ME299

PROJECT

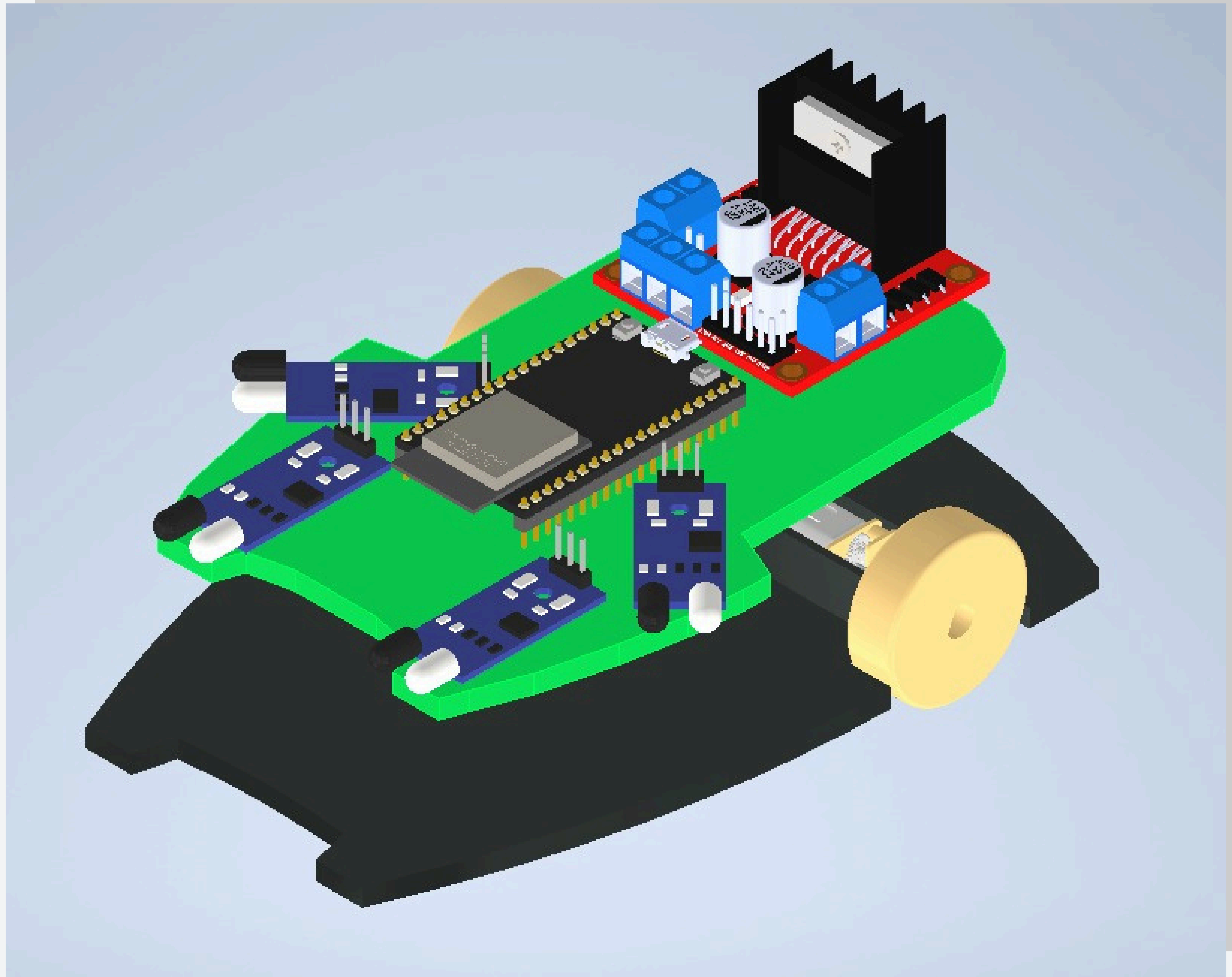
MICKEY-MICRO-MOUSE

SHAURYKUMAR PATEL

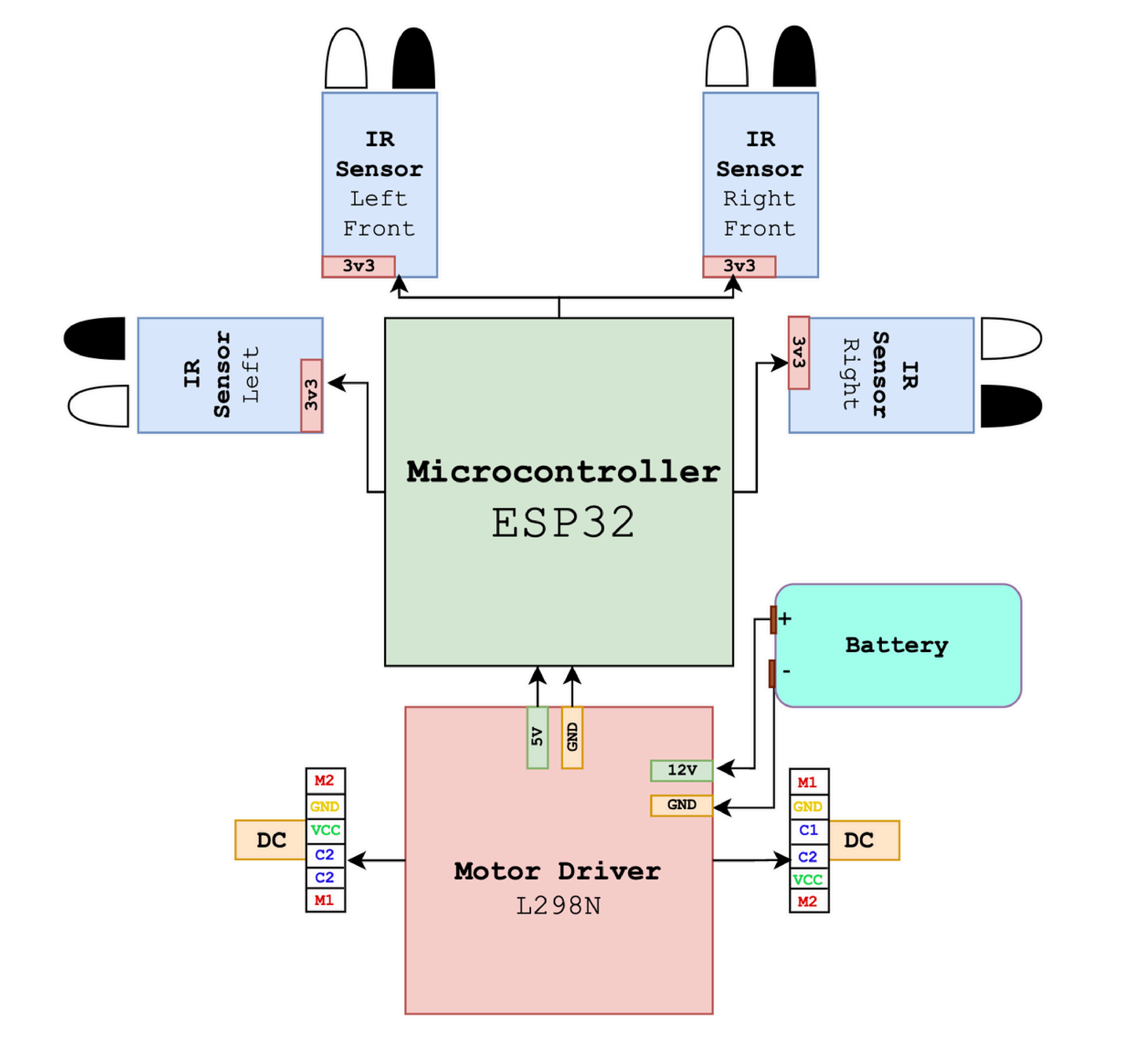
HITESH KUMAR

INTRODUCTION

The report outlines our project involving the development of a **MicroMouse, a wheeled mobile robot equipped with sensors to navigate through an unknown maze and reach a predetermined destination.** It covers key aspects including **hardware components, design, algorithms.** The document provides a comprehensive overview of the project's evolution from inception to its current state, detailing significant progress achieved over the weeks.

