Design and Implementation of an Autonomous Arduino-Based Vehicle with Line Following and Obstacle Avoidance Capabilities

Team A2: Moeez Mufti (1233013), Muhammad Ali (1233053)
Systems Engineering and Prototyping, BSc Electronic Engineering
Hochschule Hamm-Lippstadt, Lippstadt, Germany
moeez.mufti@stud.hshl.de, muhammad.ali@stud.hshl.de
8th July, 2025

Abstract

This project presents the design, modelling, and implementation of an autonomous Arduino-based vehicle which should be capable of line following, obstacle detection, and color-based responses. Developed within the framework of a systems engineering and physical prototyping course, the vehicle integrates hardware modules including infrared sensors, an ultrasonic sensor, a colour sensor, dual DC motors, and a LiPo battery. The system architecture was modeled using SysML, covering structural (BDD, IBD), behavioural (state machine, activity, sequence), requirements and parametric aspects to ensure traceability from concept to realization. The mechanical components were custom-designed and 3D printed within strict time and design constraints.

CONTENTS

List of Figures

List of Tables

I. Introduction

The concept of an autonomous Arduino car merges the fields of robotics and embedded systems to develop an autonomous vehicle capable of navigating its environment without human intervention. Utilizing the Arduino Uno as the central control unit, the system integrates various sensors, actuators, and control logic to create a compact, self-driving car. Through the combination of hardware and software, the vehicle can perceive its surroundings, make real-time decisions, and perform actions such as line following, obstacle avoidance, and colour-based responses [1].

Creating a self-driving car with Arduino represents a compelling blend of technology and hands-on engineering. This project brings together multiple hardware and software components to achieve autonomous navigation in a compact, embedded system. At the heart of the system is an Arduino microcontroller, which acts as the central processing unit, coordinating sensor input, motor control, and decision-making logic. The development process involves careful hardware integration, structured software programming, and iterative testing to ensure reliable and safe operation [1].

Beyond the technical challenges, the project also involves the use of 3D modelling software to design custom holders for various components, ensuring precise mechanical integration. Additionally, SysML diagrams are created using Visual Paradigm to streamline our development approach, allowing us to align system requirements, structure our planning, and maintain traceability throughout the design and implementation process [2].

II. Systems Design and Engineering

This section outlines the requirements for the implementation of the complete system, detailing the functionalities that must be provided. This represents the final version of our requirements diagram which has been refined throughout the semester due to evolving requirements. *Refer to Appendix I for earlier iterations.*

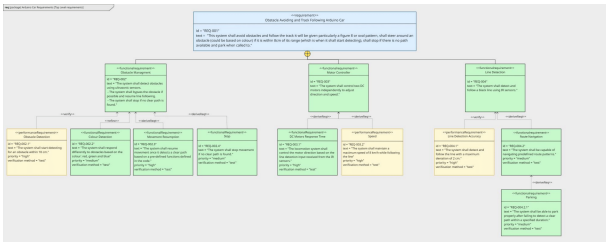


Fig. 1. Requirement Diagram

A. System Requirements

1) **Top-Level Requirement**

- The system shall avoid obstacles and follow a track. It shall steer around an obstacle (based on color) if it is within 15 cm of its range (the point at which detection starts), and shall stop if no path is available, or park when called to.

2) **Functional Requirements**

2.1) **Obstacle Management**

- **FR2:** The system shall detect obstacles using ultrasonic sensors, bypass obstacles if possible and resume line following. It shall stop if no clear path is found.
- **NF1:** The system shall start detecting obstacles within 10 cm.
- **FR2.2:** The system shall respond differently to obstacles based on their color: red, green, and blue.
- **FR2.3:** The system shall resume movement when a clear path is detected, based on pre-defined logic.
- **FR2.4:** The system shall stop movement if no clear path is found.

2.2) **Motor Controller**

- **FR3:** The system shall control two DC motors independently to adjust direction and speed.
- **FR3.1:** The locomotion system shall react to IR line detection input to adjust motor direction.

2.3) **Line Detection and Navigation**

- **FR4:** The system shall detect and follow a black line using IR sensors.
- **NFR4.1:** The system shall maintain a line-following deviation of no more than 2 cm.
- **FR4.2:** The system shall be capable of navigating predefined route patterns.
- **FR4.3:** The system shall be able to park properly after failing to detect a clear path within a specified duration.

B. Systems Architecture

The architecture of the autonomous Arduino-based vehicle is modular and structured around four primary subsystems: the **control unit**, **locomotion system**, **sensor system**, and **power system**.

As shown in the Block Definition Diagram, the **control unit** (Arduino Uno R4 WiFi) serves as the central processing element, interfacing with input sensors and actuating the motors via the motor driver. This represents the final version of our BDD. *Refer to Appendix II for earlier iterations.*

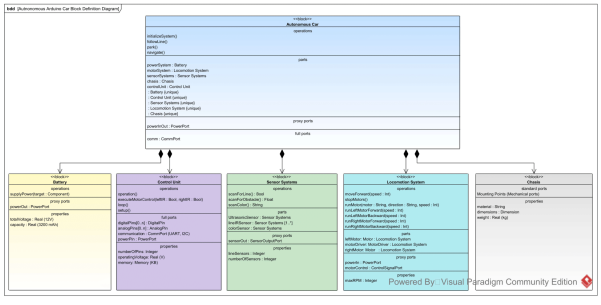


Fig. 2. Block Definition Diagram

The **locomotion system** consists of two DC motors, each independently controlled and powered through L298N motor

driver. Each motor is connected via two direction control pins (IN1/IN2 for the right motor, IN3/IN4 for the left motor) and one speed control pin (ENA for the right motor, ENB for the left motor). By setting the logic high or low on the direction pins, the motor can be made to rotate forward, backward, or stop. It also includes an L298N motor driver, which receives the input voltage and is responsible for controlling the motors' speed and direction while simultaneously supplying a regulated 5V output to power the Arduino.

The **sensor system** includes two infrared (IR) sensors for line detection which are placed closed to the ground and mounted on the frontside of the chassis to ensure maximum detection of the line; an ultrasonic sensor for obstacle avoidance which is mounted on a servo motor for rotation, and a colour sensor for contextual environmental analysis. These sensors send digital or analogue signals to the control unit. All three components are mounted together at the front of the chassis.

The **power system**, composed of a 12V LiPo battery, supplies electrical energy by connecting directly to the L298N motor driver, which in turn provides regulated 5V output to the Arduino via its `Vin` pin. From there, the Arduino's 5V output is connected to the breadboard's common power rail, distributing stable voltage to all sensors and auxiliary components through this centralized hub.

C. Behavioural Logic

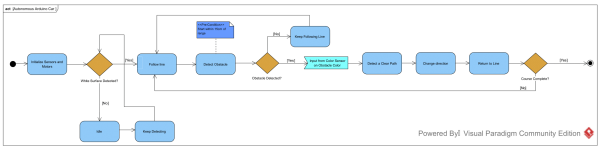


Fig. 3. Activity Diagram

The behavioral model, based on the **Activity Diagram**, defines the core operational flow of the vehicle: it starts with an initialization of its sensors and motors, then transitions into line following function: the IR sensors function by maintaining a set distance apart, with both sensors initially detecting the white surface to maintain straight-line movement. When a single IR sensor encounters a black line, the robot immediately turns in the direction of that sensor to follow the line's path. When both sensors simultaneously detect the black line, the robot stops completely, creating a simple yet effective line-following system that uses differential sensor feedback for navigation control. It will try to detect the line through continous detection.

While following the line, the ultrasonic sensor constantly monitors for obstacles and changes its path and navigates it paths back to the line through predefined angle movements.

For color sensor, we had planned to follow two paths for two types of colours: red and blue/green; for red, the car is to stop and park while for green/blue the car is supposed to navigate around the obstacle and return back to the line following the logic of obstacle detection. However, at the end our code dealt with solely color identification without any specific behavior attached to a single color.

Improvements: Firstly, we should include a loop-back from "Change Direction" or "Detect a Clear Path" to retry scanning if the first attempt fails. Next, annotate pre-conditions and constraints more clearly on decision diamonds (for example, label "White Surface Detected?" with sensor tolerance or expected readings).

See Appendix III for previous iterations of the activity diagram.

D. SysML Modeling Use

SysML was employed throughout the design process to ensure a clear, traceable, and modular system structure. It allowed us to visualize, plan, and validate both functional and structural aspects of the vehicle. We made three to four versions of each diagram as the project progressed due to evolving requirements and continuous development. *Please refer to Appendix IV for a comprehensive overview of all iterations, Appendix V for the UPPAAL and Appendix VI for Parametric Diagram.*

1) Use Case Diagram

- This outlines all interactions between the system and the user (e.g., switch on, follow line, park) to visualise the user's perspective on how the car should operate.
- Team A2 interacts with the system by switching it on. The robot then autonomously scans for the line, follows it, halts if necessary and choose a clear path and change direction based on sensor input, then return to following the line.
- It includes detecting obstacles and performing color detection to decide on further actions. It also includes extensions for idling when required, parking after failing to find a clear path, and ultimately turning off once the course is complete.
- Environmental dependencies are highlighted, such as IR sensors scanning for black and white surfaces and ultrasonic or color sensors monitoring the surroundings for obstacles.
- **Improvements:** This diagram is somewhat too detailed, attempting to capture alternative paths and internal behaviors. This can be simplified to focus on the primary interactions between the user and the system for clearer and easier maintainability.

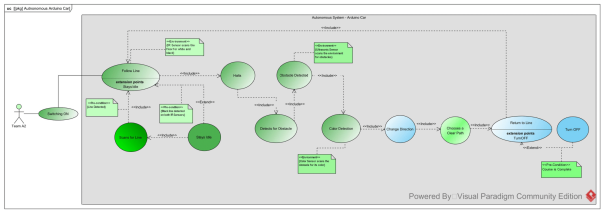


Fig. 4. Use Case Diagram

2) State Machine Diagram

- This models the internal behavioural states and transitions based on input conditions, providing a systematic framework that helps map out the various operational states and their corresponding transitions, which is essential for coding each state's specific functions and behaviours. It follows the same workflow as the activity diagram, but with state specifications.
- It begins in the Initialization state, where sensors and motors are activated.
- Once initialized, the car moves into the state of following a white surface by detecting the line. If an obstacle is detected at a distance less than or equal to 8 cm, or if there's a need to pause, it enters a Wait state, halting the motors.
- During this state, it performs obstacle detection and scans for the obstacle's color, followed by an analysis of the environment to calculate an alternative path.
- **Improvements:** Introduce a dedicated "Fault Handling" or "Sensor Recalibration" state to depict how the system would respond to sensor failures or invalid data. We should also add transitions from states like "Obstacle Detection" directly to an idle or alert state if sensor readings are outside expected ranges, representing error contingencies. We should align the state machine diagram more closely with the actual control logic in the code which make the documentation clearer and easier to maintain. It ensures that design diagrams directly support how the software behaves, which simplifies debugging, validation, and future updates.

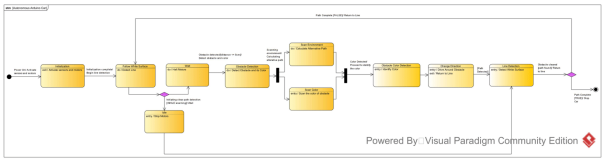


Fig. 5. State Machine Diagram

3) Sequence Diagram

- This illustrates the communication of messages between the control unit and the components.
- The sequence diagram depicts the chronological interaction between different components of the system: the control unit, battery, locomotion system, and sensor system.
- The sequence is triggered when Team A2 presses the power button, prompting the control unit to request power from the battery and initialize both the sensors and motors. The system then enters a continuous loop, fetching and processing sensor data, which results in motor activation signals for line-following.
- If an obstacle is detected, the system sends stop signals to halt the motors, checks the color of the obstacle, and evaluates alternative paths. If a new path is available, it sends adjustment commands to change direction; otherwise, it may initiate a parking sequence.
- The process can be manually terminated by pressing a button to shut down the system, which stops the motors and sensors and disconnects the power.
- **Improvements:** We should incorporate timeout or error signals to handle cases where sensor data is delayed, inconsistent, or missing, improving the system's fault tolerance. The diagram overall can be refined and shortened by removing overly detailed low-level interactions, keeping only the main functional exchanges.

