Design and Implementation of a Line Following and Obstacle Avoidance Robot

Md Akram, Onyesi John Abiagam, Omar Latif
Systems Engineering and Prototyping, BSc Electronic Engineering
Hochschule Hamm-Lippstadt, Lippstadt, Germany
{md.akram, onyesi-john.abiagam, omar.abdellatif}@stud.hshl.de

***Abstract*—This document illustrates an autonomous robot capable of line following, obstacle detection and color-based obstacle handling that we designed, developed and implemented as part of our Prototyping and System Engineering program. The robot has a variety of sensors and actuators such as IR sensors for line detection, ultrasonic sensors for obstacle detection and color sensors for obstacle colors identification. A particular behavior is triggered by each color like red for avoidance, blue for pushing, and green for parking. We utilized structured systems engineering approaches including several SysML diagrams to analyze and model the overall system. To mount the electronics components with the chassis, we designed custom 3D-printed parts in SolidWorks. We initially developed and tested all unit tests and the master code in Tinkercad to minimize risks. Additionally, we modeled the overall system on UPPAAL to show the timed behavior of the system. In real-world scenarios, the finished robot effectively displayed autonomous operation with precise line following and dynamic obstacle handling.**

***Index Terms*—Line follower robot, Obstacle avoidance, Color identification**

I. Introduction

This paper details the design, development, and implementation of an autonomous robot capable of precise line following, dynamic obstacle detection, and intelligent color-based obstacle handling. Developed as a culminating project in our Prototyping and System Engineering program, this work demonstrates the practical application of structured systems engineering principles, integrating both hardware and software components into a robust embedded system.

The robot is equipped with a variety of sensors and actuators such as IR sensors for line detection, ultrasonic sensors for identifying obstacles, and color sensors for distinguishing obstacle types. A key feature of the robot is its ability to trigger specific behaviors—such as avoidance for red objects, pushing for blue, and parking for green—based on the detected color through pre-programmed logic.

Our methodology employed structured systems engineering approaches, including the use of SysML diagrams for comprehensive system analysis and modeling. Custom 3D-printed parts, designed in SolidWorks, were utilized for seamlessly mounting electronic components onto the robot's chassis. To mitigate risks and ensure functionality, initial unit tests and master code were rigorously developed and tested in Tinkercad. Furthermore, the entire system's timed behavior was modeled and verified using UPPAAL, allowing for thorough simulation before physical implementation.

This paper is organized as follows: Section II outlines the system design and engineering, including formal requirements, state machine, and sequence diagrams. Section III explains the prototyping and design process, covering electronic components, virtual, and hardware assembly. Section IV discusses the circuit design. Section V details the Tinkercad simulation and virtual prototyping. Section VI presents the UPPAAL implementation for modeling and verifying timed behavior. Section VII discusses the software implementation for line following, obstacle detection and avoidance, and color-based obstacle handling. Section VIII covers Git usage and collaboration for project management. Finally, Section IX concludes the paper with key achievements.

A. System Requirements Overview

The system requirements for the autonomous vehicle are modeled using a SysML Requirement Diagram, which provides a hierarchical breakdown of the vehicle's expected functionalities. At the top level, the system is required to follow a line, detect and navigate around obstacles, and maintain its line-following behavior. This overarching goal is decomposed into five major requirement domains: line tracing, obstacle detection, navigation, parking, and driving route.

The **line tracing** requirement ensures that the car can follow a predefined path using IR sensors. This includes both the functional requirement to follow the line and the supporting requirement to calibrate the IR sensor. A corresponding test case verifies IR sensor performance under varying lighting conditions to ensure robustness.

The **obstacle detection** requirement mandates that the car should be capable of detecting objects using an ultrasonic sensor. Sub-requirements include detection within a specified range, as well as sensor calibration to maintain accuracy. A dedicated test case is provided to validate the sensor's ability to detect objects within a range of 10 to 100 cm.

For **navigation**, the vehicle must maneuver around detected obstacles and resume its line-following path. This involves both obstacle avoidance and path resumption. A test case is linked to verify the vehicle's ability to navigate safely around obstacles.

The **parking** functionality requires the vehicle to identify parking zones based on color detection. It uses a color sensor to determine when and where the car should stop. This is refined further into a requirement specifying color-based

decision-making for parking, and the associated test case verifies the system's ability to recognize the correct color and initiate parking behavior.

Lastly, the **driving route** requirement tests the vehicle's ability to follow complex paths. These include making a 90-degree turn, navigating in an eight-shaped path, and following an oval path. These requirements validate the vehicle's ability to handle non-linear trajectories, ensuring flexible movement in dynamic environments.

Overall, the diagram captures the logical refinement of system-level requirements into concrete functional expectations and corresponding test cases, supporting a traceable and structured approach to system implementation and verification.

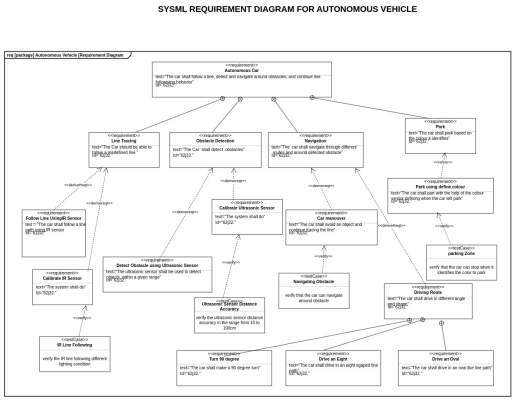

Fig. 1: SysML Requirement Diagram for Autonomous Vehicle

B. State Machine

One of the first things we created was a state machine diagram. This shows the different states our line-following robot can be in—like following the line, stopping when there's an obstacle, reacting to colors on the path, and so on. It helped us understand how the robot will switch between these states based on what the sensors detect.