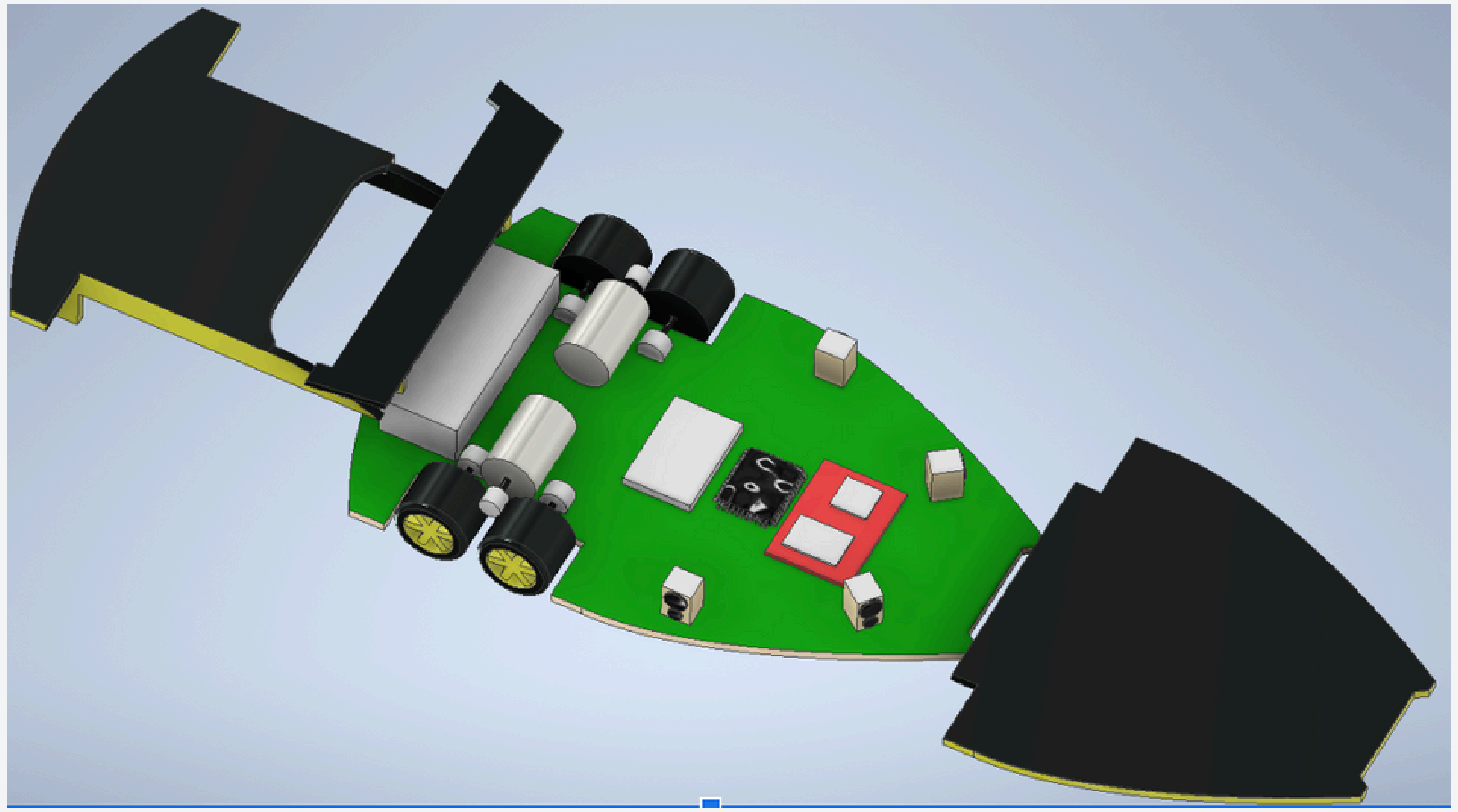


BLOCK DIAGRAM OF HARDWARE

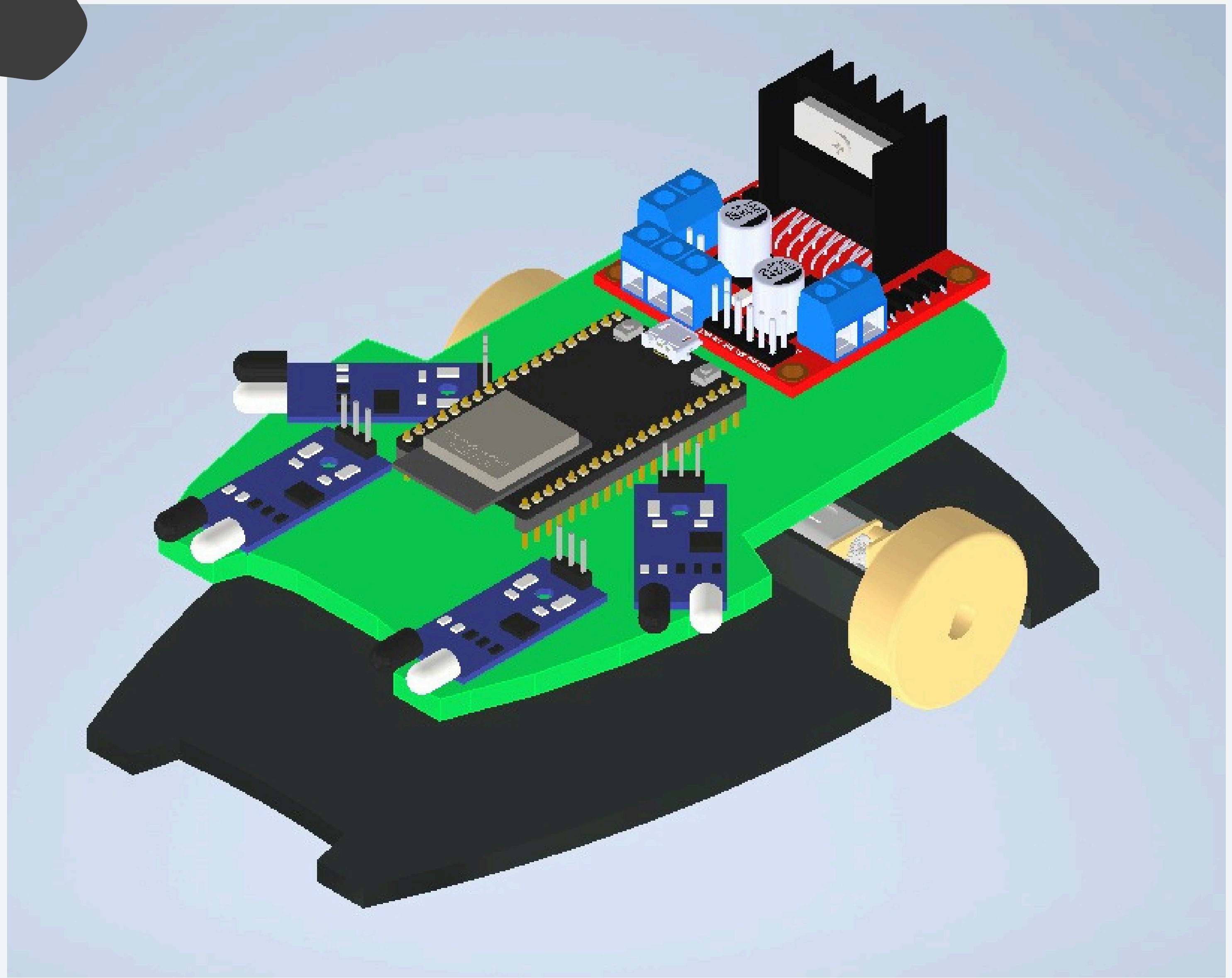


DESIGN

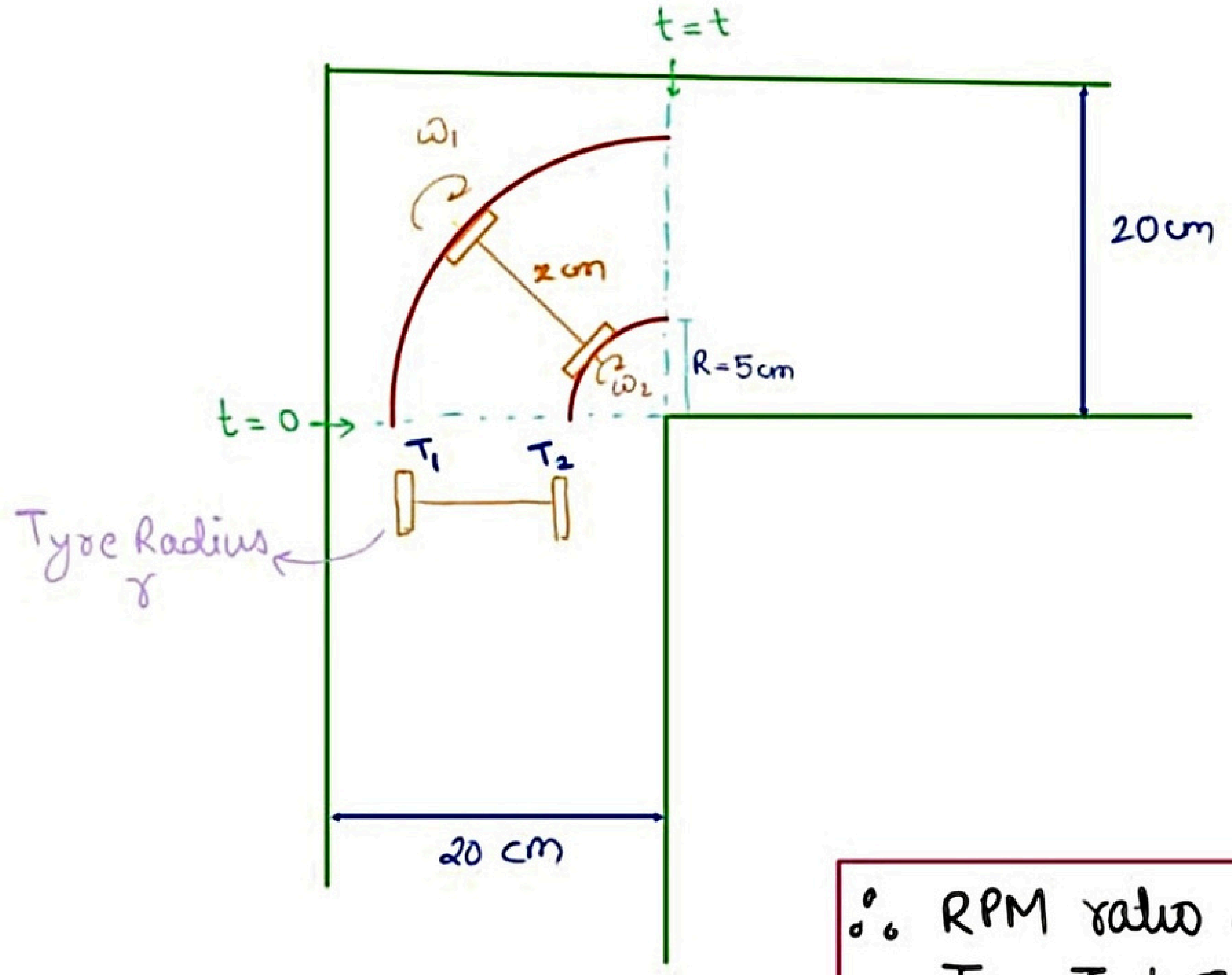
Proposed Design



Updated Design