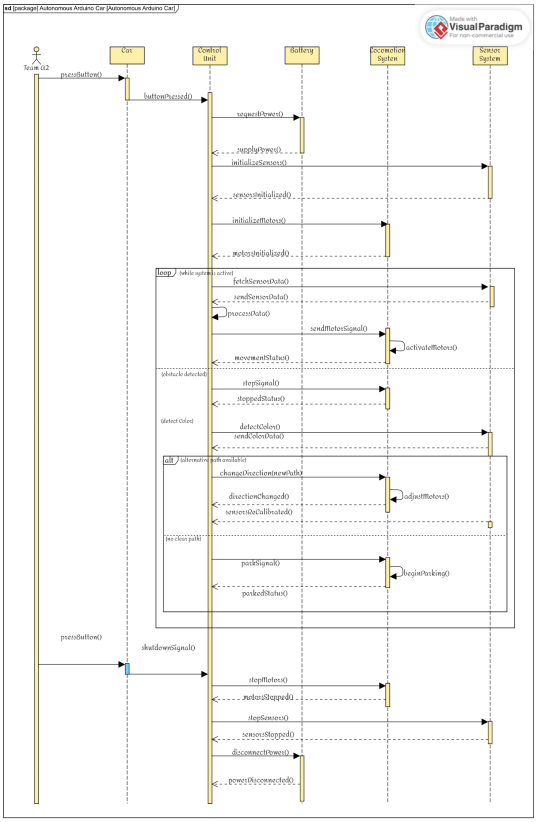


Fig. 6. Sequence Diagram

3) Internal Block Diagram

- The **Internal Block Diagram (IBD)** illustrates the interactions between the major components of the Arduino-based autonomous car, emphasizing both the flow of data signals (e.g., sensor readings, control commands) and the power connections based on the system's voltage specifications.
- The Sensor System includes an ultrasonic sensor, a color sensor, and two IR sensors, all connected by power and signal lines with specified inflow and outflow annotations to signify data directions.
- These feed data to the Control Unit, which processes the inputs and sends control signals to the Motor Driver within the Locomotion System.

- The Motor Driver, powered by the battery, independently drives the right and left motors to enable precise movement.
- The Battery supplies 12V power to the Control Unit, Motor Driver, and sensors, forming a complete, integrated electrical and signal network.

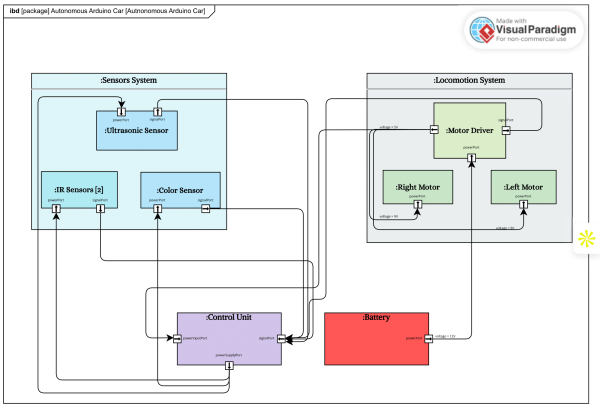


Fig. 7. Internal Block Diagram

4) UPPAAL Diagram

This UPPAAL diagram models the Arduino robot's control logic through key states with explicit time constraints. This is the final version of our UPPAAL diagram, which aligns with our implemented code.

- It starts in a powered-off state, then transitions to line following where it continuously checks the IR sensors, adjusting with `steer_left()` or `steer_right()` if a lean is detected.
- When the ultrasonic sensor measures `distance dist_to_object_limit`, it resets the timer with `t=0` and enters the obstacle detection state.
- After a minimal wait (`t0`), it moves to a scan phase, then waits until `t>6` before performing obstacle avoidance maneuvers (`avoidObstacle()`), resetting the timer again.
- Finally, after another timed wait of `t>10` to complete avoidance checks and turns, it attempts to realign by finding the line, thereby accurately modeling the Arduino delays and sequential logic in `followLine()`, `avoidObstacle()`, and sensor-based decisions.

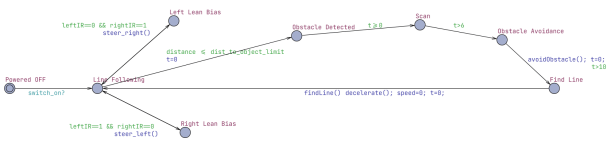


Fig. 8. UPPAAL State Machine

E. Parameterization

The vehicle's design incorporates several constraints, as captured in the Parametric Diagram, to ensure safe and efficient operation. Key operational parameters include:

- **Battery Voltage:** 12V
- **Minimum Obstacle Detection Range:** 15 cm
- **Maximum Line Deviation:** 2 cm (for effective line-following)
- Each motor should get around the **same voltage** to allow for equal speed and operation.
- Light conditions can impact the accuracy of the color sensor. These variables were accounted for through parameter tuning in the hardware (e.g., IR sensor placement).

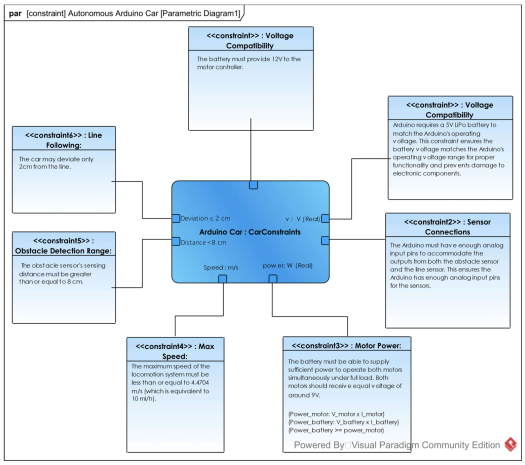


Fig. 9. Parametric Diagram

III. Prototyping and Design

This section will look into the electronic components used in the car, the visual prototypes and physical assembly of each and every component to allow functioning as a single unit.

A. Electronic Components

1) Microcontroller

- **Arduino Uno R4 Wifi**:
  - Acts as the control unit.
  - 14 digital I/O pins (used for sensors/motors), 6 analog inputs.
  - Handles PWM for motor speed control and digital logic for direction.

2) Sensors

- **IR Sensors (2x)**: Line detection by sensing black/white contrast.
- **Ultrasonic Sensor**: Detects obstacles in front of the car (15 cm).
- **Color Sensor**: colors and decides the behavior triggered based on result.

3) Motor Driver

- L298N:
  - Directly receives the entire power supply from the battery through direct connection.
  - Provides power to and receives signals from Arduino.
  - Controls 2 DC motors (direction + speed).
  - 2 inputs per motor (IN1/IN2 for right, IN3/IN4 for left) + ENA/ENB for PWM speed control.

4) Motors

- **DC Gear Motors (2x)**: Drive wheels independently.
  - Controlled via motor driver.

5) Power Supply

- **12V LiPo Battery**: Powers both Arduino and motors.
  - Ensure correct voltage regulation for Arduino (5V via Vin through the motor driver).
  - Must supply enough current to drive both motors simultaneously.

B. Virtual and Physical Properties

We relied on 3D printing to create three-dimensional objects from digital models designed in AutoDesk Fusion [3]. This technology enables precise part production that is crucial for high-level prototyping and manufacturing, but since the process is time-consuming and material-intensive, we had to carefully consider the size and time taken by each component before printing. We primarily used this method to create holders for sensors and actuators, as these components required greater precision and structural integrity.

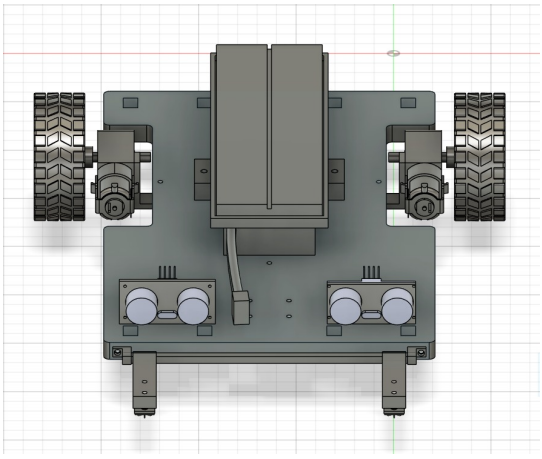


Fig. 10. Virtual Assembly

1) Breadboard and Battery Holder

Since we had already finalized the placement of components on the chassis, we designed the breadboard and battery holder to fit securely on top of each other. The breadboard is enclosed within a cavity in the holder, while the battery is placed underneath the back of the cavity and secured using screws on the protruding sides of the holder.

For the second version, we kept the fundamentals intact and just increased the thickness of the protruding sides to ensure the sides do not crack under the tension and pressure of screwing the screws.

Improvements:

There are three protruding sides on our holder; however, we utilized only the ones on the left and right. The front protrusion was facing inward instead of outward, making it impossible to screw into the chassis. To improve this, we would plan to redesign the front extension by elongating it to allow proper securing to the chassis.

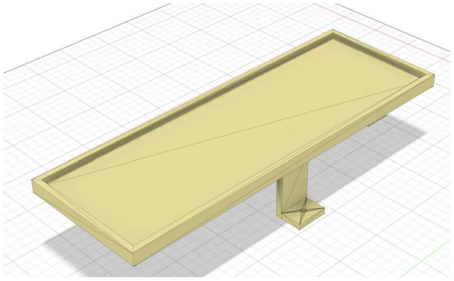


Fig. 11. 3D Modelled Breadboard and Battery Holder V1

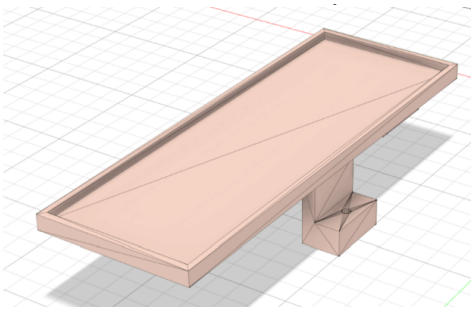


Fig. 12. 3D Modelled Breadboard and Battery Holder V2

2) IR Sensors Holder

The IR sensor holder was designed by imagining the IR Sensors mounted on a railing, and working like a curtain. The two rectangular bricks had the IR Sensors mounted on it through screw holes aligning directly with the holes in the IR Sensors. Those two bricks would then have a long rod inserted collectively through both of them using the square hole on the front of the bricks. This allowed us to vary the position and distance between the IR Sensors based on our requirements. Then, that rod with the bricks is connected through the holes in the screw holders which are then screwed into the front side of the chassis.

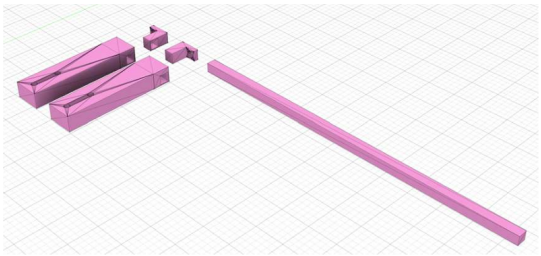


Fig. 13. 3D Modelled IR Sensors Holder V1

For the second version, we centered the square holes on the bricks because the holes in the first version were not properly printed which rendered our design unusable as the railing would not be secured inside them. Lastly, we made the screw holders thicker for stability.