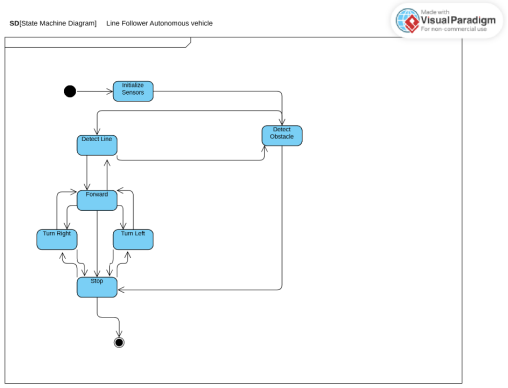

Fig. 2: State machine diagram showing the different robot behavior states.

C. Sequence Diagram

Next, we built a sequence diagram. This was useful for showing how all the components of our robot—like the two IR sensors, ultrasonic sensor, color sensor, motor driver, and the two motors—communicate with each other over time. Since all these parts need to work together smoothly for the robot to follow the line safely and efficiently, the sequence diagram helped us plan the order and timing of their actions.

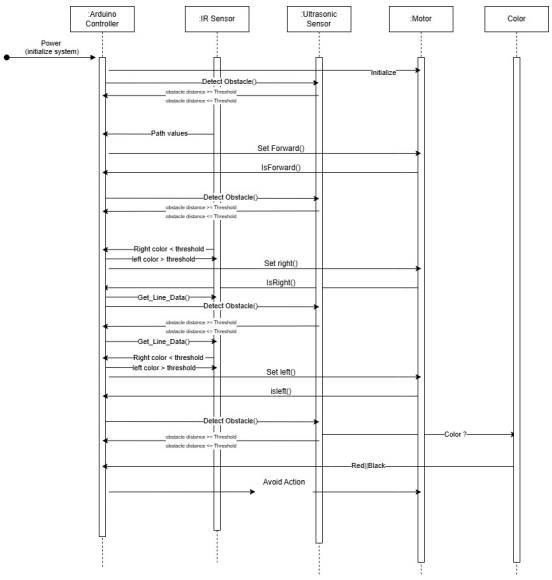

Fig. 3: Sequence diagram showing the different robot behavior states.

To take it a step further, we used a tool called UPPAAL to simulate our system. We turned our sequence diagram into something called timed automata—basically a way to model how our robot behaves over time. With UPPAAL, we could run simulations and check if everything works correctly and on time, even before building the real robot. In short, by designing the system first—with diagrams and simulations—we made sure we had a solid plan. This helped us understand the robot's behavior better and gave us more confidence moving forward into the implementation phase.

II. Prototyping and Design

A. Electronic Components

This section provides a detailed overview of each component used in the autonomous line-following car, describing its purpose, operating principle, and how it integrates into the overall system.

1) Arduino UNO **Description:** The Arduino UNO is an open-source microcontroller board based on the ATmega328P. It features 14 digital I/O pins, 6 analog inputs, and a 16 MHz quartz crystal.
Working Principle: The microcontroller executes the embedded code uploaded from the Arduino IDE. It reads input signals from sensors (IR and ultrasonic), processes

the logic, and generates output signals to control the motors via the motor driver.
Role in System: Acts as the central processing unit of the robot, handling sensor readings, motor control, and decision-making.

Fig. 4: Arduino UNO Integration

2) L298N Motor Driver Module **Description:** The L298N is a dual H-bridge motor driver IC capable of driving two DC motors. It can control direction and speed using digital signals and PWM.
Working Principle: Each motor is connected to an H-bridge that allows current to flow in either direction, enabling forward and reverse motion. The enable pins (ENA, ENB) receive PWM signals to vary speed. Input pins (IN1–IN4) control the rotation direction.
Role in System: Interfaces between the Arduino and DC gearmotors. It amplifies the low-power control signals from the Arduino to drive higher-current motors.

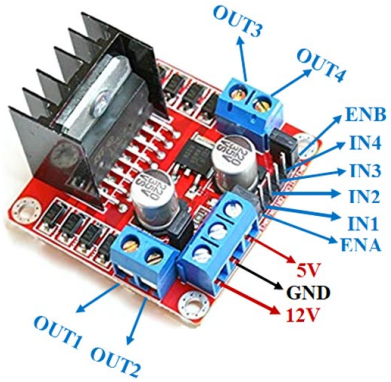

Fig. 5: L298N Motor Driver Module

3) DC Geared Motors (x2) **Description:** These are 6V DC gear motors, typically with high torque and low RPM, used to drive the robot's wheels.
Working Principle: A gear train attached to a standard DC motor reduces speed and increases torque. Polarity of voltage across terminals determines direction.
Role in System: Provide the driving force for the robot's movement. Controlled via the L298N motor driver.

Fig. 6: DC Geared Motors

4) IR Line Sensors (x2) **Description:** IR line sensors are modules that consist of an IR transmitter and receiver. They detect reflective contrast (usually between a black line and white surface).
Working Principle: The IR LED emits infrared light. If the surface reflects (e.g., white), the receiver detects the signal. If the surface absorbs IR (e.g., black line), the receiver gets little to no signal. The sensor outputs HIGH (line not detected) or LOW (line detected) based on the reflection.
Role in System: Used for line detection. The readings from these sensors guide the robot to follow a predefined path marked by a black line.

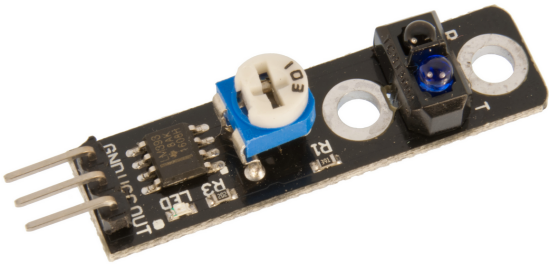

Fig. 7: IR Line Sensor Integration

5) HC-SR04 Ultrasonic Sensor **Description:** A distance-measuring sensor that uses ultrasonic waves to detect obstacles in front of the robot.
Working Principle: The TRIG pin sends out an ultrasonic pulse. The ECHO pin receives the reflected signal. The time taken for the pulse to return is used to calculate distance using the formula:

$$\text{Distance (cm)} = \frac{\text{Time}(\mu s)}{58}$$

Role in System: Provides obstacle detection. If an obstacle is within a defined threshold distance, the robot can stop or perform avoidance maneuvers.