SOME MAJOR CALCULATIONS



$T_1 \rightarrow 2\pi(x+R)/4$

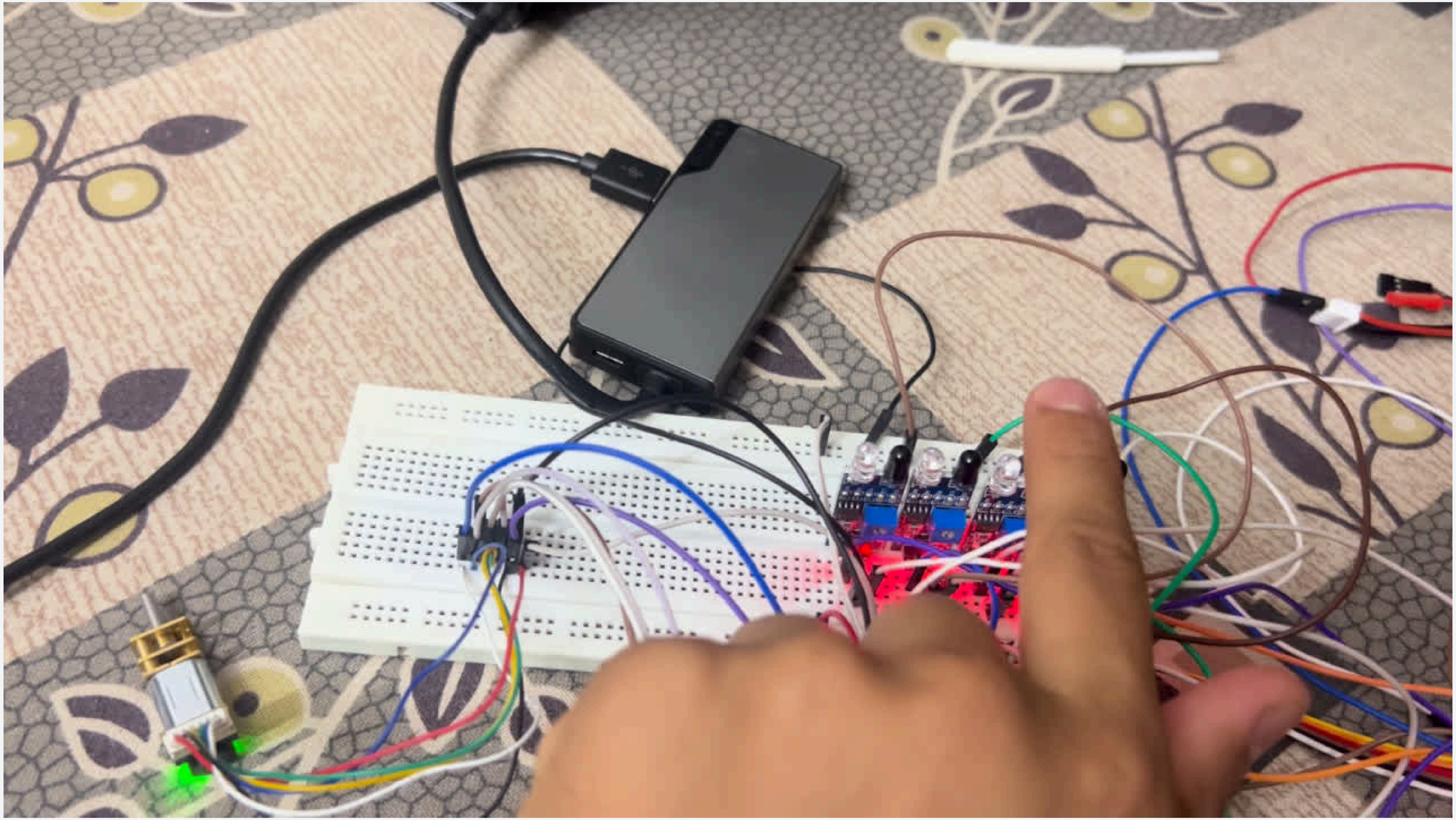
$$\text{No. of revolutions of } T_1 = \frac{2\pi(x+R)/4}{2\pi r} = \frac{x+R}{4r}$$