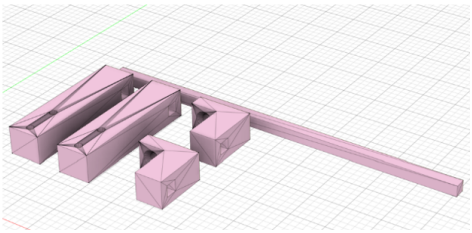


Fig. 14. 3D Modelled IR Sensors Holder V2

3) Motor Holders

We designed 3D printed motor holders to ensure a snug fit for the motors. The holders included covers on both the front and back to prevent the motors from slipping during sharp turns.

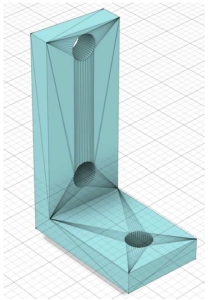


Fig. 15. 3D Modelled Motor Holders V1

For our second version, we added champers to our screw holes to better accommodate the screws to avoid them protruding outwards. We did not, however, use these in our final assembly because our car tilted heavily towards the front which hindered its ability to drive properly.

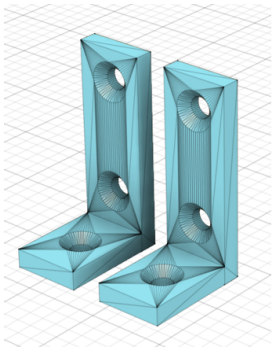


Fig. 16. 3D Modelled Motor Holders V2

4) Ultrasonic Holders

We designed a simple ultrasonic sensor holder to be mounted on the front of the chassis by referencing the provided 3D ultrasonic sensor file and creating a thicker base for enhanced rigidity and stability. The holes for the ultrasonic sensor's speakers were designed to match the exact diameter specified in the 3D file, with minor tolerances incorporated to ensure a secure, fixed attachment.

We did not, however, employ these in our final assembly because we had assumed the usage of two ultrasonic sensors, and not the servo motor. We, instead, used the standard ones available in the lab to mount the sensor on top of the servo motor.

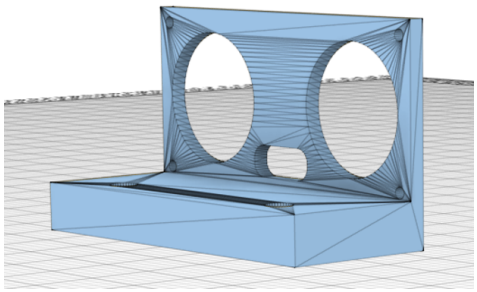


Fig. 17. 3D Modelled Ultrasonic Holder

C. Hardware Assembly

The hardware assembly of the autonomous Arduino car involved mounting all electronic components onto a laser-cut wooden chassis. A central breadboard was positioned across the bridge between the two extended sides to optimize space and wiring management. The Arduino Uno and L298N motor driver module were situated at the front-left and front-right corners, respectively, allowing efficient wiring routing. For obstacle detection, an ultrasonic sensor mounted on a servo motor was fixed at the front edge of the chassis to provide a wide scanning range which was combined with a color sensor to become a single component using a 3D modelled holder. The two DC motors were secured to the sides of the bridge using motor holders, enabling stable locomotion and balanced weight distribution.

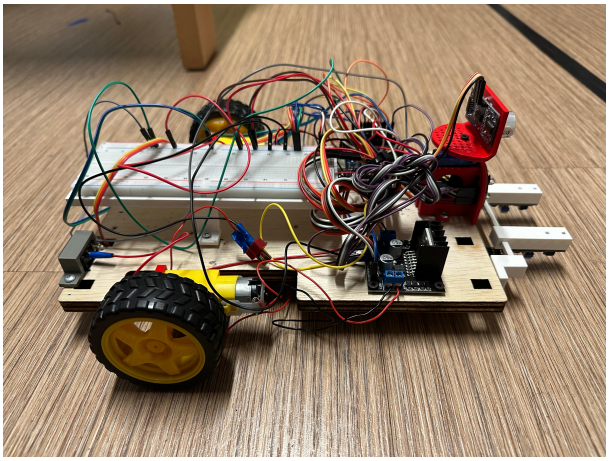


Fig. 18. Assembled Arduino Car (side view)

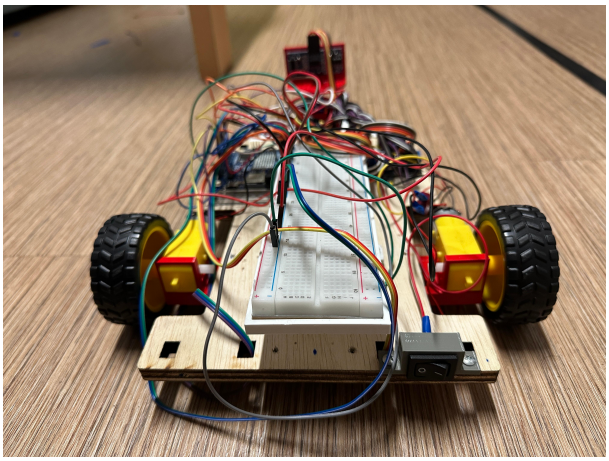


Fig. 19. Assembled Arduino Car (back view)

IV. Circuit Design

This figure illustrates the wiring schematic of our circuit, which served as the basis for our final hardware assembly. A detailed explanation of the wiring connections is provided below. *Refer to Appendix VII for earlier schematic iterations and alternative layouts considered during development.*

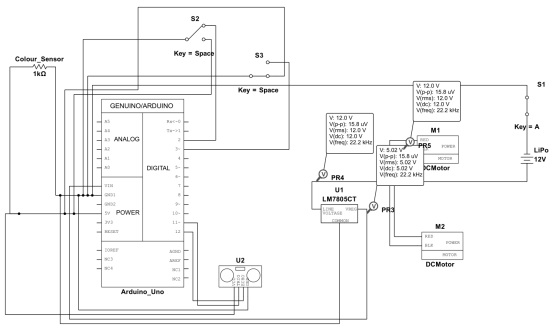


Fig. 20. Circuit Design

A. Motors (Connected via Motor Driver):

- Right Motor:
  - Speed Control (ENA) = **Pin 10**
  - Direction Control = **IN1: Pin 4, IN2: Pin 5**
- Left Motor:
  - Speed Control (ENB) = **Pin 11**
  - Direction Control = **IN3: Pin 6, IN4: Pin 7**

B. Infrared (IR) Line Sensors:

- Left IR Sensor = **Pin 12**
- Right IR Sensor = **Pin 13**

C. Ultrasonic Sensor (HC-SR04):

- Trigger Pin (TRIG) = **Pin 9**
- Echo Pin (ECHO) = **Pin 8**

D. Color Sensor (TCS3200):

- Frequency Scaling Pins = **S0: Pin 2, S1: Pin 3**
- Color Filter Selection Pins = **S2: A0, S3: A1**
- Output Signal (OUT) = **A2**
- Output Enable (OE) = **A3** (Set LOW to enable sensor)
- Onboard LED Control = **A4** (Set HIGH to turn on)

V. Software Implementation

A. Initialization of Sensors and Actuators

The `setup()` function in the Arduino sketch is responsible for initializing all sensors, actuators, and peripheral components. This includes setting pin modes for input/output functionality, starting serial communication for debugging, and configuring the color sensor for operation. Below is a code snippet demonstrating the initialization logic:

```
void setup() {
  Serial.begin(9600);

  // Initialize Motor Driver pins
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENB, OUTPUT);

  // Initialize IR sensors
  pinMode(IR_LEFT, INPUT);
  pinMode(IR_RIGHT, INPUT);

  // Initialize Ultrasonic Sensor
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // Initialize Color Sensor (TCS3200)
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(OE, OUTPUT);
  pinMode(LED, OUTPUT);
  pinMode(OUT, INPUT);

  // Configure Color Sensor frequency scaling and enable it
  digitalWrite(S0, HIGH);
  digitalWrite(S1, HIGH);
  digitalWrite(OE, LOW);    // Enable sensor output
  digitalWrite(LED, HIGH);  // Turn on sensor LEDs

  stopMotors(); // Ensure motors are stopped at startup
}
```

Fig. 21. Motor and Sensors Initialization Code

B. Movement Control Algorithms

The vehicle's movement is controlled via a set of modular motor functions. These functions allow the car to move forward, stop, or perform turns by independently controlling each DC motor through an H-bridge motor driver. Pulse-width modulation (PWM) is used on the speed control pins (ENA and ENB) to regulate motor speed.

Key movement patterns implemented:

- **moveForward():** Both motors run in the forward direction.
- **runLeftMotor() / runRightMotor():** Enables movement of one motor for turning behavior.
- **stopMotors():** Immediately halts all motor activity.
- **runLeftMotorReverse() / runRightMotorReverse():** Enables backward motion used during obstacle avoidance routines.

C. Behavioural Decision Logic (Code Centric)

The vehicle's high-level logic is implemented inside the `loop()` function. The logic combines line-following, obstacle detection using ultrasonic distance measurement. This logic has also been illustrated in the activity diagram (Refer to Fig 3).

- **Line Following Logic:**

It starts with an initialization of its sensors and motors, then transitions into line following function: the IR sensors function by maintaining a set distance apart, with both sensors initially detecting the white surface to maintain straight-line movement while both sensors read **LOW**. When a single IR sensor encounters a black line that sensor reads **HIGH**, the robot immediately turns in the direction of that sensor to follow the line's path. When both sensors simultaneously detect the black line, both reads **HIGH**, the robot stops completely, creating a simple yet effective line-following system that uses differential sensor feedback for navigation control.

- **Obstacle Detection:**

If the ultrasonic sensor detects an obstacle within 15 cm, the car stops and enters the detection and avoidance routine.

- **Avoidance Logic:**

If an obstacle is found, the robot scans both left and right directions to find a clear path using a sequence of turning and distance-checking motions. If neither side is clear, the robot halts.

Refer to Appendix IX for complete code.

VI. Git Usage and Collaboration

The project development process was meticulously managed using GitHub, enabling version control, collaboration, and progress tracking. A branching strategy was adopted to separate different tasks such as wire testing, SysML diagram development, and component testing.

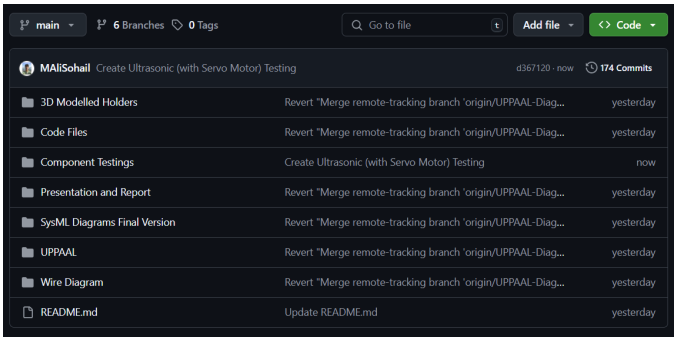


Fig. 22. The Main Branch (pre-merge)

The GitHub repository includes the main directory and branches like Component-Testing, SysML-Diagrams, Wire-Testing, Version-3, and Line-Following-Code-Versions, each reflecting different stages and focus areas of the development process.

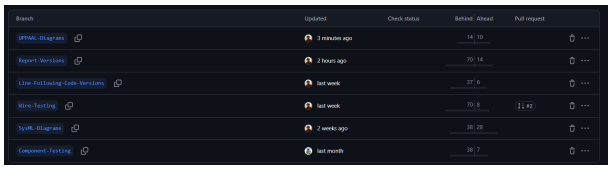


Fig. 23. Branches

Each branch included multiple versions of their respective folders in the main branch which were merged at the end when we had uploaded our final versions [4]. *Refer to Appendix VIII for detailed breakdown of each branch.*

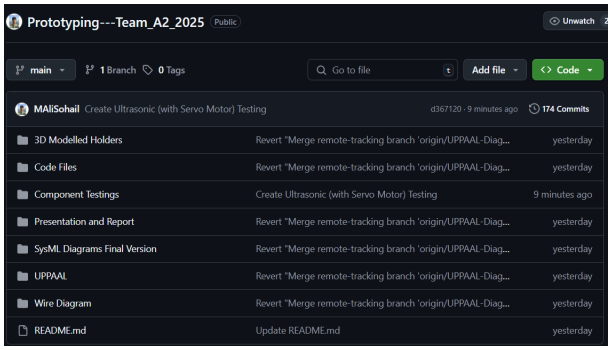


Fig. 24. Final Main Branch (post-merge)

A. Improvements

1) Lack of Pull Requests & Reviews

- Most branches have **no pull requests (PRs)**, and no merge documentation.