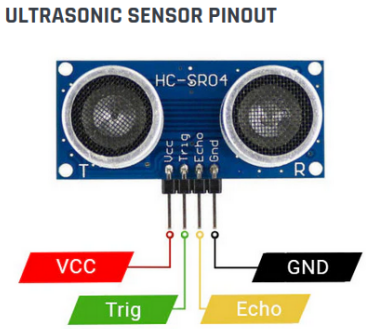

Fig. 8: HC-SR04 Sensor Placement

6) Switch **Description:** A simple on/off switch used to control the power flow from the battery pack to the rest of the circuit.
Working Principle: Acts as a mechanical gate that either completes or interrupts the electrical circuit.
Role in System: Serves as the main power control for the robot, allowing safe shutdown and startup.

Fig. 9: Switch

7) Lipo Battery **Description:** A Lipo battery was used to supply approximately 7.4V power.
Working Principle: Provides DC voltage to the motor driver, which then powers the motors and the Arduino UNO (via onboard voltage regulator).
Role in System: Primary power source for the entire robot.

8) Jumper Wires and Breadboard **Description:** Jumper wires are used to make temporary or semi-permanent connections between components. A breadboard can be used for prototyping.

Fig. 10: Lipo Battery

Role in System: Enable all electrical connections without soldering. Helps route signals and power between modules.

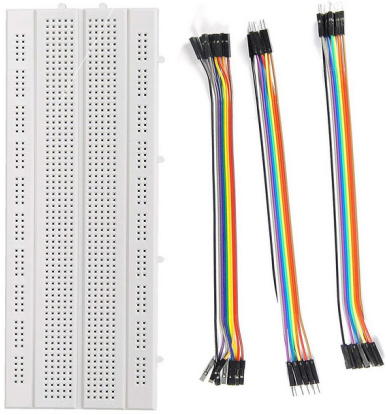

Fig. 11: Breadboard and Jumper Wires

9) TCS3200 Color Sensor **Description:** The TCS3200 is a color recognition sensor capable of detecting a wide range of colors using an array of photodiodes filtered for red, green, blue, and clear light.
Working Principle: The sensor has four types of photodiodes, each sensitive to red, green, blue, or no filter (clear). Control pins (S2 and S3) select which color filter is active. The sensor outputs a square wave on the OUT pin, with frequency proportional to the intensity of the selected color. The Arduino measures this frequency using `pulseIn()` or timers to determine the dominant color.
Role in System: Used to identify the color of objects in front of the robot. Based on detected color (e.g., red = parking zone, green/blue = obstacle), the robot can make intelligent decisions like stopping, avoiding, or parking.

10) Servo Motor (SG90 or Similar) **Description:** A small, lightweight servo motor used for angular positioning, commonly featuring 180° of rotation.
Working Principle: Operates using a control signal in the form of PWM. The duration of the HIGH pulse (usually between 1ms and 2ms) determines the angle of the servo horn. The servo maintains its position using internal feedback control.

Fig. 12: TCS3200 Color Sensor

Role in System: Mounted with the ultrasonic sensor to allow left-right scanning for obstacle detection. During avoidance, the servo rotates the sensor to check distances in different directions, helping the robot decide the safest path forward.

Fig. 13: Servo Motor

B. Virtual and Physical Prototypes

Since our electronic sensors and actuator requirements were clear so to transform from concept to real implementation, we initially need to design a prototype. We designed all virtual prototypes by using SolidWorks, a professional 3D computer-aided design (CAD) software [1]. It was very critical to design a precise and suitable 3D design that makes the connection between the components with the chassis of the robot.

1) Custom 3D Parts: To make our robot functional we designed total of five custom 3D parts including:
   a) Motor Holder:

      We used two DC geared motors, which were possible to connect with two wheels, but to connect those geared motors with the chassis, we needed to design holders. So, for two geared motors, we designed two precise and perfect motor holders. The width and height were 34 mm and 23 mm, respectively, while the big hole was 9 mm and the small hole was 5 mm in diameter. To connect the

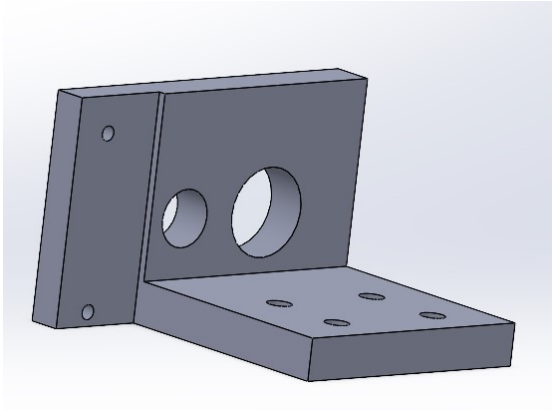

Fig. 14: DC Geared Motor Holder

motor holder with the geared motor and chassis, we used two and four screw holes respectively.

   b) Ultrasonic Sensor Holder:

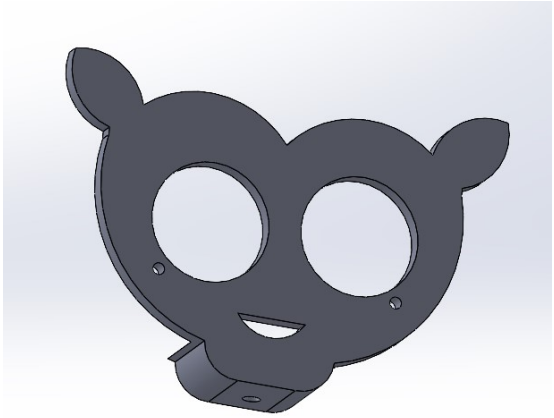

Fig. 15: Ultrasonic Sensor Holder

      We used one ultrasonic sensor for detecting obstacles. So, to attach that ultrasonic sensor with servo motor we designed one ultrasonic sensor holder. The diameter of two holes was 20 mm, while the distance was 26 mm, which ensured that the sensor will fit with that holder. Finally, to attach the ultrasonic sensor holder with the servo, we made a 2 mm diameter screw hole on the downside of the holder.