$T \rightarrow 2\pi R/4$

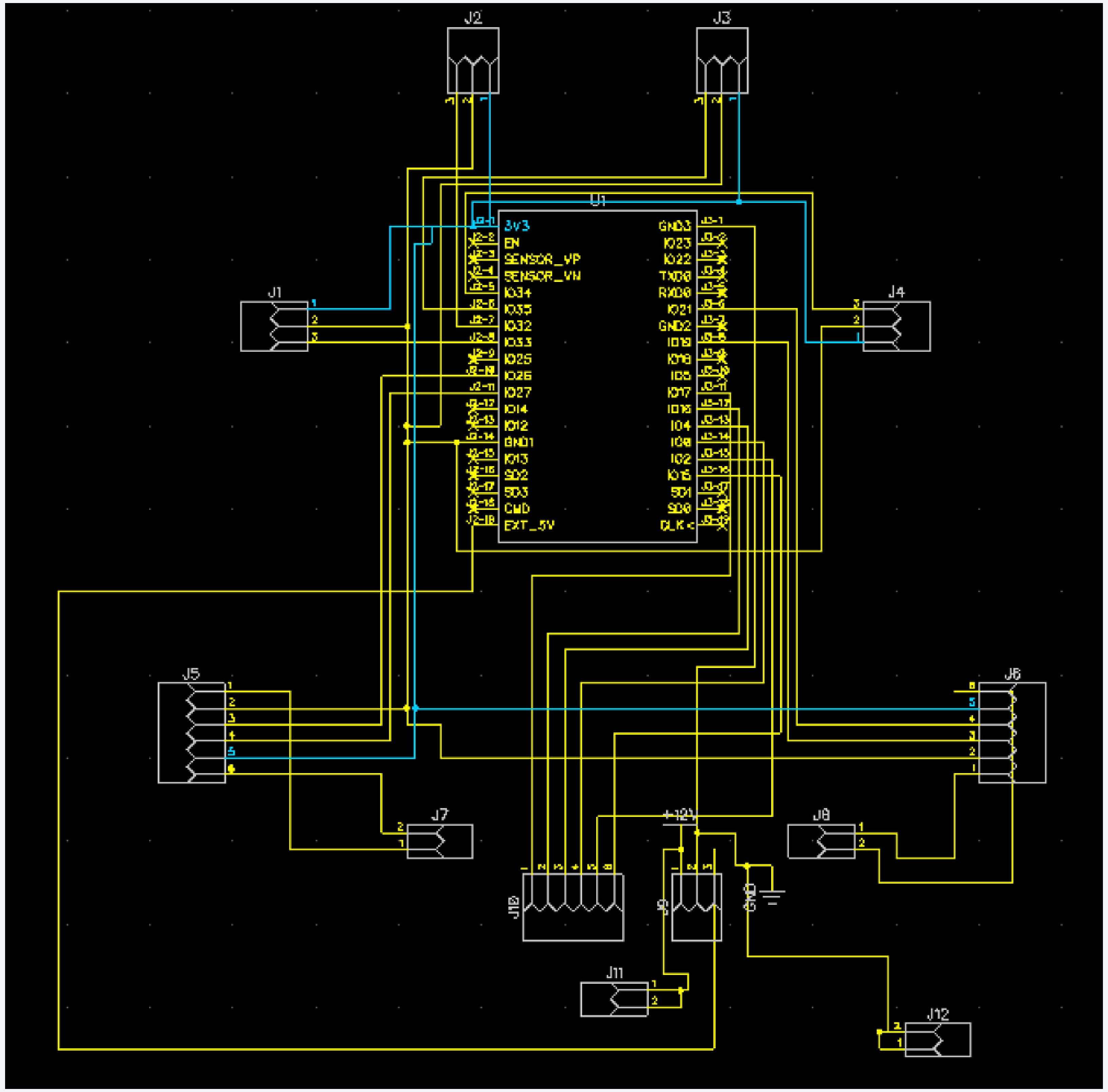
$$\text{No. of revolutions of } T_1 = \frac{2\pi R/4}{2\pi r} = \frac{R}{4r}$$

$$\boxed{\therefore \text{RPM ratio of Tyre } T_1 \text{ to } T_2 = \frac{x+R}{R}}$$

