Prototyping And System Engineering

Moye Nyuysoni Glein Perry
Dept: Electronic Engineering
Hochshcule Hamm-Lippstadt
Lippstadt, Germany
moye-nyuysoni.glein-perry@stud.hshl.de

Md. Tawhidur Rahman Tuhin
Dept: Electronic Engineering
Hochshcule Hamm-Lippstadt
Lippstadt, Germany
md-tawhidur-rahman.tuhin@stud.hshl.de

Raihan Uddin
Dept: Electronic Engineering
Hochshcule Hamm-Lippstadt
Lippstadt, Germany
raihan.uddin@stud.hshl.de

Bodrul Amin Murad
Dept: Electronic Engineering
Hochshcule Hamm-Lippstadt
Lippstadt, Germany
bodrul-amin.murad@stud.hshl.de

***Abstract*—This document provides an in-depth overview of our project, including the hardware design, electronic components, and the algorithm implemented to ensure the robot meets its intended objectives as outlined at the beginning of the semester. The project involves developing an autonomous robot capable of line following, obstacle avoidance, and parking based on color using an Arduino-based control system. The robot utilizes infrared (IR) sensors for line detection, an ultrasonic sensor for obstacle detection, and a TCS3200 color sensor for identifying parking zones. The system is implemented using an L293 motor driver to control the motors.**

***Index Terms*—Autonomous Robot, Line Following, Obstacle Avoidance, Parking, Arduino, Infrared Sensors, Ultrasonic Sensor, Color Sensor, Motor Driver**

I. Introduction

The goal of this project is to develop an autonomous robot capable of navigating a predetermined path, avoiding obstacles, and parking in designated zones. Upon detecting an obstacle, the robot determines its color: if blue, the robot parks; if red, it avoids the obstacle. This report explains the components used, the software implementation, and the overall system operation.

II. Requirements

We were tasked with developing a robot based on the following requirements:

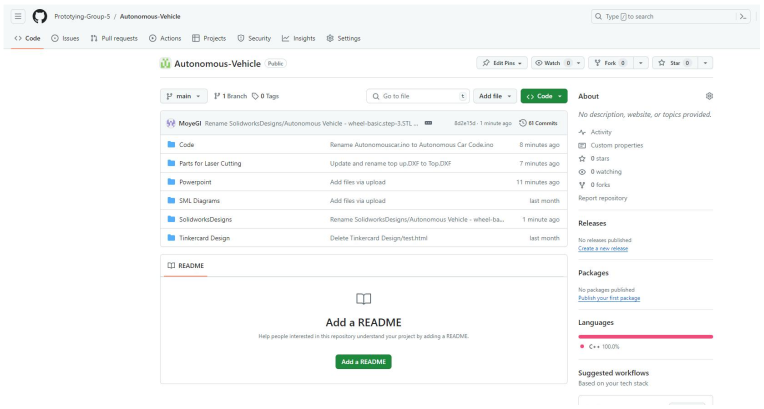
A. Develop an autonomous vehicle capable of driving independently on a given track using line detection.

B. Implement obstacle detection to avoid red obstacles and park upon detecting blue obstacles.

C. Optimize the vehicle's speed.

D. Program the vehicle to drive in various patterns, such as an oval or figure-eight.

Afterward, we developed our first prototype on Tinkercad, showcasing the electronic circuit design and the chassis design of our car. We then started developing our SysML diagrams and completed our Tinkercad design, which we uploaded to our GitHub repository.



Prototying-Group-5 / Autonomous-Vehicle

Autonomous-Vehicle Public

main · 1 Branch · 0 Tags

MoyeGI Rename SolidworksDesigns/Autonomous Vehicle - wheel-basic.step-3.STL ... 8d2e15d · 1 minute ago · 61 Commits

Code	Rename Autonomouscar.ino to Autonomous Car Code.ino	8 minutes ago
Parts for Laser Cutting	Update and rename top up.DXF to Top.DXF	7 minutes ago
Powerpoint	Add files via upload	11 minutes ago
SML Diagrams	Add files via upload	last month
SolidworksDesigns	Rename SolidworksDesigns/Autonomous Vehicle - wheel-ba...	1 minute ago
Tinkercard Design	Delete Tinkercard Design/test.html	last month

README

Add a README

Help people interested in this repository understand your project by adding a README.

About

No description, website, or topics provided.

0 stars
0 watching
0 forks

Releases

No releases published

Packages

No packages published

Languages

C++ 100.0%

Suggested workflows

Based on your tech stack

Fig. 1. GITHUB Repository

III. SML Diagrams

This section provides an overview of all the diagrams.