   c) Servo holder:

      If any obstacles were detected on the front then we need to check the side clearance to avoid the obstacles from the right or left side. So, to rotate the ultrasonic sensor to the right or left, we need one servo motor and to mount the servo with the chassis we designed one servo holder. The height and width were 27 mm and 21 mm respectively. There were total 4 screw holes of 2 mm diameter,

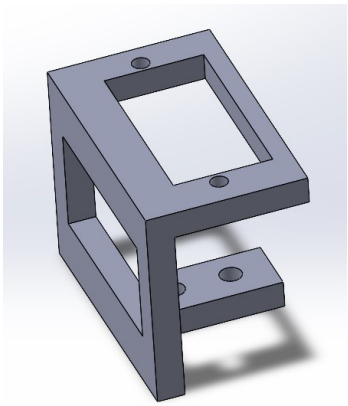

Fig. 16: Servo Holder

which were used to connect the holder with the servo and chassis.

d) Battery holder:

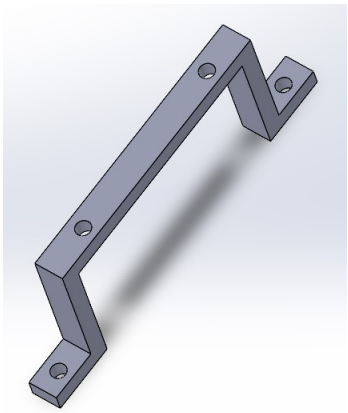

Fig. 17: Battery Holder

we used a Lipo battery to ensure the continuous power supply for the DC geared motor and micro-controller. So to make the battery stable with the chassis, we used two battery holders whose height and width were 47 mm and 25 mm respectively.

e) Color Sensor Holder:

To identify the color of obstacles, we need to use a color sensor and to connect the color sensor with the chassis, we designed one color sensor holder. For analysis of the color, the color sensor needs to get enough replicated light from obstacles that's why we give the shape of the color sensor something like a basket. The height and width were 29 mm and 46 mm respectively.

2) Virtual Assembly: we know that for a proper mechanical connection, all dimensions need to be exactly accurate. Otherwise, those parts will not fit with those components and that can affect the performance of the overall system.

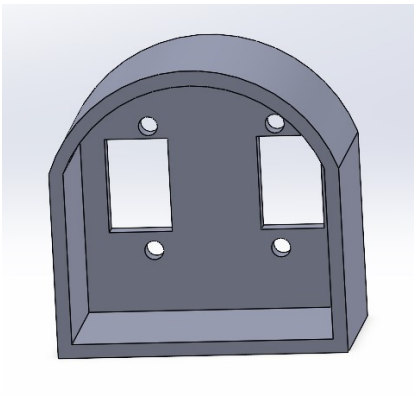

Fig. 18: Color Sensor Holder

To ensure that all 3D-designed parts were suitable and perfect with those components and chassis, we made a virtual assembly, which provided an opportunity to recheck and verify our 3D designs. The virtual assembly also gives us knowledge about physical assembly after getting the 3D parts.

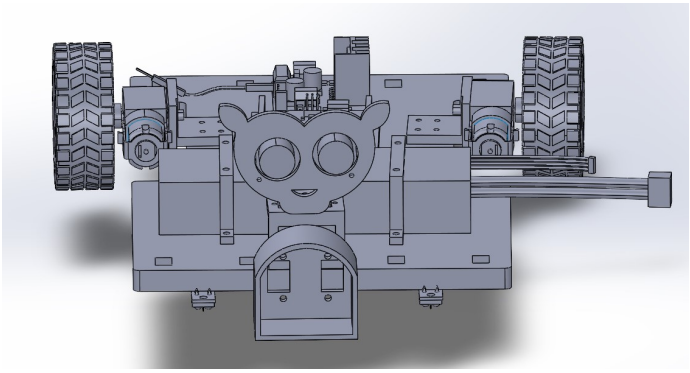

Fig. 19: Virtual Assembly

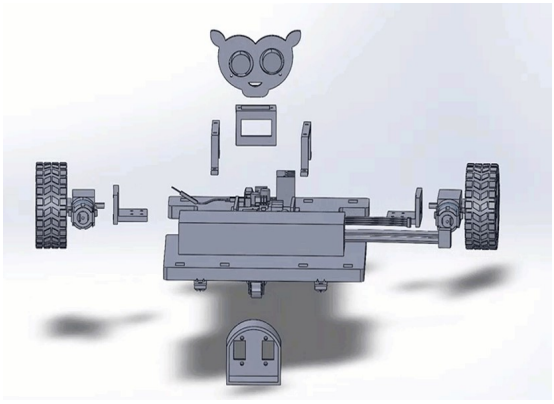

Fig. 20: SolidWork Exploded view

C. Hardware Assembly

In this section, I describe the hardware setup process I followed to assemble and wire the components of my autonomous line-following car. This setup uses an Arduino UNO as the central controller, along with an L298N motor driver, IR line sensors, and an HC-SR04 ultrasonic sensor for obstacle detection. The steps outlined here assume a basic working knowledge of microcontroller interfacing and electronics.