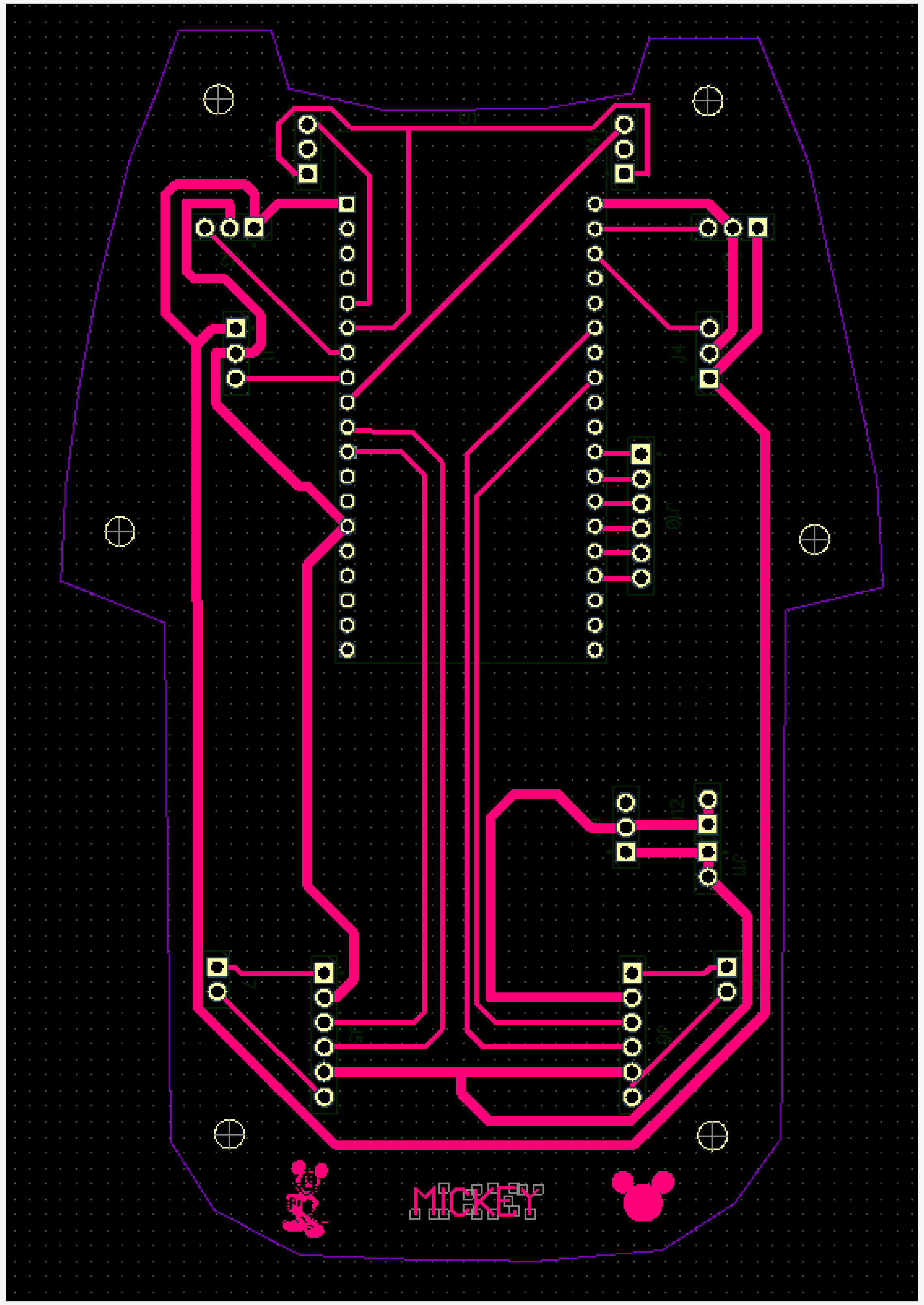
PROTOTYPE



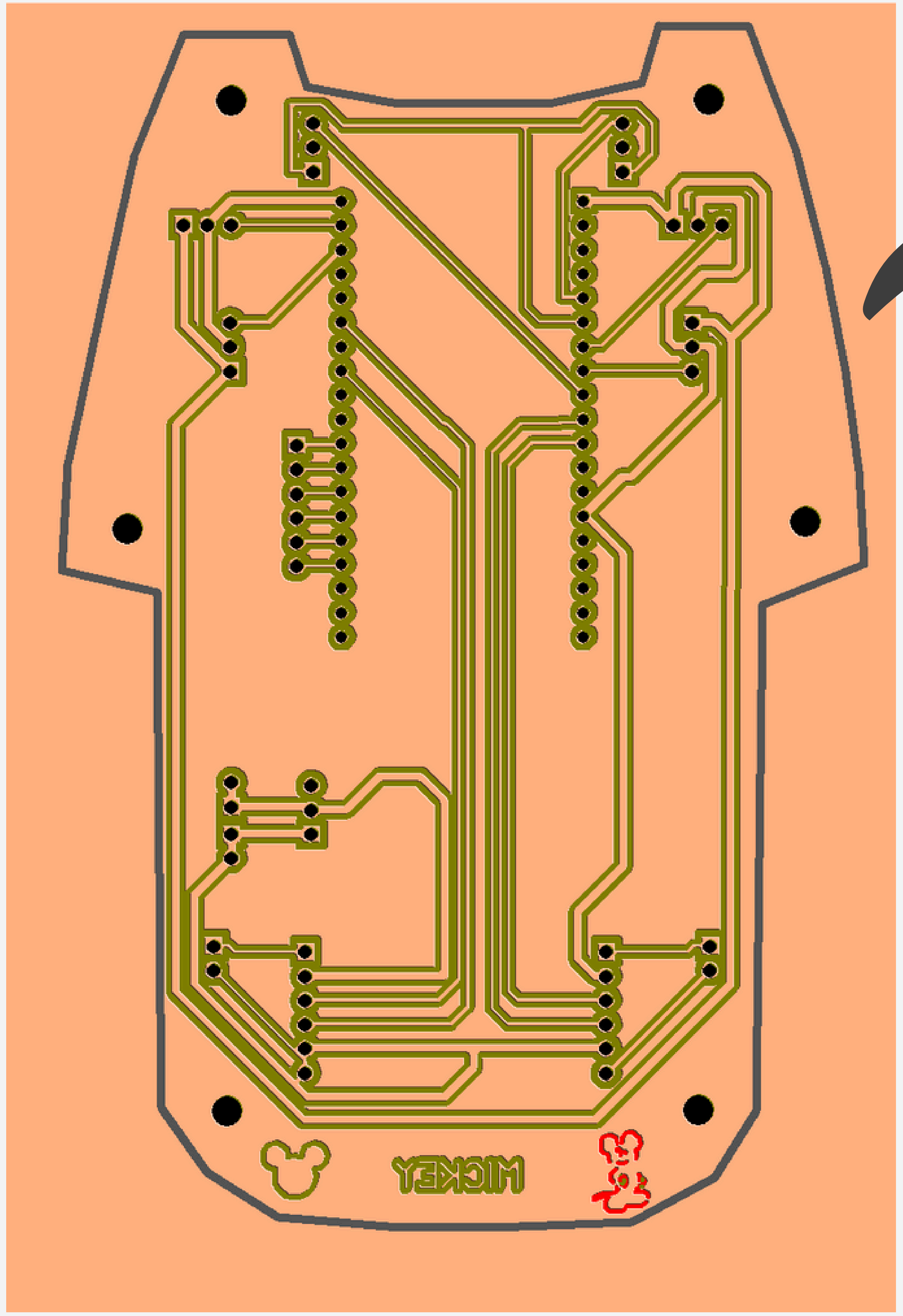
PCB Schematic



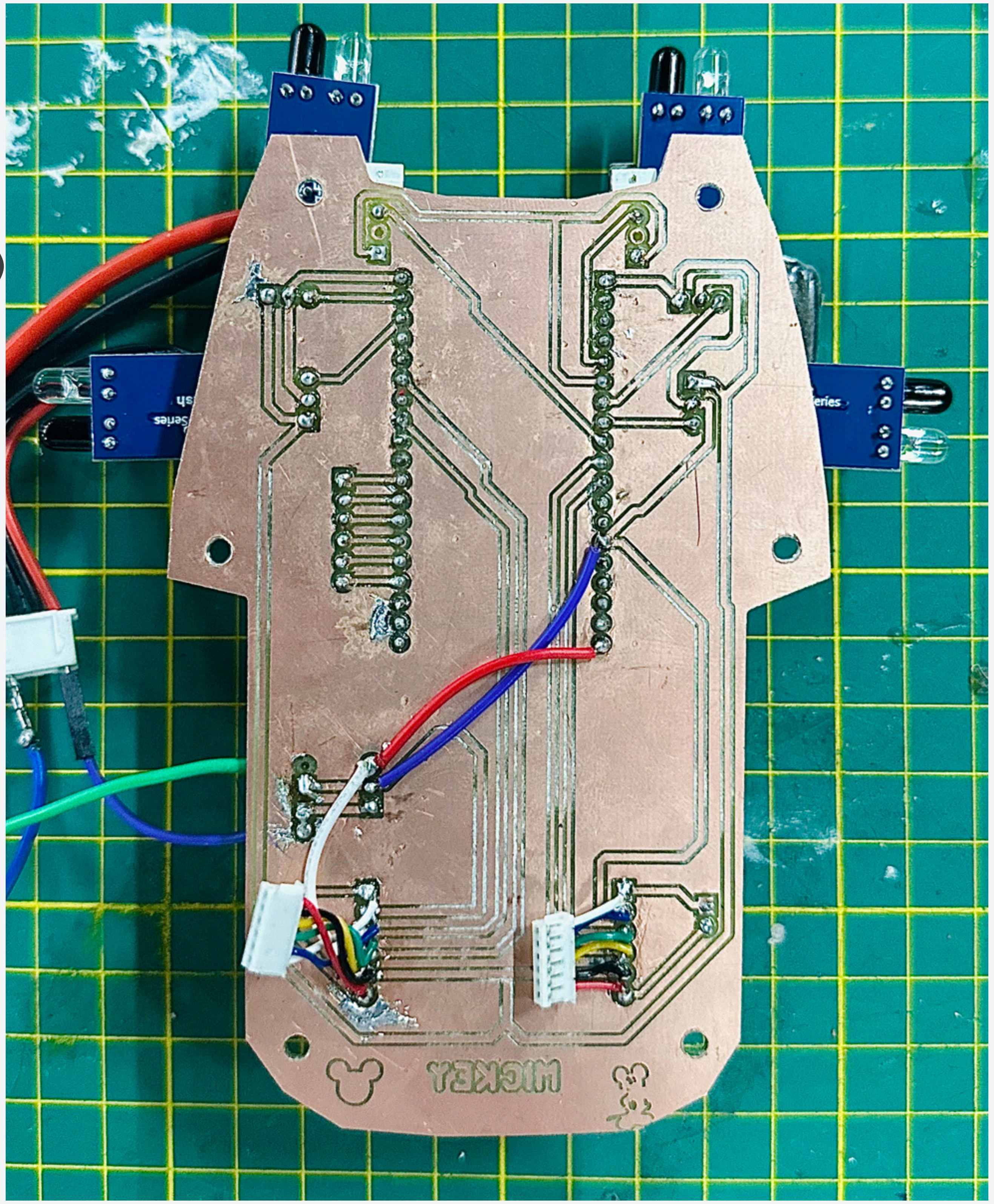
PCB Layout

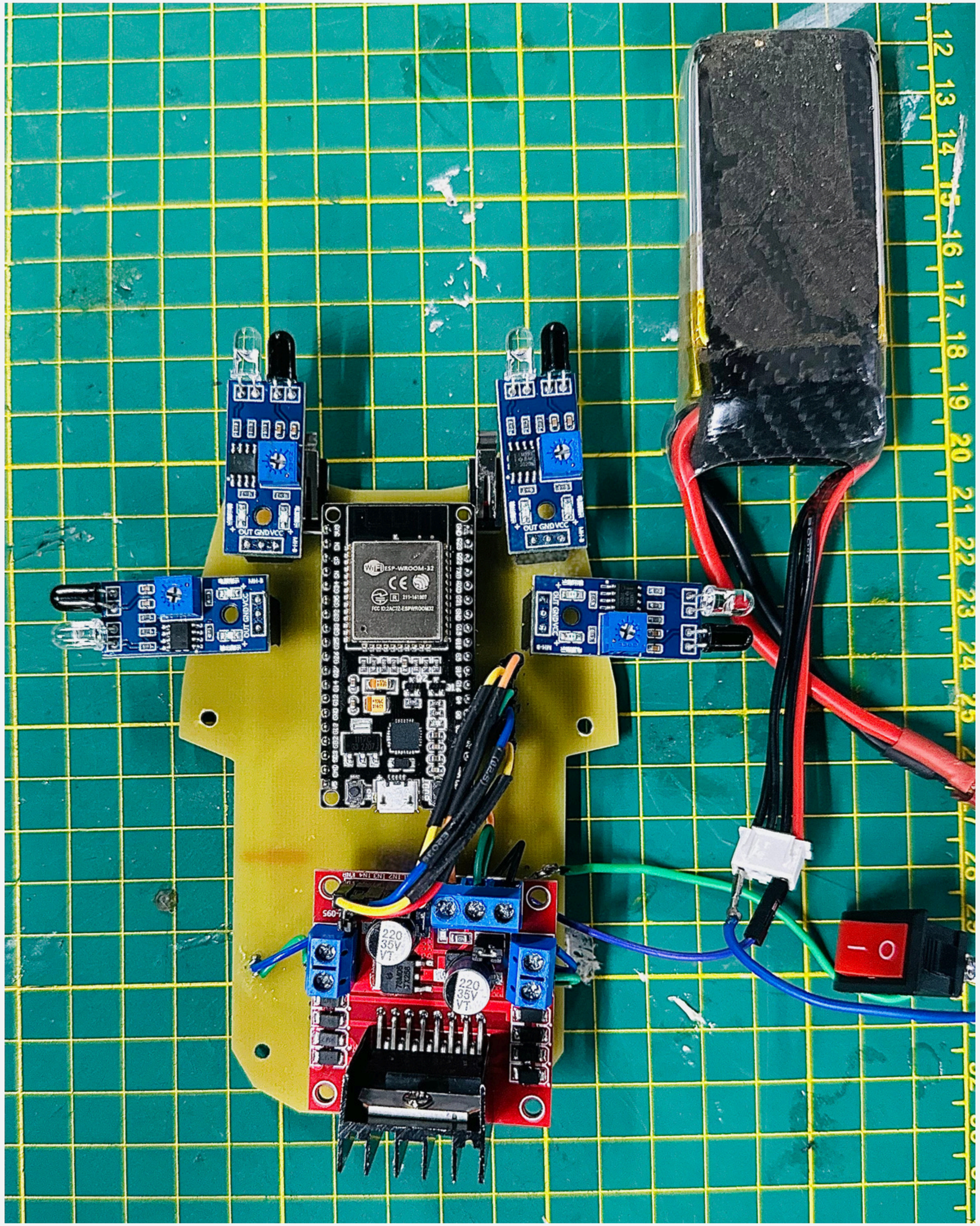


Copper Cam



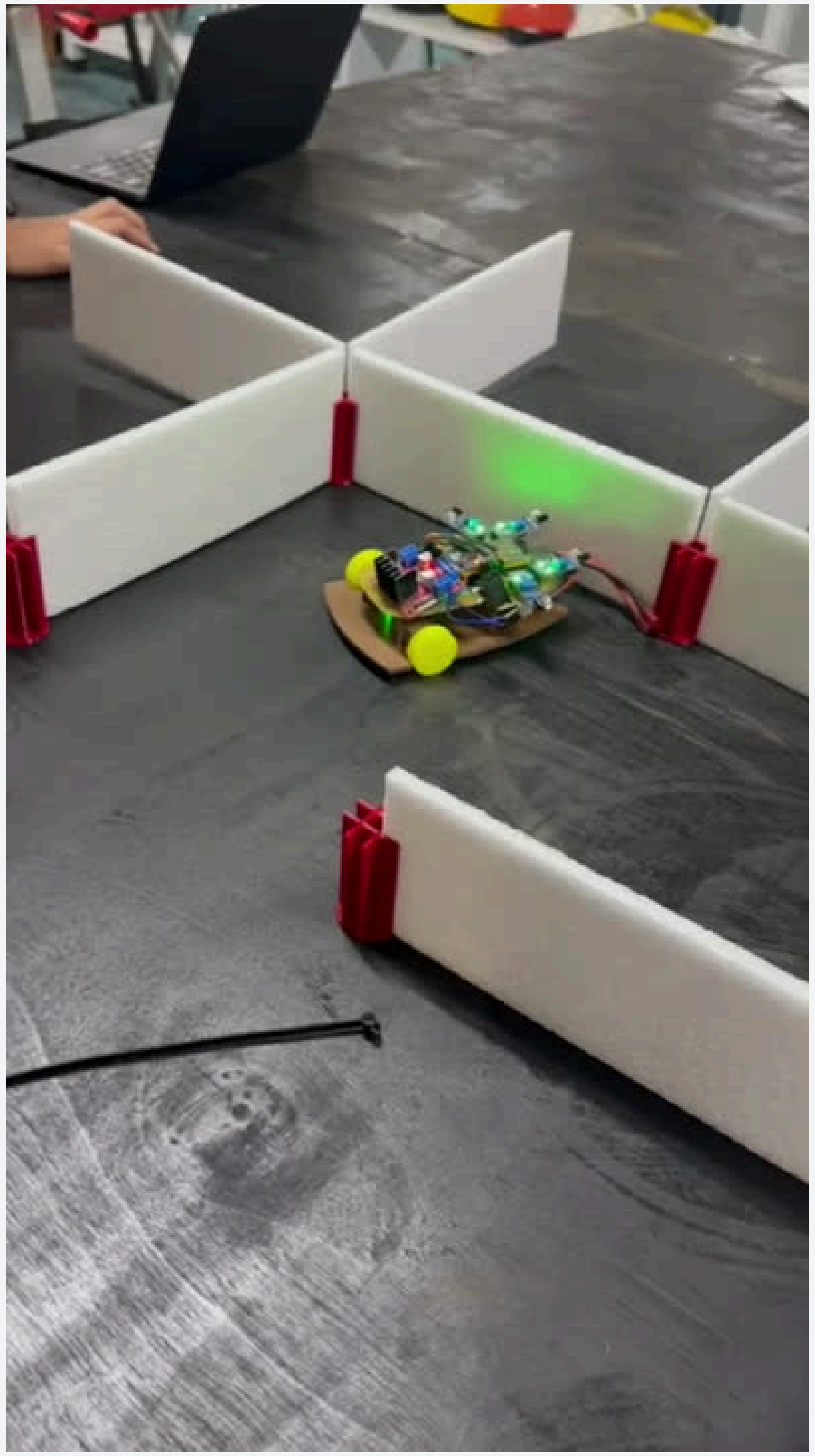
Final PCB