2) Branch Overlap and Inconsistency

- Ahead/behind statistics suggest merges have not occurred properly or frequently, risking merge conflicts.

3) No Issue Tracking or Milestones

- No use of **GitHub Issues**, **To-do list**, **Milestones**, or **Projects** to plan tasks or track bugs/features.

4) Commit Messages

- Our commit messages such as *"Add files via upload"* or *"Rename Arduino Code..."* are vague. This weakens long-term traceability.

B. Possible Future Implementation

For improvements we can use the following structure to help us document better and do version control for our github which includes: introducing branches like **develop** (merge all feature branches for final testing before promoting to main), **feature/obstacle-logic**, and **hotfix/ir-sensor-failure**. This approach ensures stable code in **main** (only store final, tested versions of our code and other tasks), organized feature development, and fast bug fixes without disrupting ongoing work [5].

While our current repository merges all work directly into the main branch following multiple merges, a more robust branching strategy could be implemented for future iterations or similar projects. This would involve using dedicated branches, in this example for our line following code, such as:

- **feature/line-following**: to develop and test new line-following algorithms (e.g., PID tuning or state-based steering).
- **develop**: to integrate multiple features before promoting stable versions to `main`.
- **hotfix/ir-sensor-failure**: to isolate and quickly patch sensor-specific issues.

C. Contribution Breakdown:

TABLE I
GIT COMMITS PER TEAMMATE

Team Member	Number of Commits
Moeez Mufti	138
Muhammad Ali	34
Total	174

- **Moeez Mufti (138 commits)**: Was responsible for GitHub repository management, SysML modeling, version control, wire-testing and diagrams and initial component testing. He was also responsible for detailed diagram creation and software-hardware integration.
- **Muhammad Ali (34 commits)**: Designed 3D-printed holders for the IR sensors, motors, and ultrasonic sensor using AutoCAD, ensuring accurate fit and positioning. He took part in early assembly by mounting components onto the breadboard and battery holder and verifying sensor alignment. Muhammad developed both the initial and final Arduino code, handling motor control, sensor integration, and obstacle avoidance. He also helped wire the system, ensuring reliable connections between sensors, the motor driver, and Arduino, and contributed to testing and debugging sensor performance during validation.

VII. CHALLENGES AND IMPROVEMENTS

Throughout the design and implementation of this project, we encountered the following technical-related challenges:

- **Absence of Colour Logic:**

The TCS3200 colour sensor was not properly integrated into our final build during development due to limited time for proper calibration. If we were to revisit the project, we would take a more gradual approach: focusing on properly understanding the sensor, testing it early on, and using available guides or tutorials to help with calibration and integration. This would allow us to successfully implement colour-based actions in future versions of the car.

- **Line Following Stability:**

Maintaining a consistent line-following path proved difficult due to slight sensor misalignments and the sensitivity of the IR sensors, which made the system prone to steering off track, especially around curves. One of the IR sensors had to be replaced due to a manufacturing defect, wherein the infrared LEDs were positioned too far inward, adversely affecting the sensor's ability to detect reflected signals.

Improvements: Introduce a proportional–integral–derivative (PID) control algorithm to dynamically adjust motor speed based on IR sensor readings, improving responsiveness around curves.

- **Motor Speed Control:**

The DC motors had to be carefully tuned using PWM to achieve sufficiently low speeds for precise navigation. Even at lower speeds, minor differences in motor characteristics sometimes caused uneven turns. The parts given were not of identical specifications, which leads to one motor receiving more voltage than the other. Upon experimentation with individual power supplies for each motor, it was confirmed that both motors operated at equivalent voltage levels when isolated, indicating that the imbalance arose from the combined circuit configuration.

Improvements: Calibrate PWM outputs individually for each motor to compensate for small voltage/current differences, or add inline resistors or motor driver tuning to balance power.

- **Ultrasonic Sensor Integration:**

Ensuring reliable obstacle detection within the desired minimum range (15 cm) proved challenging, as our code could not be effectively aligned to accommodate obstacle avoidance. Thus, this functionality did not work as intended in the final implementation.

Improvements: Firstly, conduct systematic threshold tests under controlled distances to determine optimal trigger points. Secondly, use software averaging (take 3-5 readings and average) to reduce false positives caused by noise and add time-based failsafes.

- **3D Printing Precision:**

Initial designs for the IR sensor and motor holders faced issues like thin or misaligned screw holes that cracked under

stress. These had to be redesigned with increased thickness and adjusted hole positions.

Improvements: Refer to the *Virtual and Physical Properties* section for details on iterative adjustments.

- **Version Control Practices:**

While GitHub was used to manage code and design files, the lack of formal pull requests, detailed commit messages, and consistent merging practices made it harder to track incremental progress and integrate different features. This ultimately led to inconsistencies and an incorrect method in version control.

Improvements: Refer to the *GitHub Usage and Collaboration* section for recommendations on structured workflows and branch management.

VIII. Final Presentation and Conclusion

During the final presentation of our car, unfortunately, due to the challenges faced as explained in this report, our car did function as intended: owing to the speed imbalance and the technical difficulties in the motors, our car turned out to be quite fast which rendered it difficult to navigate through sharp turns and as a result it would venture off from the path given, and even though we had modules our motors to operate at the lowest we could make it to, it was still fast enough. However, within a controlled environment, the car was able to perform as expected. Lastly, due to issues pertaining to the code, our ultrasonic was not able to detect the obstacles and would crash into them.

This report has detailed the comprehensive development of an autonomous Arduino-based vehicle that should be capable of line following, obstacle detection, and color-based response. The project combined mechanical 3D-printed components, pre-selected electronic modules, and modular software to produce a functional prototype. The use of SysML modeling throughout ensured a disciplined engineering process, guiding the design from clear requirements through to behavioral and structural diagrams. Furthermore, the creation of UPPAAL models validated the timing and logical sequences of the system prior to hardware deployment.

Collectively, these approaches ensured consistency between the conceptual models and the implemented system. The experimental outcomes demonstrated that the prototype could navigate a predefined path, respond to detected obstacles, and execute basic avoidance maneuvers. However, challenges in precise line following, uneven motor characteristics, and incomplete color sensor calibration highlighted areas for refinement. Looking ahead, future iterations envision integrating encoders for real-time speed correction, or implementing PID control to significantly enhance path stability. Additional improvements include refining the obstacle detection by calibrating ultrasonic thresholds under controlled conditions and employing averaging techniques to minimize false readings. These steps would ultimately enable the development of a vehicle that operates as initially envisioned, achieving the level of functionality and autonomy that was originally planned.

In conclusion, this project gave us an insight towards how an engineering project is taken from conception to fruition and how it evolves over a given period time with evolving requirements, constant deadline and and the essential role of iterative experimentation in managing variable outcomes throughout the development process. Despite facing challenges that posed significant obstacles and some that were unable to triumph over, the forward-looking steps outlined earlier demonstrate both an acknowledgment of the current system's limitations and a clear strategy to advance it into a more robust, responsive, and intelligent autonomous platform.

IX. Key Achievements

- Successfully 3D modelled a modular IR sensor mount
- Designed and implemented a complete SysML model (requirements, structural, behavioral, and parametric diagrams) to ensure traceability from concept to realization.
- Completed all tasks assigned on a weekly basis which helped keep the project on track and ensured that deadlines were met.

Acknowledgment

We would like to express our sincere gratitude to our instructors at Hochschule Hamm-Lippstadt for their invaluable guidance and support throughout the development of this project. Their feedback and encouragement played a crucial role in helping us navigate the challenges of system integration, design, and prototyping. This project would not have been possible without the combined knowledge, creativity, and teamwork of everyone involved.

References

[1] M. N. R. Bhukya, R. M. Sanapathi, V. S. V. Koti, and D. S. V. Sravya, "Self Driving AI Robot Car Using Arduino Uno," *International Journal of Engineering Research & Technology (IJERT)*, vol. 12, no. 5, pp. 1210–1214, May 2023.

[2] Visual Paradigm International, "Visual Paradigm," 2024. [Online]. Available: https://www.visual-paradigm.com/ [3] Autodesk Inc., "Autodesk Fusion 360," Autodesk, 2024. [Online]. Available: https://www.autodesk.com/products/fusion-360/overview

[4] MAliSohail, "GitHub - MAliSohail/Prototyping—Team$_A$2$_2$025," $GitHub, 2025. https://github.com/MAliSohail/Prototyping---Team_A2_2025$.

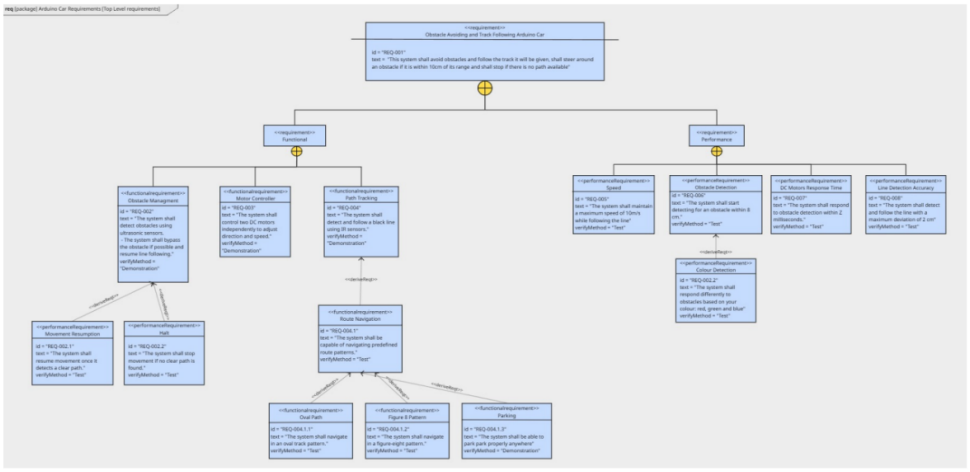
[5] V. Driessen, "A successful Git branching model," nvie.com, 2010. https://nvie.com/posts/a-successful-git branching-model/

Declaration of Originality

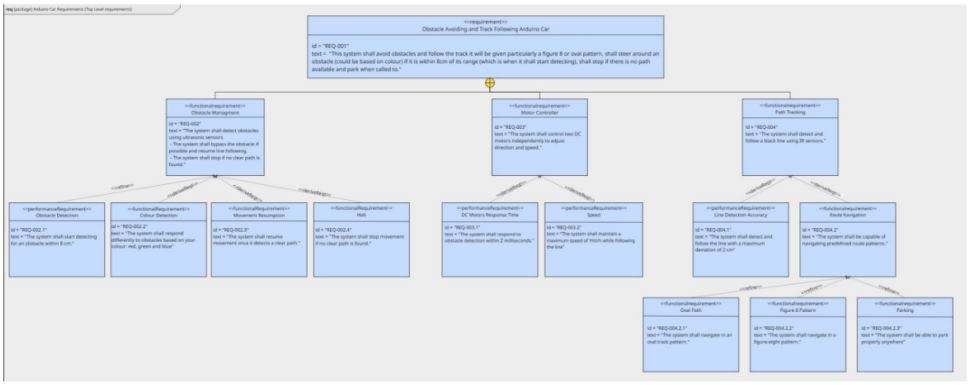
We hereby declare that this report is our own work and that we have acknowledged all sources used.