1) Setting Up the Arduino UNO To begin, I securely mounted the Arduino UNO onto the robot chassis using screws and plastic standoffs. This ensured that the board stayed stable during motion and allowed easy access to I/O pins.
2) Connecting the L298N Motor Driver The L298N motor driver is responsible for driving the two DC gearmotors. I took extra care during power and signal wiring to ensure proper operation.
3) Power Connections
   - **Battery to Rocker Switch:** I connected the positive wire (+) from the battery holder to the input terminal of the rocker switch.
   - **Common Grounding:** The negative wire from the battery pack was connected to the GND pins of both the L298N and Arduino UNO.
   - **Switch to Motor Driver:** The output of the rocker switch was connected to the VCC pin on the L298N to supply motor voltage.
   - **Powering Arduino:** I used the 5V output from the L298N to power the Arduino UNO through its 5V pin. (*Note: I avoided USB or external power simultaneously to prevent voltage conflict.*)
4) Motor Connections
   - **Left Motor (Motor A):** Connected to OUT1 and OUT2.
   - **Right Motor (Motor B):** Connected to OUT3 and OUT4.
5) Control Signal Wiring
   - **Enable Pins:** ENA → D5, ENB → D6 (for PWM speed control).
   - **Direction Pins:** IN1 → D12, IN2 → D13, IN3 → D10, IN4 → D11.
6) Connecting the IR Line Sensors I installed two IR sensors at the front underside of the robot chassis, making sure they were close enough to detect line contrast on the floor.
   - **Left Sensor:** VCC → 5V, GND → GND, OUT → D4.
   - **Right Sensor:** VCC → 5V, GND → GND, OUT → D2.
7) Connecting the HC-SR04 Ultrasonic Sensor The ultrasonic sensor was mounted front-facing on the chassis. For optimal performance, I aligned it horizontally.
   - VCC → 5V, GND → GND
   - TRIG → D9, ECHO → D8
8) Power Supply Setup After inserting the lipo battery (providing roughly 7V) into the battery holder, I ensured the switch was in the *off* position before making the final connections. Once everything was connected properly, I used the switch to safely turn the system on and off.
9) Grounding Note: To maintain a stable and consistent operation, I connected all ground lines from the Arduino, motor driver, sensors, and battery holder together. This common ground reference is critical for reliable communication between the modules and for avoiding unexpected behavior.
10) Final Prototype: After assembly all components with chassis then we connected the electrical connection according the Circuit diagram which available on Circuit Design section (Figure 19). On the below figure we can see our final prototype which is ready for testing on real implementation.

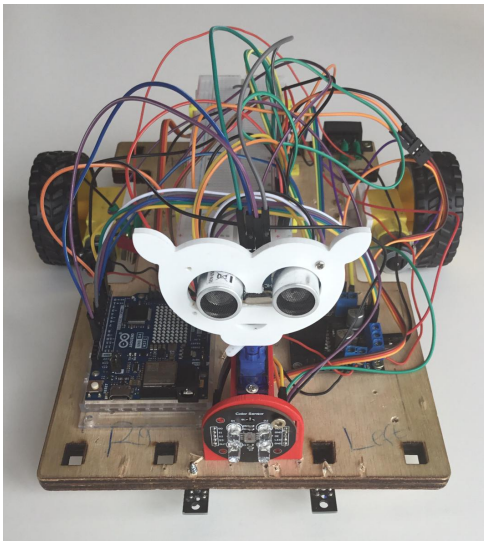

Fig. 21: Final assembled autonomous line-following car with all hardware components mounted.

III. Circuit Design

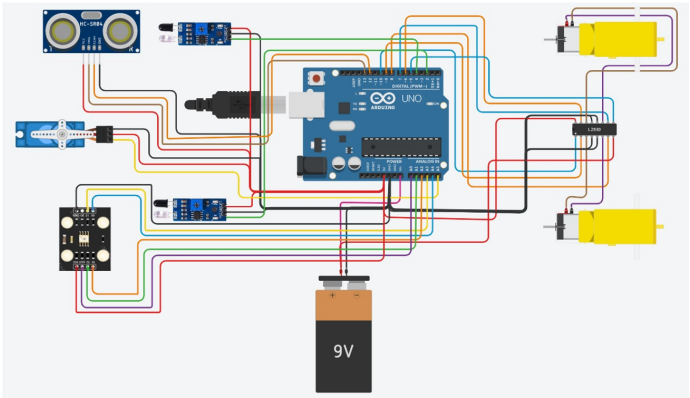

Fig. 22: Circuit Diagram

Circuit design was very crucial parts for making the robot functional. We made unit test one by one so wiring was not much complex for us. According our master code we connecting all components with microcontroller. This is the overall schematic diagram for full system.

IV. Thinkercad Simulation

Tinkercad Circuits is a commonly usable tool for simulation and virtual prototyping during a project implementation [2]. After sending our 3D design for printing in the lab, meanwhile, we started working on the Tinkercad programming and simulation part. Iterative and risk-averse testing of the robot's functionalities was made possible by this platform. The structure of the simulation process was as follows:

1) Individual Component Testing: Since in Tinkercad we have the facilities to program and visualize the behavior of the component as output in simulation that's why we write separate programs for each component including IR sensor, Ultrasonic sensor, Dc geared motor and motor driver. Sometimes because of wrong wiring there is a possibility of damage to electronic components and due to incorrect programming, the expected output is not possible to achieve. So Tinkercad helped us to check the correctness of wiring and understand the behavior of each component.