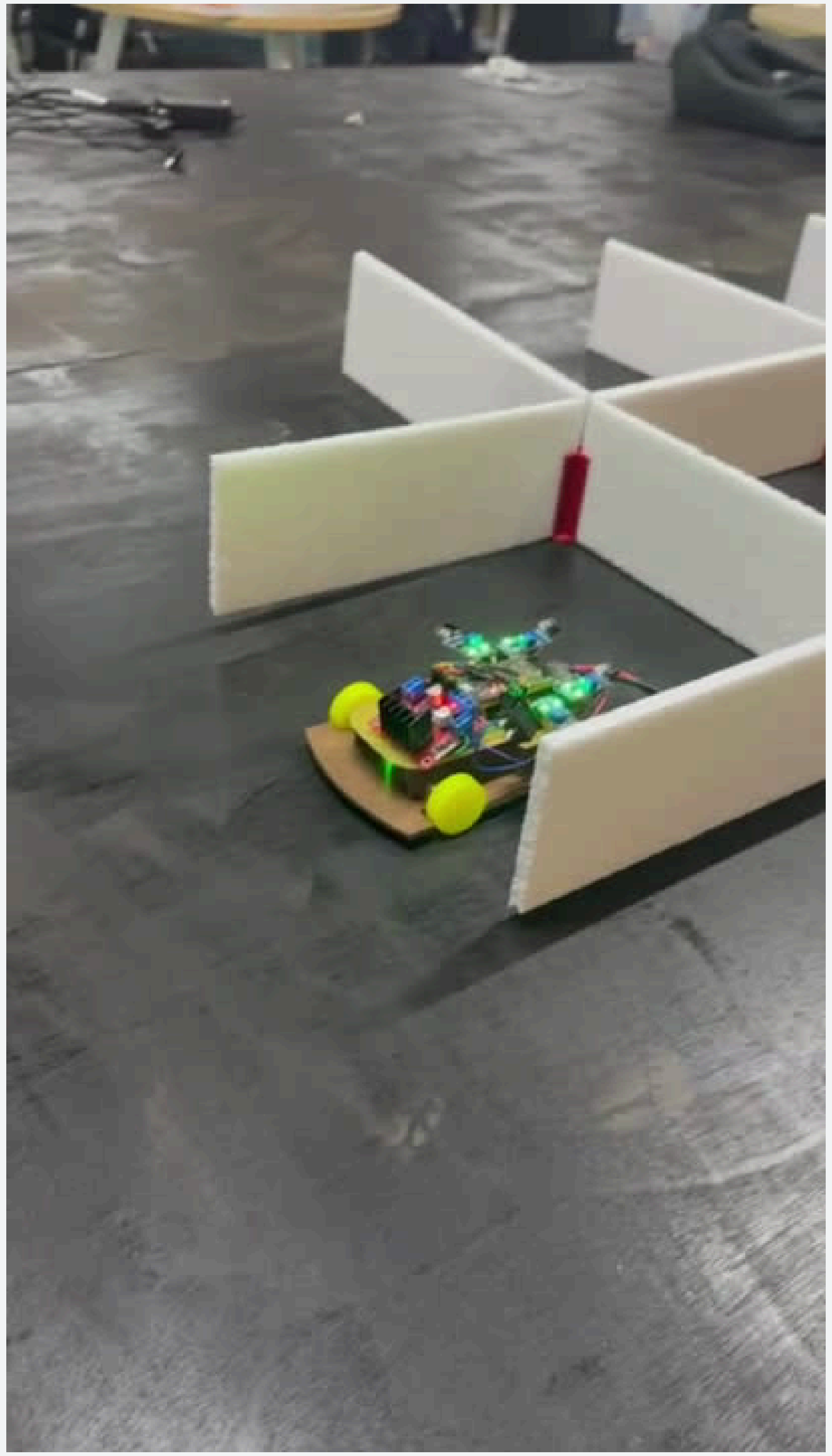




Final Assembly

SECOND PROTOTYPE TESTING





FLOOD FILL ALGORITHM

- **Maze Representation:** The maze is represented as a grid, with each cell containing information about surrounding walls and its current state.
- **Wall Detection:** The algorithm checks for walls on the left, right, and front sides of the current position in the maze.
- **Flood Fill:** The flood fill algorithm is used to calculate the shortest path from a starting point to a destination. It assigns values to each cell indicating the distance from the starting point.