APPENDICES

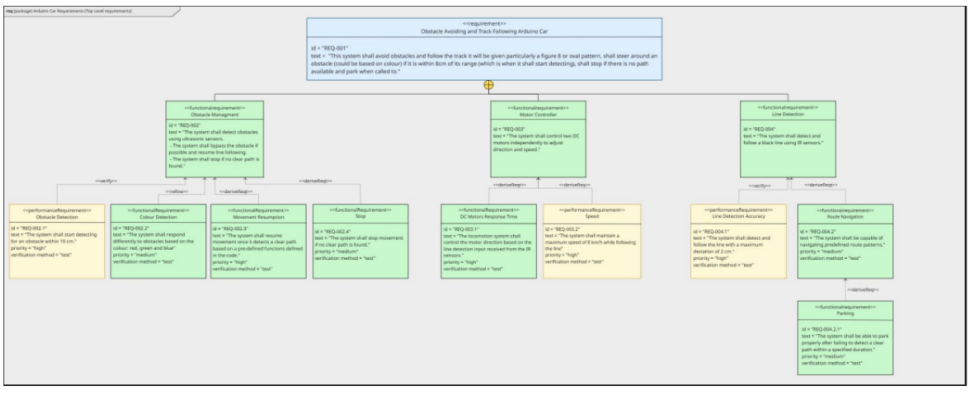
Appendix I: SysML Diagrams Iteration (Requirements Diagram)



Version 1

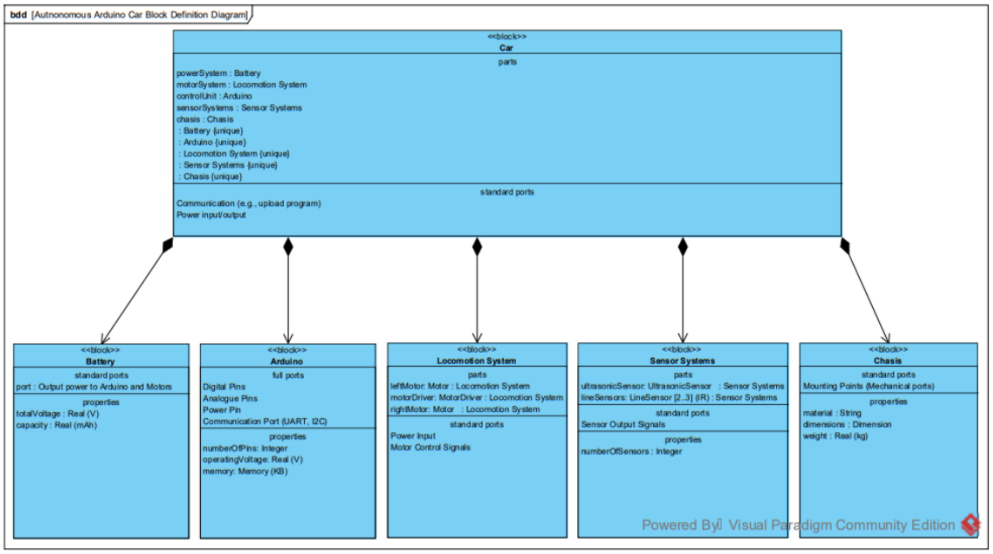


Version 2

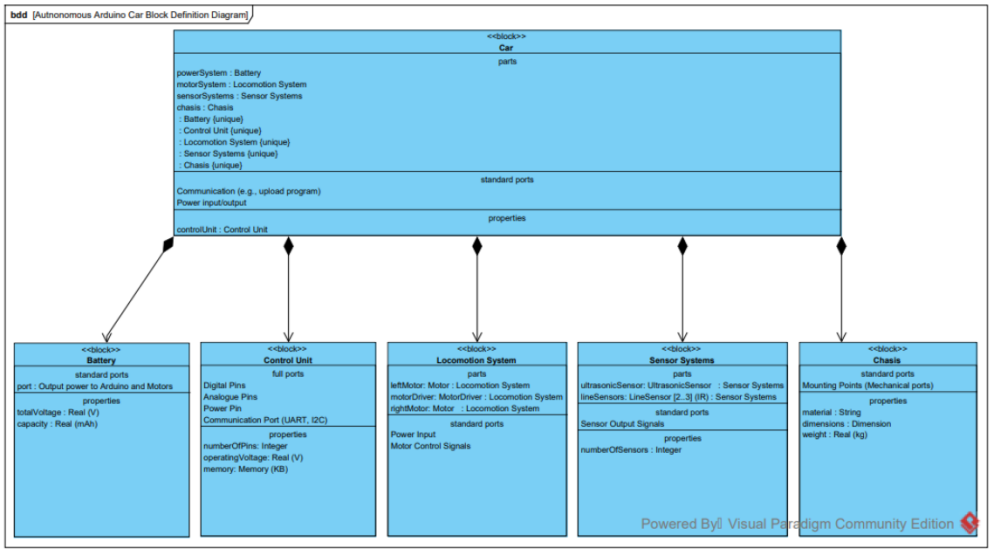


Version 3

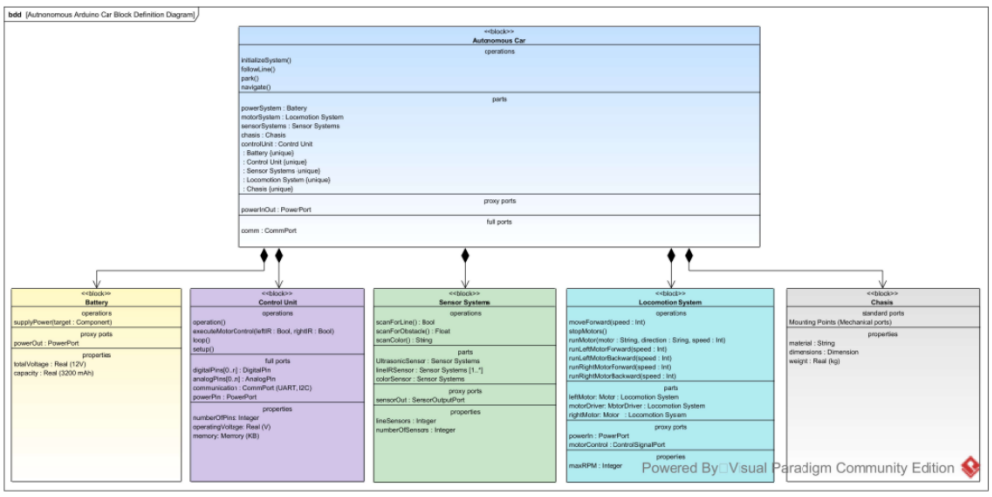
Appendix II: SysML Diagrams Iteration (BDD)



Version 1

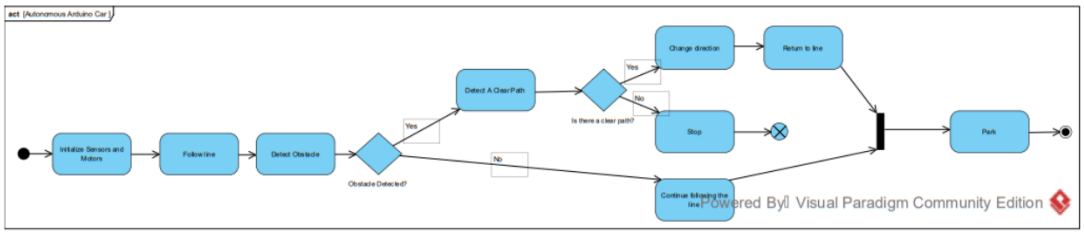


Version 2

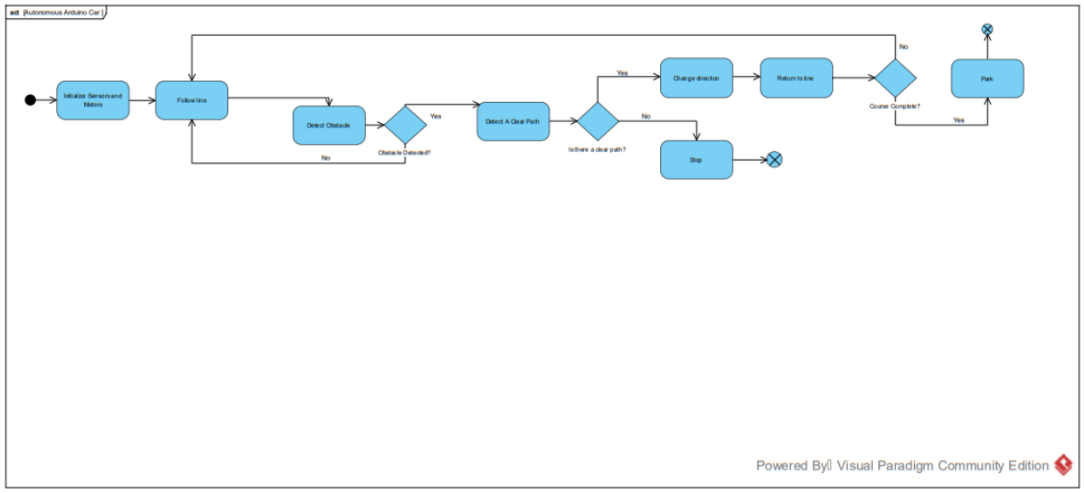


Version 3

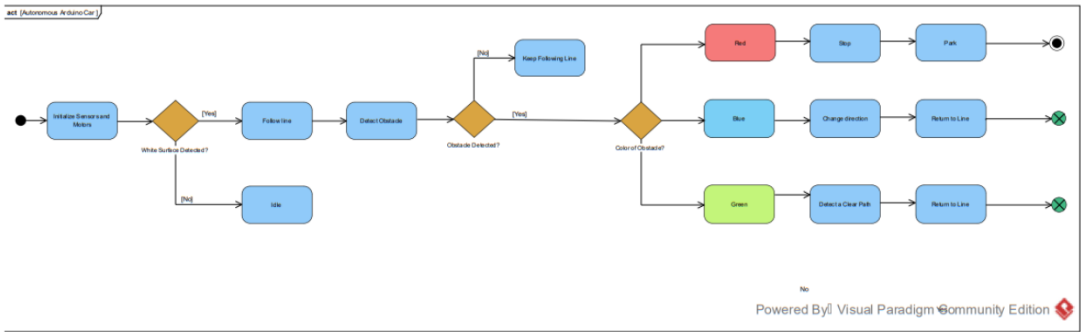
Appendix III: SysML Diagrams Iteration (Behavioral Diagram - Activity)