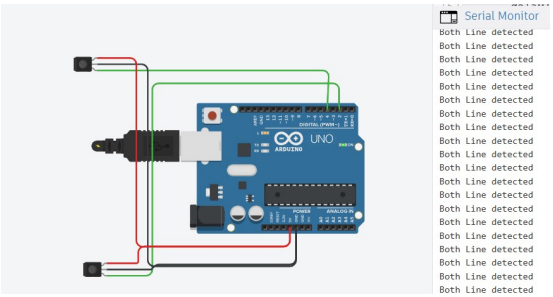

Fig. 23: IR Testing Simulation

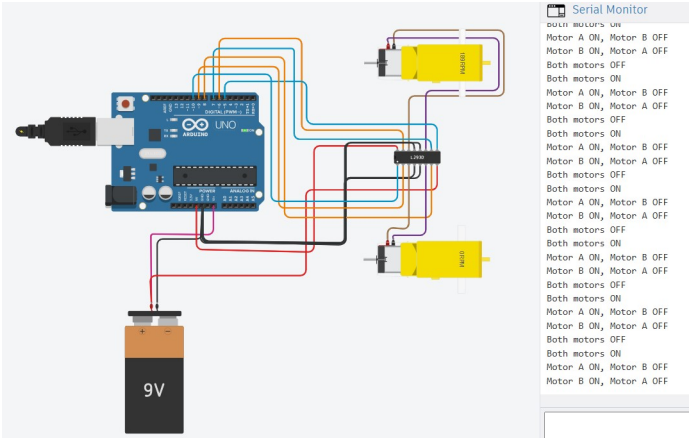

Fig. 24: Motor Driver Testing Simulation

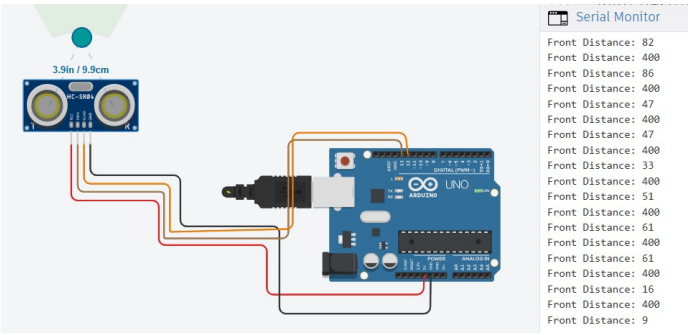

Fig. 25: Ultrasonic Sensor Testing Simulation

2) Simulation and Hardware Validation:

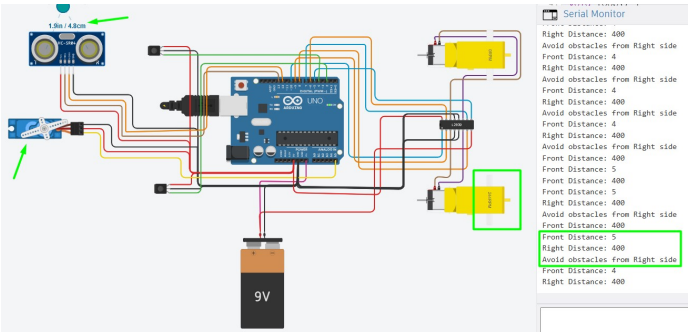

Fig. 26: Thinkercad Simulation

Once the Thinkercad showed our expected output on simulation and the serial monitor, this time we saved the code on a separate file and then tested that program on real hardware. So when real hardware and simulation showed the same result then we decided that the hardware was functional and we reserved the code for using on master code.

3) Combined Master Code Simulation: After the unit test, we verified that all components are working perfectly and then we started working on making a master code step by step from our reserved code of the unit test. Initially we combined IR sensor and motor driver program together for line-following functionality. Since line following worked, we added Ultrasonic sensor code to detect obstacles and stop. Once we saw that our robot was able to follow the line and detect the obstacles then we merged the obstacle avoidance program with the previous code including the servo and left-side or right-side avoidance function. So whenever obstacle was detected within 10 cm then the robot stopped and always checked the right side first. If the right side was clear, then it always avoided it from the right, which saved time. In case the front and right sides were blocked then it checked the left side if that side was free then it avoided obstacle from left side. At the time when front, right and left were all blocked, this time the robot went backward for enough space and avoided that from right

side. Here is the simulation of the master code with all available components on Tinkercad for our robot.

V. UPPAAL Implementation

After completing the system design through state and sequence diagrams, the next step was to test and simulate the behavior of our robot using UPPAAL. This tool allowed us to model and verify the time-based interactions between the components by creating a set of timed automata that reflect the logic defined in our sequence diagram. In UPPAAL, we represented each key component or behavior of the robot as a separate automaton. These automata interact with each other to simulate how the robot would behave in real-world scenarios. This includes transitions between states such as following the line, detecting an obstacle, responding to specific colors, and stopping or changing direction. We used **guards** to specify when a transition between states is allowed. These guards are based on conditions such as elapsed time or sensor inputs. For example, a guard might ensure that the robot only moves to the next state after a sensor detects a line or a certain time has passed. In addition, we applied **invariants** to control how long the system can remain in a particular state. This helps enforce timing constraints—such as how long the robot should wait before rechecking sensor data or how quickly it should respond to obstacles. The figure below shows a screenshot of our UPPAAL model in action, with all automata working together to simulate the robot's behaviour over time:

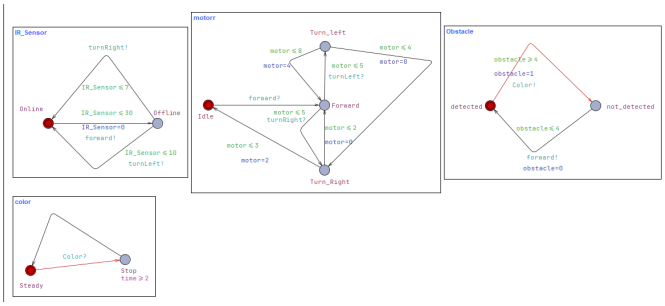

Fig. 27: UPPAAL simulation model showing the timed automata for the robot's behavior.

By using UPPAAL, we were able to simulate the full behavior of our robot and verify that the timing and communication between components were working correctly. This step was crucial in validating our design before moving on to physical implementation.

VI. Software Implementation

A. Line Following

Line following is one of the major parts of our project. So initially we worked on line following and when out robot was able to follow the line then we integrated others functionalities with line following program. Here is the details about line following:

1) Initialization: To perform line-following, two infrared (IR) sensors were used and initialized as digital input pins (L S and R S). These sensors detect the black line by identifying the reflectivity of the surface.

```
void setup() {
  pinMode(L_S, INPUT);
  pinMode(R_S, INPUT);

  pinMode(ENA, OUTPUT); pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(ENB, OUTPUT); pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(servo, OUTPUT);
  pinMode(S0_PIN, OUTPUT);
  pinMode(S1_PIN, OUTPUT);
  pinMode(S2_PIN, OUTPUT);
  pinMode(S3_PIN, OUTPUT);
  pinMode(OUT_PIN, INPUT);
  digitalWrite(S0_PIN, HIGH);
  digitalWrite(S1_PIN, LOW);

  // Initialize servo to face forward
  servoPulse(servo, 70);
  digitalWrite(S0_PIN, HIGH);
  digitalWrite(S1_PIN, LOW);
  Serial.begin(9600);
  delay(1000);
}
```

Fig. 28: IR Sensor Initialization

2) Movement control:

```
void forward() {
  analogWrite(ENA, speedL); analogWrite(ENB, speedR);
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
}
void backward() {
  analogWrite(ENA, speedL); analogWrite(ENB, speedR);
  digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
}

void turnRight() {
  analogWrite(ENA, speedL); analogWrite(ENB, speedR);
  digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
}

void turnLeft() {
  analogWrite(ENA, speedL); analogWrite(ENB, speedR);
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
}

void Stop() {
  analogWrite(ENA, 0); analogWrite(ENB, 0);
  digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
}
```

Fig. 29: Line sensor Movement control

According the IR sensor value the movement functions including forward(), turnLeft(), turnRight()and Stop()were implemented using PWM control via the L298N motor driver. We used sperate speed value for each motor for smooth maneuvering on the track.

