;
}
float calculate_distanceL(float totalAngle1)
{
  float distanceL = (totalAngle1 * 28 * pi) / 360;
  return ((distanceL) * -1);
}
void updateEncoder()
{
  ReadRawAngle1();
  ReadRawAngle2();
  correctAngle1();
  correctAngle2();
  read_total();
}
```

- vlx.ino

```
// address we will assign if dual sensor is present
#define LOX1_ADDRESS 0x30
#define LOX2_ADDRESS 0x31
#define LOX3_ADDRESS 0x32

// set the pins to shutdown
/*#define SHT_LOX1 PA11
  #define SHT_LOX2 PA12
  #define SHT_LOX3 PA15
*/
#define SHT_LOX1 8
#define SHT_LOX2 7
#define SHT_LOX3 4

/*
    Reset all sensors by setting all of their XSHUT pins low for delay(10), then set all
XSHUT high to bring out of reset
    Keep sensor #1 awake by keeping XSHUT pin high
    Put all other sensors into shutdown by pulling XSHUT pins low
    Initialize sensor #1 with lox.begin(new_i2c_address) Pick any number but 0x29 and it must
be under 0x7F. Going with 0x30 to 0x3F is probably OK.
    Keep sensor #1 awake, and now bring sensor #2 out of reset by setting its XSHUT pin high.
```

```
    Initialize sensor #2 with lox.begin(new_i2c_address) Pick any number but 0x29 and
whatever you set the first sensor to
*/
void setID() {
  // all reset
  digitalWrite(SHT_LOX1, LOW);
  digitalWrite(SHT_LOX2, LOW);
  digitalWrite(SHT_LOX3, LOW);
  delay(10);
  // all unreset
  digitalWrite(SHT_LOX1, HIGH);
  digitalWrite(SHT_LOX2, HIGH);
  digitalWrite(SHT_LOX3, HIGH);
  delay(10);

  // activating LOX1 and reseting LOX2
  digitalWrite(SHT_LOX1, HIGH);
  digitalWrite(SHT_LOX2, LOW);
  digitalWrite(SHT_LOX3, LOW);

  // initing LOX1
  if (!lox1.begin(LOX1_ADDRESS)) {
    Serial.println(F("Failed to boot first VL53L0X"));
    while (1);
  }
  delay(10);

  // activating LOX2
  digitalWrite(SHT_LOX2, HIGH);
  delay(10);

  //initing LOX2
  if (!lox2.begin(LOX2_ADDRESS)) {
    Serial.println(F("Failed to boot second VL53L0X"));
    while (1);
  }

  // activating LOX3
  digitalWrite(SHT_LOX3, HIGH);
  delay(10);

  //initing LOX3
  if (!lox3.begin(LOX3_ADDRESS)) {
    Serial.println(F("Failed to boot third VL53L0X"));
    while (1);
  }
}

void read_triple_sensors() {
```

```
  lox1.rangingTest(&measure1, false); // pass in 'true' to get debug data printout!
  lox2.rangingTest(&measure2, false); // pass in 'true' to get debug data printout!
  lox3.rangingTest(&measure3, false); // pass in 'true' to get debug data printout!

  // print sensor one reading
  Serial.print(F("1: "));
  if (measure1.RangeStatus != 4) {     // if not out of range
    Serial.print(measure1.RangeMilliMeter);
  } else {
    Serial.print(F("Out of range"));
  }

  Serial.print(F(" "));

  // print sensor two reading
  Serial.print(F("2: "));
  if (measure2.RangeStatus != 4) {     // if not out of range
    Serial.print(measure2.RangeMilliMeter);
  } else {
    Serial.print(F("Out of range"));
  }

  Serial.print(F(" "));

  // print sensor three reading
  Serial.print(F("3: "));
  if (measure3.RangeStatus != 4) {     // if not out of range
    Serial.print(measure3.RangeMilliMeter);
  } else {
    Serial.print(F("Out of range"));
  }

  Serial.println();
}

void vlx_setup() {
  pinMode(SHT_LOX1, OUTPUT);
  pinMode(SHT_LOX2, OUTPUT);
  pinMode(SHT_LOX3, OUTPUT);
  digitalWrite(SHT_LOX1, LOW);
  digitalWrite(SHT_LOX2, LOW);
  digitalWrite(SHT_LOX3, LOW);
  setID();
}
```

***Note:***

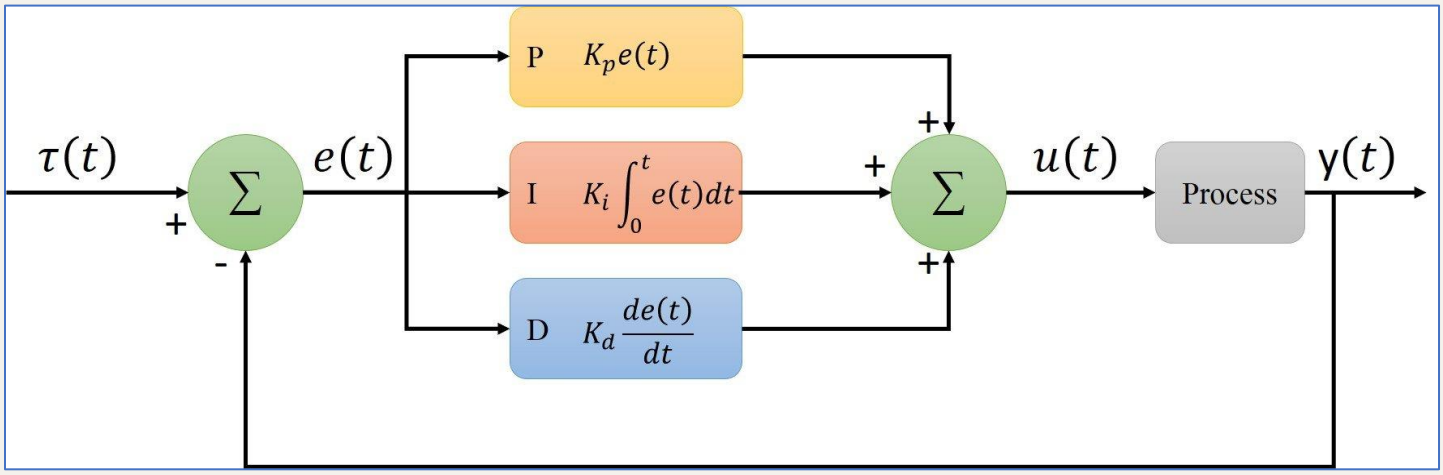
***At the moment of finalizing this report, codes of Algorithm & OLED and Motion & Sensors were not yet merged.***

# PID:

*PID control is a well-established way of driving a system towards a target position or level.*

- A proportional–integral–derivative controller (PID controller or three-term controller) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control to keep the actual output from a process as close to the target or setpoint output as possible..
- A PID controller continuously calculates an error value {e(t)} as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively), hence the name.
- The controller's PID algorithm restores the measured speed to the desired speed with minimal delay and overshoot by increasing the power output of the engine in a controlled manner.

*Therefore, the PID controlling method is used to sustain stable robot motion and minimizing the overshoot. It guarantees straight forward motion with neglectable drifting error and turning almost an exact 90 and 180 degrees by continually correcting the error occurred during the motion*




*PID-Diagram*