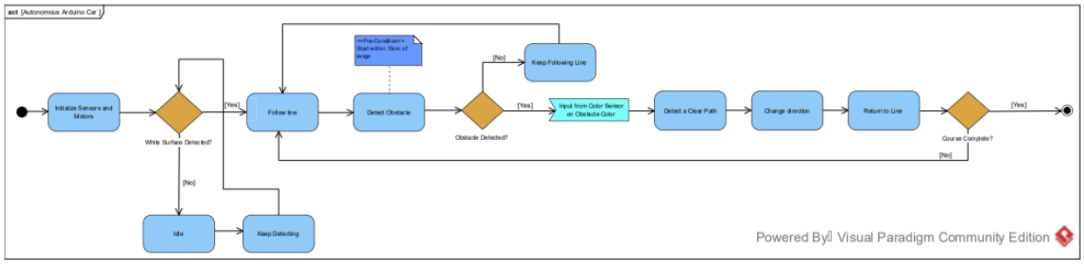
Version 1



Version 2

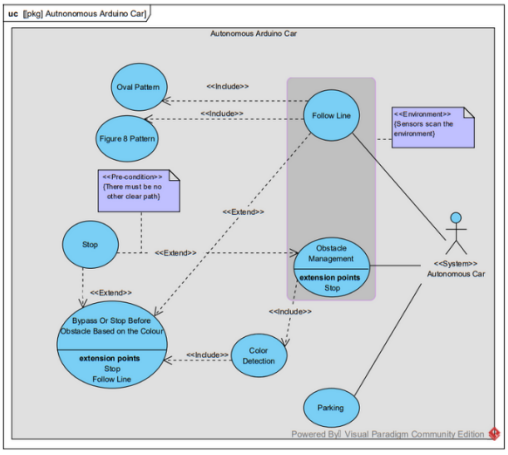


Version 3

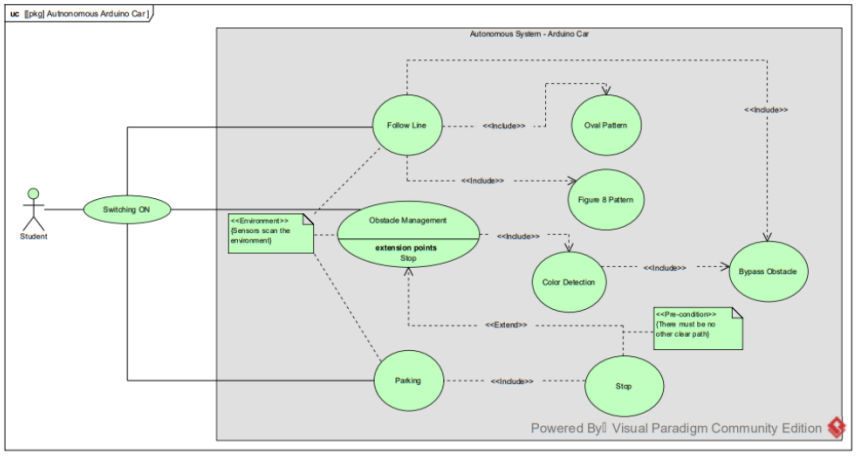


Version 4

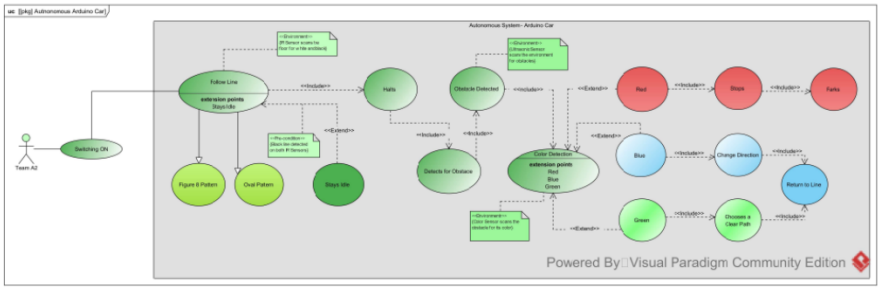
Appendix IV: SysML Diagrams Iteration (SysML Modeling Use Diagrams)