- **Orientation and Movement:** The algorithm maintains the orientation of the agent (e.g., north, east, south, west) and determines the direction to move based on the shortest path calculated by flood fill.
- **Turning and Forward Movement:** Depending on the direction determined by the flood fill, the agent turns left, right, or moves forward to navigate through the maze.
- **Updating Maze State:** As the agent moves, it updates the state of the maze, including the walls detected and the cells visited.

- **Iterative Exploration:** The algorithm may employ iterative exploration strategies, such as visiting multiple destinations or optimizing the path to minimize movement.
- **Dynamic Path Planning:** It dynamically adjusts its path based on the current state of the maze and any changes in the environment (e.g., newly discovered walls).

CODE SNIPPET

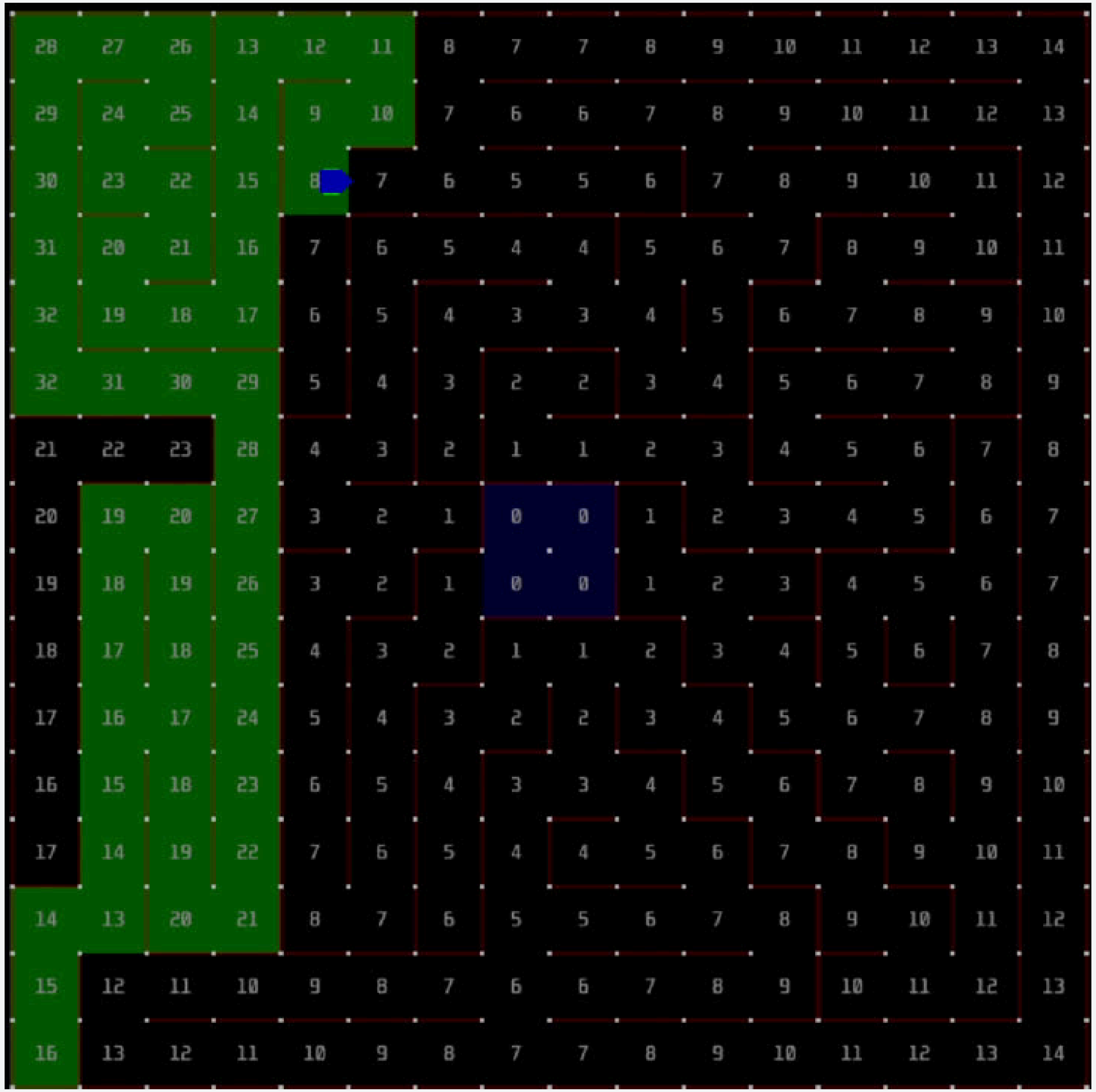
```
bool changed = true;
while (changed) {
    changed = false;
    for (int i = 0; i < MAZE_MAX_SIZE; i++) {
        for (int j = 0; j < MAZE_MAX_SIZE; j++) {
            if (flood_vals[i][j] != INF) {
                Cell current = {i, j};
                Cell* neighbours = getNeighbours(current);
                for (int orient = 0; orient < 4; orient++) {
                    Cell neighbour = neighbours[orient];
                    if (neighbour.x >= 0 && neighbour.x < MAZE_MAX_SIZE &&
                        neighbour.y >= 0 && neighbour.y < MAZE_MAX_SIZE &&
                        !isWall(current, orient)) {
                        int new_dist = flood_vals[current.x][current.y] + 1;
                        if (new_dist < flood_vals[neighbour.x][neighbour.y]) {
                            flood_vals[neighbour.x][neighbour.y] = new_dist;
                            path_info[neighbour.x][neighbour.y].prev = current;
                            path_info[neighbour.x][neighbour.y].dist = new_dist;
                            changed = true;
                        }
                    }
                }
                free(neighbours); // Free the allocated memory for neighbours
            }
        }
    }
}
```

INTEGRATION WITH SIMULATOR

```c
#pragma once

int API_mazeWidth();
int API_mazeHeight();

int API_wallFront();
int API_wallRight();
int API_wallLeft();

int API_moveForward();  // Returns 0 if crash, else returns 1
void API_turnRight();
void API_turnLeft();

void API_setWall(int x, int y, int direction);
void API_clearWall(int x, int y, char direction);

void API_setColor(int x, int y, char color);
void API_clearColor(int x, int y);
void API_clearAllColor();

void API_setText(int x, int y, char* str);
void API_clearText(int x, int y);
void API_clearAllText();

int API_wasReset();
void API_ackReset();
```

- Including the header file containing the input and outputs from the simulator.
- This allows us to interact with the simulation environment.

SIMULATION



CONVERSION TO ESP32 SUPPORTED CODE

- Converting the functional definitions for interacting with the hardware
- ESP32 works as a bridge between IR sensors and the Motor Driver

```
void API_moveForward(){
  runMotor(M1,100);
  runMotor(M2,100);
  delay(1000);
}

void API_turnLeft(){
  runMotor(M1,100);
  runMotor(M2,0);
  delay(1000);
}

void API_turnRight(){
  runMotor(M1,0);
  runMotor(M2,100);
  delay(1000);
}

bool API_wallFront(bool ir2, bool ir3){
  bool wallf = ir2 || ir3;
  return wallf;
}

bool API_wallLeft(bool ir1){
  return ir1;
}

bool API_wallRight(bool ir4){
  return ir4;
}
```

NON FUNCTIONING STATE IN PHYSICAL ENVIRONMENT

- Due to the delays we faced in receiving the hardware components, we were not able to make the fully working bot due to the lack of time
- Also we were not able to perform enough test and trials in the short span of time during examinations

THANK YOU