3) Behavioural Decision logic: According the IR sensor value we give total four type of behavior of our robot. Those are:
   a) Both sensors on line: move forward.
   b) Only left sensor on line: turn right.
   c) Only right sensor on line: turn left.
   d) No sensor detects line: stop.

B. Obstacles Detection and Avoidance

An essential part of our robot's functionality is its ability to detect and avoid obstacles while following the line. Without this feature, the robot could easily collide with objects on the path, making it unreliable in real-world scenarios. To solve this, we integrated an ultrasonic sensor into the system to

```
void loop() {
  distance_F = Ultrasonic_read();
  Serial.print("Distance Forward = ");
  Serial.print(distance_F);
  Serial.println(" cm");
  int leftSensor = digitalRead(L_S);
  int rightSensor = digitalRead(R_S);

  if ((rightSensor == 0) && (leftSensor == 0)) {
    if (distance_F > Set) {
      forward();
    } else {
      Stop();
      delay(1000);
      int color = getColor();
      if (color == 1) {
        avoidObstacle();
        //red = 1 so aboidobstacle
      } else if (color == 2) {
        push();
        // blue = 2 so push
      } else if (color == 3) {
        ParkCar();  // Green = 3 so park
      }
    }
  } else if ((rightSensor == 1) && (leftSensor == 0)) {
    turnRight();
  } else if ((rightSensor == 0) && (leftSensor == 1)) {
    turnLeft();
  } else if ((rightSensor == 1) && (leftSensor == 1)) {

    Stop();
  }
```

Fig. 30: Line Following

constantly monitor the distance between the robot and any object in front of it.

The obstacle avoidance logic was also incorporated into our state machine. When the ultrasonic sensor detects an object within a certain threshold distance, the robot transitions from the *Line Following* state to an *Obstacle Detected* state. In this state, the robot temporarily stops or takes a predefined action, such as waiting, turning, or navigating around the object.

On the Arduino side, we implemented a dedicated function to perform the avoidance maneuver. The function stops the robot, turns, moves forward, and reorients itself to get back on track. Below is the code snippet used for obstacle avoidance:

Listing 1: Arduino function for obstacle avoidance

```
void avoidObstacle(){
  Stop();
  delay(300);
  aturnRight();
  delay(400);
  Stop();
  delay(400);
  aforward();
  delay(500);
  Stop();
  delay(200);
  aturnLeft();
  delay(400);
  Stop();
  delay(200);
  Reforward();
  delay(400);
}
```

In our UPPAAL model, this behavior was implemented using both guards and transitions. For example, a guard condition checks if the distance measured by the ultrasonic sensor falls below a certain value. If true, the automaton transitions to the obstacle-handling state. Once the path is clear again, the robot returns to the line-following state and continues its task.

This reactive behavior ensures that the robot can operate safely and efficiently, even in dynamic environments where obstacles may appear unexpectedly. By simulating this mechanism in UPPAAL, we were also able to verify the timing and correctness of transitions, ensuring the robot avoids obstacles within the expected time constraints.

C. Color Based Obstacles Handling

We used TCS3200 color sensor which give us opportunity to identify the color of the obstacles in front of it. Although it is possible to idenfiy a varity of color by using this sensor but here we were identified three colors including Red, Blue and Green. Here is the initialization, movement control, behavioural Decision logic for color sensor in our project.

1) Initialization: The TCS3200 color sensor is calibrated to detect red, green, and blue by filtering light and measuring frequency. The color sensor is initialized with output and filter selection pins:

```
void setup() {
  pinMode(L_S, INPUT);
  pinMode(R_S, INPUT);

  pinMode(ENA, OUTPUT); pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(ENB, OUTPUT); pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(servo, OUTPUT);
  pinMode(S0_PIN, OUTPUT);
  pinMode(S1_PIN, OUTPUT);
  pinMode(S2_PIN, OUTPUT);
  pinMode(S3_PIN, OUTPUT);
  pinMode(OUT_PIN, INPUT);
  digitalWrite(S0_PIN, HIGH);
  digitalWrite(S1_PIN, LOW);

  // Initialize servo to face forward
  servoPulse(servo, 70);
  digitalWrite(S0_PIN, HIGH);
  digitalWrite(S1_PIN, LOW);
  Serial.begin(9600);
  delay(1000);
}
```

Fig. 31: Color Sensor Initialization

2) Movement control:
   According the color of obstacles we give three quique movements. I noticed the Red color detection accouracy is very high then Blue and Green respectively. So according that we give three unique movements to our robot. Those are:
   a) Push: Move forward briefly to push blue objects.
   b) Park: Turn, move backward and stop near green object.
   c) Avoidance: Use servo-based scanning and motion to avoid red objects.
3) Behavioural Decision logic: Once the object detected then by using getColor() function it identify the color of obstacles. Color detection is done by reading pulse values under each filter configuration and selecting the lowest frequency value like 1 for Red, 2 For blue and

```
void loop() {
  distance_F = Ultrasonic_read();
  Serial.print("Distance Forward = ");
  Serial.print(distance_F);
  Serial.println(" cm");
  int leftSensor = digitalRead(L_S);
  int rightSensor = digitalRead(R_S);

  if ((rightSensor == 0) && (leftSensor == 0)) {
    if (distance_F > Set) {
      forward();
    } else {
      Stop();
      delay(1000);
      int color = getColor();
      if (color == 1) {
        avoidObstacle();
        //red = 1 so aboidobstacle
      } else if (color == 2) {
        push();
        // blue = 2 so push
      } else if (color == 3) {
        ParkCar();  // Green = 3 so park
      }
    }
  } else if ((rightSensor == 1) && (leftSensor == 0)) {
    turnRight();
  } else if ((rightSensor == 0) && (leftSensor == 1)) {
    turnLeft();
  } else if ((rightSensor == 1) && (leftSensor == 1)) {

    Stop();
  }
```