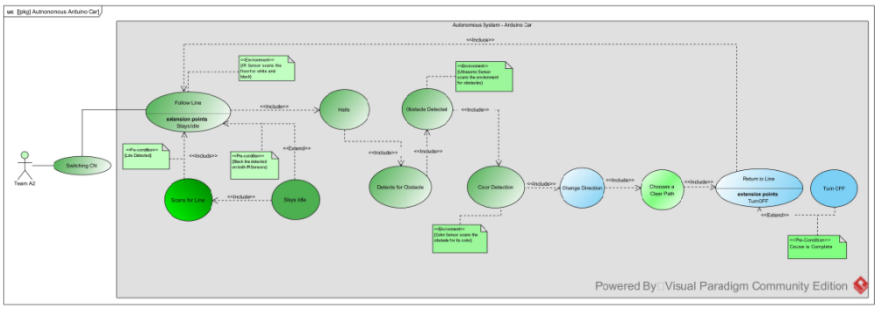
Version 1



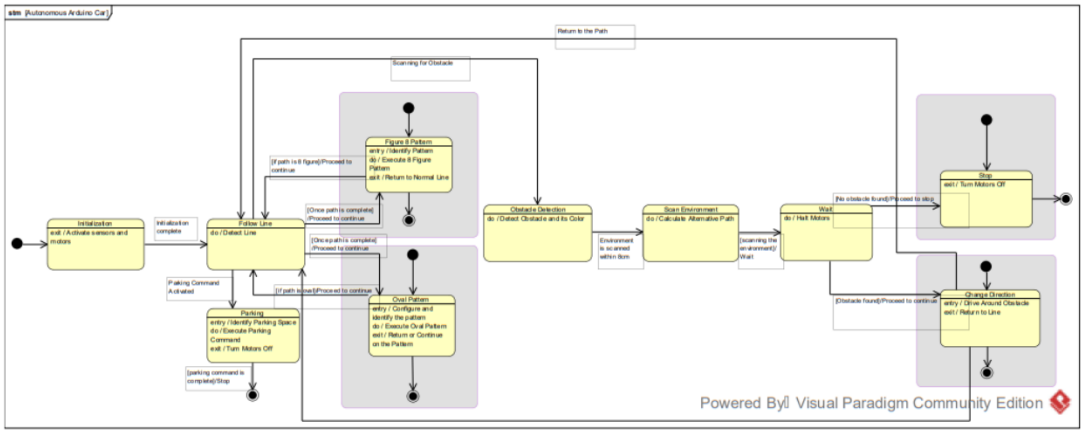
Version 2



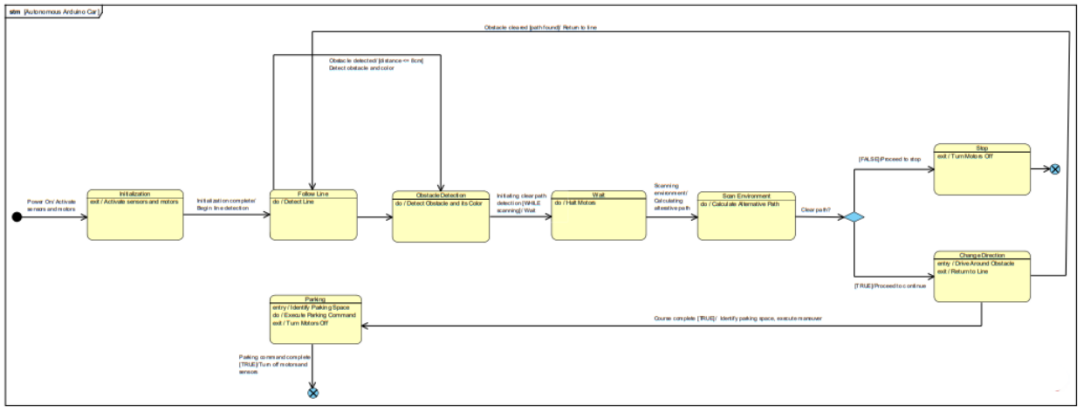
Version 3



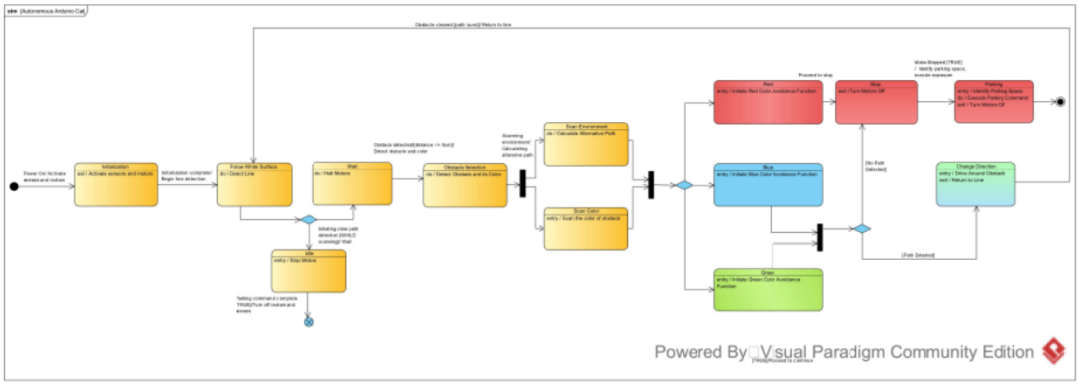
Version 4



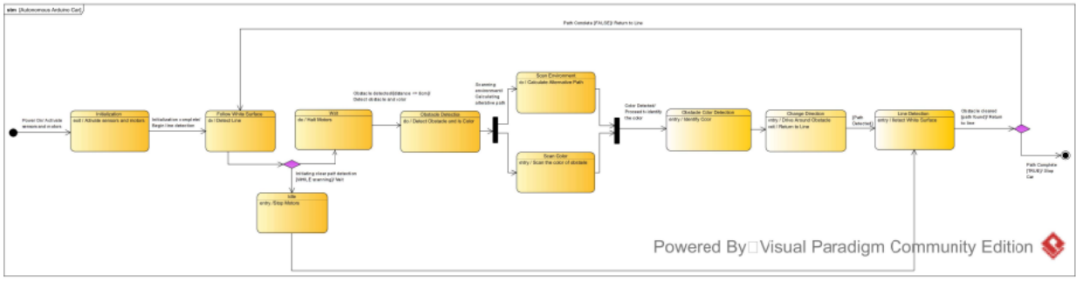
Version 1



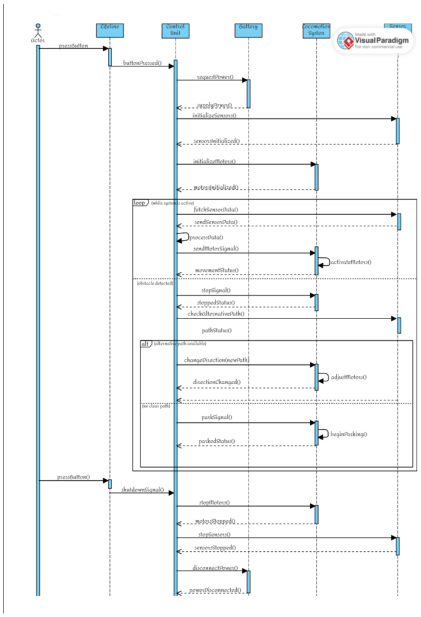
Version 2



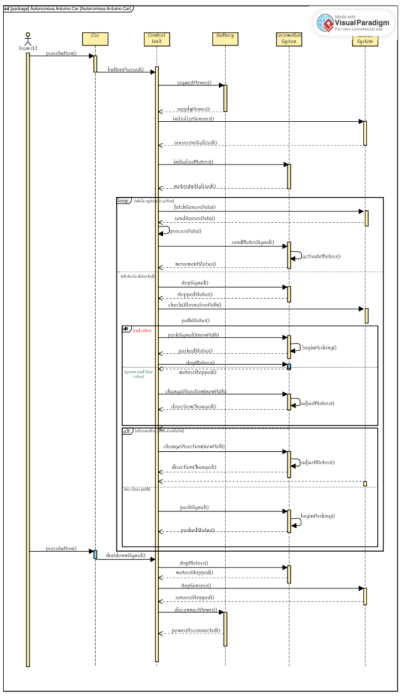
Version 3



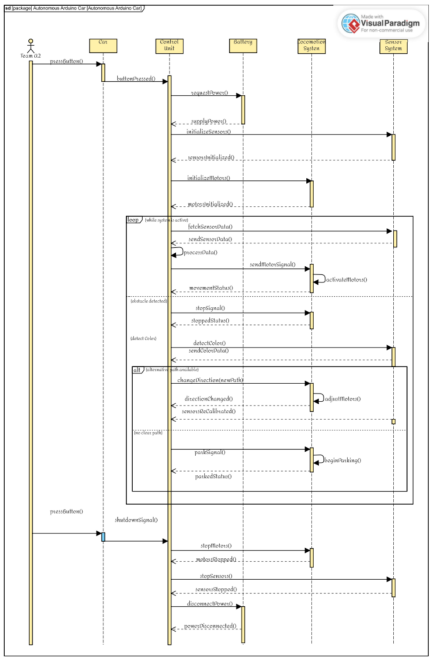
Version 4



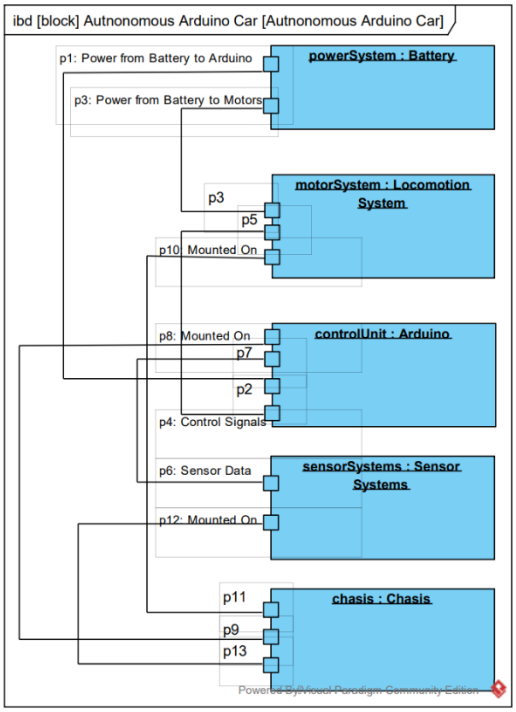
Version 1



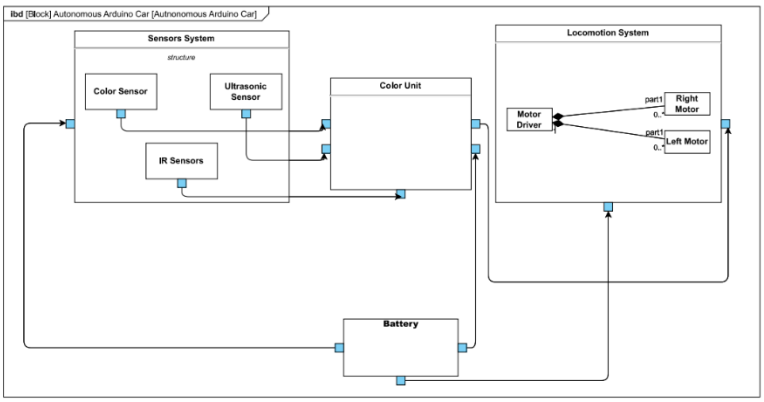
Version 2



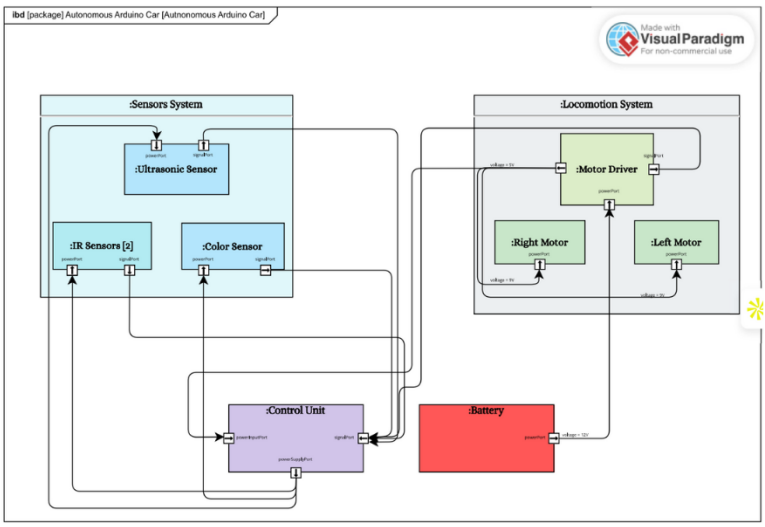
Version 3



Version 1

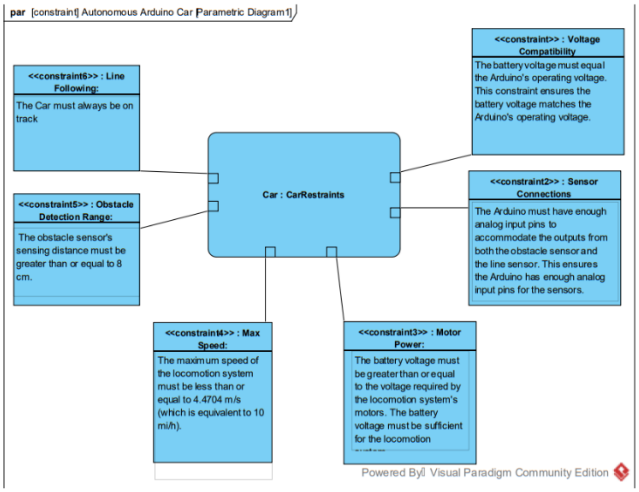


Version 2

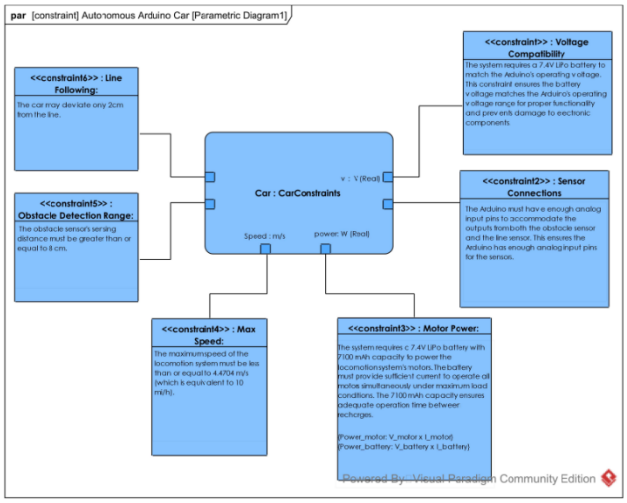


Version 3

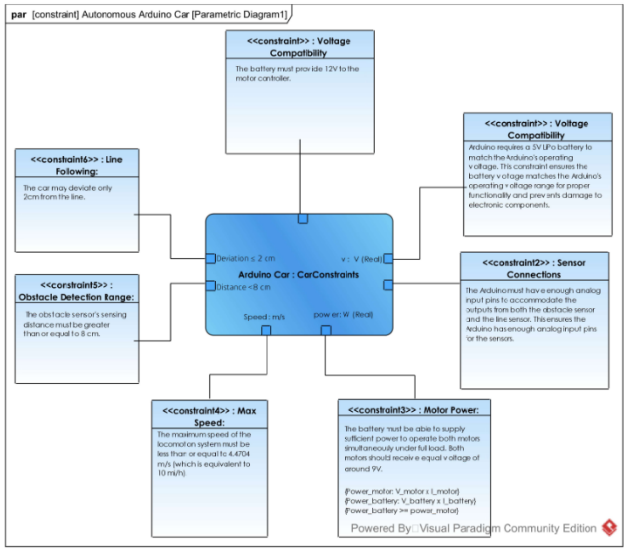
Appendix V: SysML Diagrams Iteration (Parametric Diagrams)



Version 1

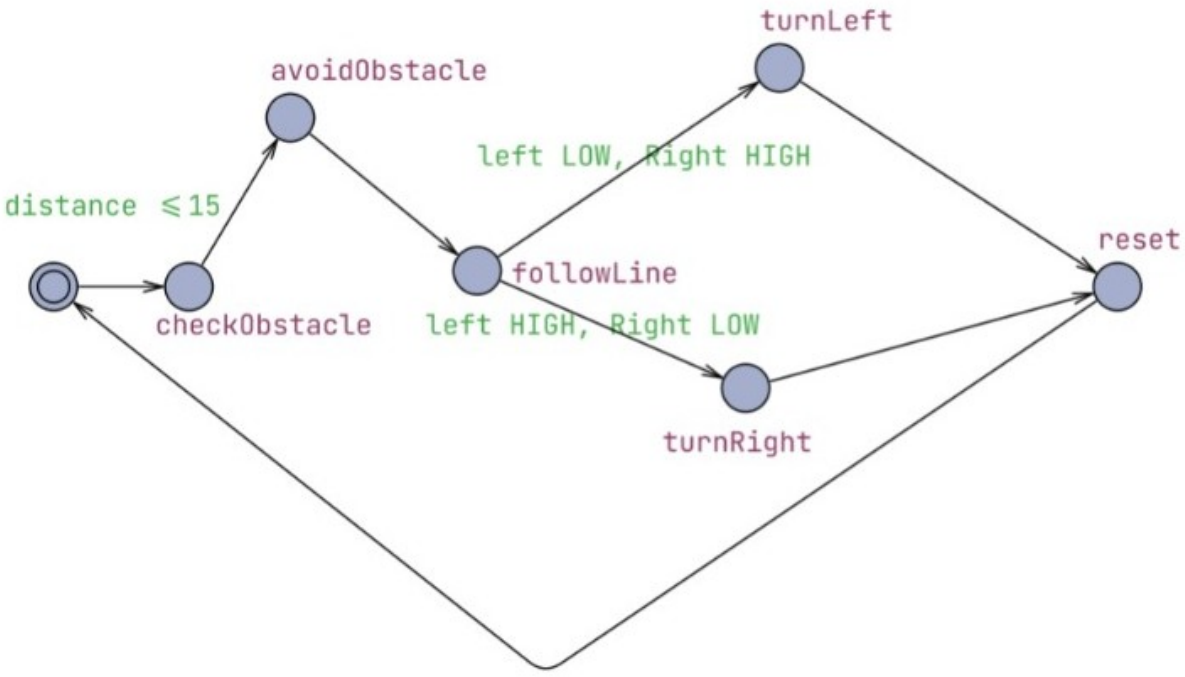


Version 2



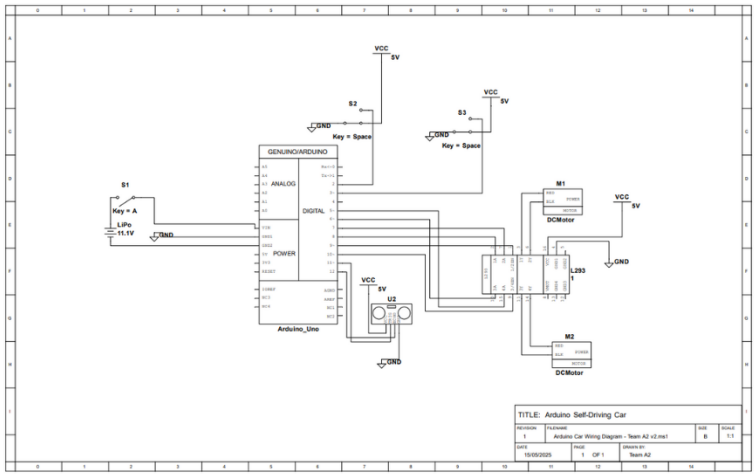
Version 3

Appendix VI: UPPAAL Diagram (all iterations)

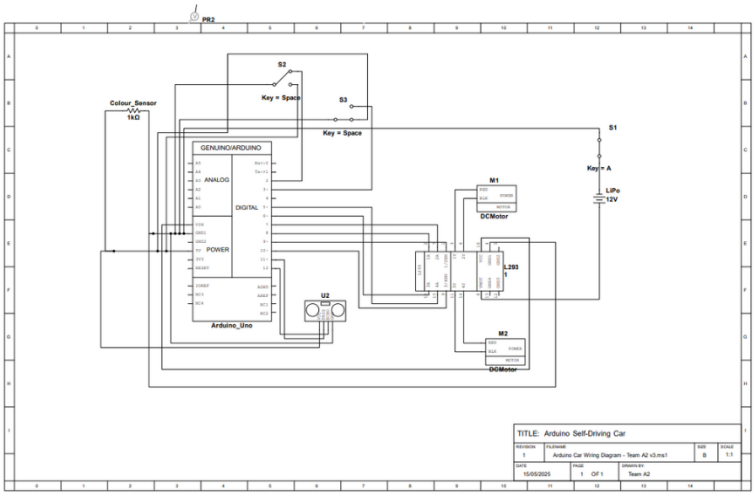


Version 1

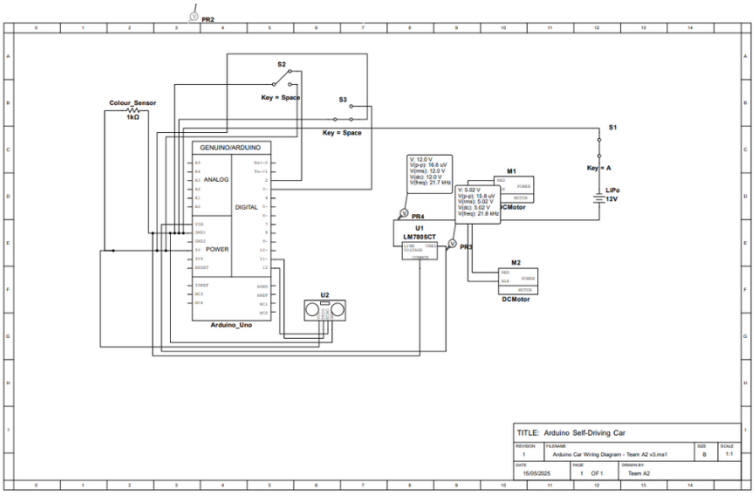
Appendix VII: Wiring Diagram



Version 1

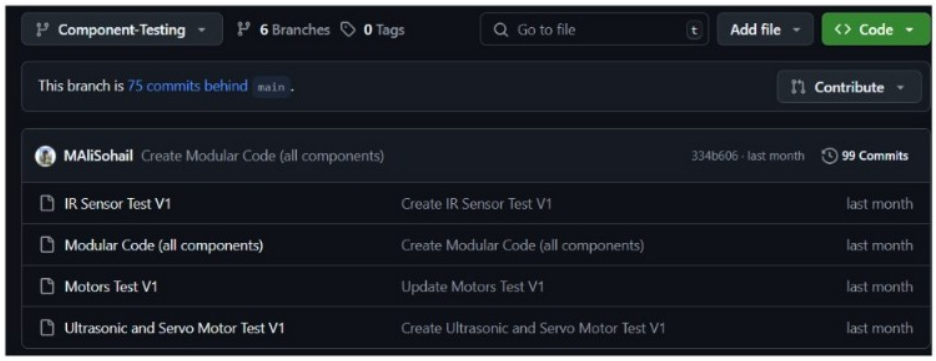


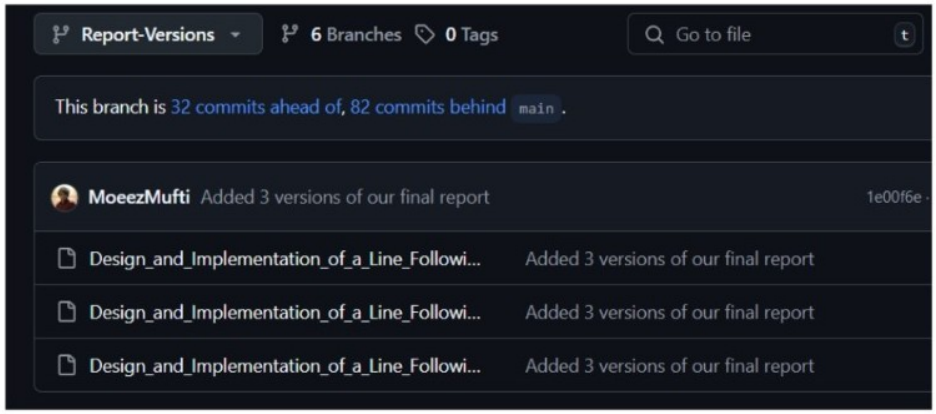
Version 2

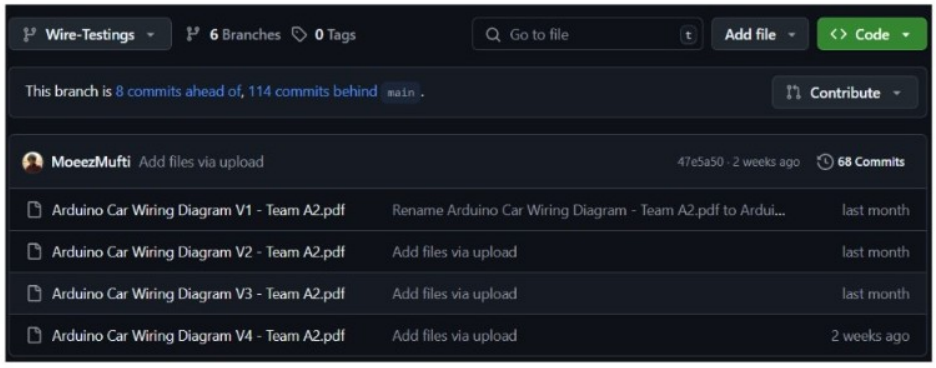


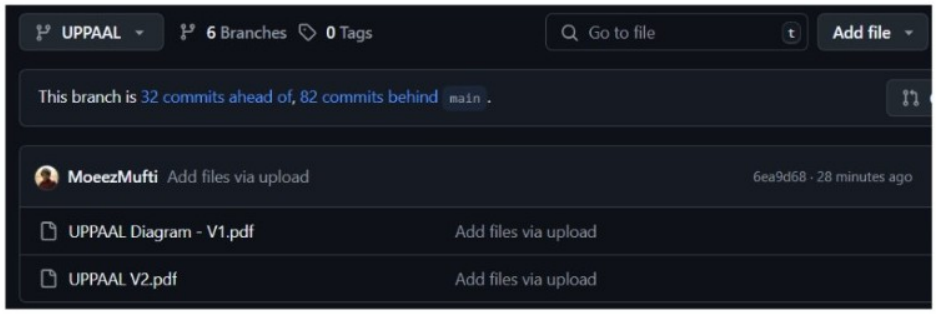
Version 3

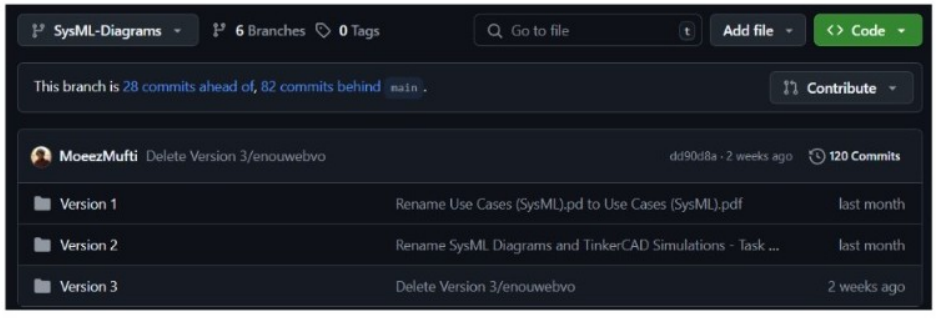
Appendix VIII: Github Branch Breakdown











Appendix IX: Final Arduino Code

This page is intentionally added at the end to include the final Arduino Code we used.

Remarks: It is attached at the end of this document.

```
/****************************************
 * Line Following + Obstacle Avoidance +
 * Color Detection
 * Group A2 - Prototyping
 * SS25 HSHL
 ****************************************/

// Pin Definitions

// Motor Definitions
#define ENA 10   // Right motor speed
#define IN1 4    // Right motor direction
#define IN2 5    // Right motor direction

#define ENB 11   // Left motor speed
#define IN3 6    // Left motor direction
#define IN4 7    // Left motor direction

// IR Sensors
#define IR_LEFT 12
#define IR_RIGHT 13

// Ultrasonic sensor
#define TRIG_PIN 9
#define ECHO_PIN 8

// TCS3200 Color Sensor pins
#define S0 2
#define S1 3
#define S2 A0
#define S3 A1
#define OUT A2
#define OE A3  // Output Enable (LOW = sensor ON)
#define LED A4 // LED control (HIGH = ON)

// Adjustable speed
int motorSpeed = 65; // 0-255

void setup() {
  Serial.begin(9600);

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

  // IR Sensors
  pinMode(IR_LEFT, INPUT);
  pinMode(IR_RIGHT, INPUT);

  // Ultrasonic
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // TCS3200
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(OE, OUTPUT);
  pinMode(LED, OUTPUT);
  pinMode(OUT, INPUT);

  // Configure TCS3200 frequency scaling
  digitalWrite(S0, HIGH);
  digitalWrite(S1, HIGH);
  digitalWrite(OE, LOW);    // Enable output
  digitalWrite(LED, HIGH);  // Turn on LEDs

  stopMotors();
}

void loop() {
  long distance = getDistance();

  if (distance < 15 && distance > 0) {
    stopMotors();
    delay(200);

    detectColor();
    avoidObstacle();
  } else {
    followLine();
  }
}

// Modular

// Line Following

void followLine() {
  int leftIR = digitalRead(IR_LEFT);
  int rightIR = digitalRead(IR_RIGHT);

  // Case 1: Both LOW -> Both motors forward
  if (leftIR == LOW && rightIR == LOW) {
    moveForward();
  }
  // Case 2: Both HIGH -> Both motors stopped
  else if (leftIR == HIGH && rightIR == HIGH) {
    stopMotors();
  }
  // Case 3: Mismatch - run motor on side with LOW
sensor only
  else {
```

```
    if (leftIR == LOW && rightIR == HIGH) {
      runRightMotor();
      stopLeftMotor();
    }
    else if (leftIR == HIGH && rightIR == LOW) {
      runLeftMotor();
      stopRightMotor();
    }
  }
}

// Motor control

void moveForward() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, motorSpeed);

  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  analogWrite(ENB, motorSpeed);
}

void stopMotors() {
  stopLeftMotor();
  stopRightMotor();
}

void runLeftMotor() {
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  analogWrite(ENB, motorSpeed);
}

void stopLeftMotor() {
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  analogWrite(ENB, 0);
}

void runRightMotor() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, motorSpeed);
}

void stopRightMotor() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 0);
}

// Ultrasonic sensor

long getDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH, 20000);
  long distance = duration * 0.034 / 2;
  return distance;
}

// Color detection

void detectColor() {
  Serial.println("Detecting color...");

  int red = readColor(LOW, LOW);
  int green = readColor(HIGH, HIGH);
  int blue = readColor(LOW, HIGH);

  Serial.print("R: ");
  Serial.print(red);
  Serial.print(" G: ");
  Serial.print(green);
  Serial.print(" B: ");
  Serial.println(blue);

  if (red < green && red < blue) {
    Serial.println("Detected: RED");
  } else if (green < red && green < blue) {
    Serial.println("Detected: GREEN");
  } else if (blue < red && blue < green) {
    Serial.println("Detected: BLUE");
  } else {
    Serial.println("Unknown Color");
  }
}

int readColor(bool s2, bool s3) {
  digitalWrite(S2, s2);
  digitalWrite(S3, s3);
  delay(50);
  return pulseIn(OUT, LOW);
}

// Obstacle avoidance

void avoidObstacle() {
  Serial.println("Scanning for clear path...");

  // Turn left to check
  runLeftMotor();
  runRightMotorReverse();
```

```
  delay(500);
  stopMotors();
  delay(200);

  if (getDistance() > 15) {
    Serial.println("Path clear left");
    runLeftMotor();
    runRightMotorReverse();
    delay(500);
    return;
  }

  // Turn right to check
  runLeftMotorReverse();
  runRightMotor();
  delay(1000);
  stopMotors();
  delay(200);

  if (getDistance() > 15) {
    Serial.println("Path clear right");
    runLeftMotorReverse();
    runRightMotor();
    delay(500);
    return;
  }

  // Stuck - stop
  stopMotors();
  Serial.println("No clear path found");
}

// Helper functions for reverse motors (used in
avoidance)

void runLeftMotorReverse() {
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  analogWrite(ENB, motorSpeed);
}

void runRightMotorReverse() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, motorSpeed);
}
```