Fig. 32: Color Sensor Movement control

```
void push(){
  // 2 blue p[sh
  pforward();
  delay(500); //need to adjust
  Stop();
  delay(500);

}
void ParkCar() {
  // 3 Green park
  aturnRight();
  delay(300); //changed from 800
  Stop();
  delay(1000);
  backward();
  delay(5000);
}
```

Fig. 33: Color Sensor Movement control

3 for Green. According the color of the obstacles, the behavior of the robot always different like it avoid red obstacles, push green obstacles and park the robot when identify Green obstacles.

VII. Git Usage and Collaboration

We used GitHub for collaboration and project management.

1) GitHub Commits Table: The below table represent the individual and total Commits for the project

```
int getColor() {
  int redValue, greenValue, blueValue;

  // Red
  digitalWrite(S2_PIN, LOW);
  digitalWrite(S3_PIN, LOW);
  redValue = pulseIn(OUT_PIN, LOW);
  // Green
  digitalWrite(S2_PIN, HIGH);
  digitalWrite(S3_PIN, HIGH);
  greenValue = pulseIn(OUT_PIN, LOW);
  // Blue
  digitalWrite(S2_PIN, LOW);
  digitalWrite(S3_PIN, HIGH);
  blueValue = pulseIn(OUT_PIN, LOW);
  Serial.print("Red: ");
  Serial.print(redValue);
  Serial.print("\tGreen: ");
  Serial.print(greenValue);
  Serial.print("\tBlue: ");
  Serial.println(blueValue);
  // Identify lowest value as dominant color
  if (redValue < greenValue && redValue < blueValue) {
    return 1; // Red
  } else if (blueValue < redValue && blueValue < greenValue) {
    return 2; // Blue
  } else if (greenValue < redValue && greenValue < blueValue) {
    return 3; // Green
  } else {
    return 0; // Unknown or balanced
  }
}
```

Fig. 34: Color identification

Team Member	Number of Commits
Md Akram	53
John Abiagam	23
Omar Latif	10
Total	**86**

TABLE I: GitHub Commits by Team Members

2) Github commits statistics: The below diagram shows the total overview of Github commits statistics over the time.

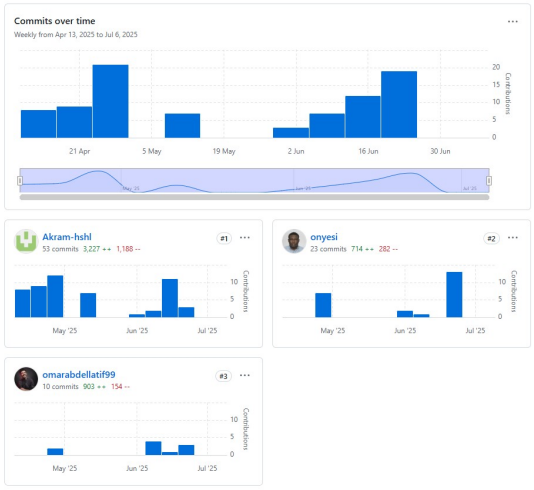

Fig. 35: Commits Statistics

3) Github Task Management system: To organize tasks among our group-mates we created a project within our Githum repositories which was really helpful for successfully finish our project.

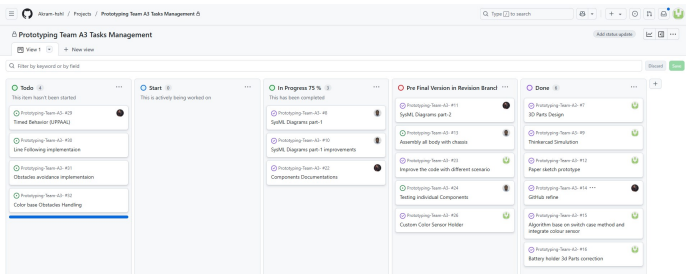

Fig. 36: Task Management

VIII. Key Achievements

For that project we did learn many professional and personal skills including programming, 3D design, GitHub management, Team work and communication which will be really effective for our future carrier. Although there was a lot of achievement but below I mentioned some remarkable achievements from that project:

1) Properly designed and implemented an autonomous Arduino-based robot capable line following, obstacle detection and color based obstacle handling.
2) Integrated a complex but functional system with several sensors such as IR, Ultrasonic, color sensor and motor drive with microcontroller.
3) Designed custom 3D printed parts for assembly everything.
4) Developed modular, well-structured Arduino code for precise and predefine behavior.
5) Implemented servo-controlled ultrasonic scanning for dynamic path planning and Color based dynamic obstacles handling features.

IX. Team Members and Their Roles

We are Team A3 having total 3 members. We all provided our best from our side to make the project successful. In below we discribed our major responsibilities and contribution for the project.

1) Md Akram: Designed the 3D prototype, Circuit diagram, Tinkercad simulation and Color based obstacles handling implementation.
2) John Abiagam: Responsible for drawing major SysML diagrams, mechanical assembly of components and Line Following implementation.
3) Omar Latif: Developed the UPPAAL model for system verification, some SysML diagram and Obstacles detection and avoidance implementation.

Acknowledgment

We would like to express our sincere gratitude to our professor and all instructors for their time, guidance and support to finish that project. We are also grateful to our teammates and all classmates for their help and cooperation. Finally we appreciate for all facilities and provided resources from our lab and university to make the project successful.

References

[1] solidwork crop(2023) Documentation. https://www.solidworks.com
[2] Tinkercad Simulation. https://www.tinkercad.com

Declaration of Originality

We hereby declare that this report is our own work and that we have acknowledged all sources used.