A. USE CASE DIAGRAM:

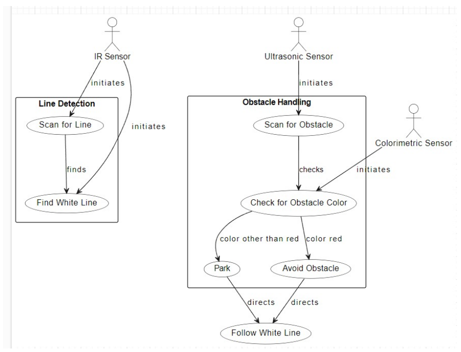


Fig. 2. Use Case Diagram

Fig.2 allows for a closer look at the autonomous system, showing you how every part talks or will talk to each other like the IR Sensor, Ultrasonic Sensor and Colorimetric sensor.

All the sensors have different tasks that are interconnected with one another to work and help in proper navigation among environmental variables from the system. Navigation: IR Sensor It searches the environment for lines thereby triggering line detection. This is an important step as it allows the system to recognize and track established paths, often in white-lined lanes. The sensor can tell the two apart to make certain that it follows its planned course-a crucial capability, enabling guidance commands for the desired path of travel. At the same time, it is improving the safety elements of the system by detecting live obstacles and changes in response. As for the vehicles, it scans within its reach to see if there are any sort that can cause technical problems for manoeuvring well. Active obstacle detection is so important because it means the system can detect when a collision might occur before it happens, and therefore act to prevent one. When it comes to detecting obstacles, the Colorimetric Sensor plays a vital role. The system measures the colour of obstacles and this is what Twin Otter calls (deciding how to react). When the sensor sees red (one of the alert colours meaning major non-critical or stop condition) then the system will start running the ”Avoid Obstacle” procedure. It is a complex function, requiring real-time calculations that ensure the vehicle can safely manoeuvre around but at the same time not give away its overall mission. Alternatively, if the object is any colour other than grey then ”Park” may be engaged by the system. This decision usually implies a desire to have the system pause temporarily so it can figure out what is going on or maybe even wait for further instructions depending on the context. After the immediate threat or obstacle is handled, it returns to ”Follow White Line,” which was its basic driving operation. In highway lane following, it is important to switch back to the primary navigation target (white line) and ensure that the vehicle continues tracking on the white route again. This ability to change from handling obstacles back to line following is a seamless one and was designed as an example of how adaptive the system is (and also its resistance in sticking true despite environmental shenanigans). In addition, the use case diagram reinforces the inclusion of sensor inputs in decision-making scenarios with their strategic placement. The video illustrates how each sensor provides a unique perspective on what is happening in the environment which can be used by intelligent decision-making algorithms to make sense of them, reducing confusion that might happen with complex scenarios. This intelligent integration ensures that the autonomous system does not just respond to current conditions but also pre-empt any potential challenges with this, setting the highest operational efficiency and safety standards. In general, The except use case diagram then is not just a sketch but also the development of the System is based on high tech framework. It shows a straightforward, systematic way from sensing to commanding that highlights the complexity and robustness of our autonomous system navigation and safety methods.

B. PACKAGE DIAGRAM:

Fig. 3 below depicts a package diagram of the structured system architecture for autonomous navigation systems, which shows how components are organized and distributed in three basic packages: Sensor Suite, Operational Logic and Decision Logic. Each package fulfils independent but very much related roles in the system, facilitating a well-coordinated and fine-grained functionality. The Sensor Suite is the bottom-most

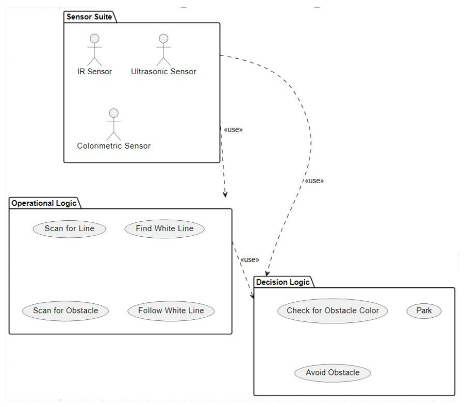


Fig. 3. Package Diagram

level of our system and it contains IR, Ultrasonic and Colorimetric sensors. These sensors are extremely important when it comes to how this system interfaces with the environment. IR Sensor - line detection which helps to navigate the vehicle throughout redecided paths. Ultrasonic Sensor detects obstacles eliminating potential collisions, and the Colorimetric sensor starts assessing the colour characteristics of these detected obstacles determining high-level threats based on their colours. The operational package continues that trend, implementing important use cases like ”Scan for Line”, ”Find White Line”, and the course of it all, some workflow starting with three basic bricks. These functions are also immediately linked to the navigation acting on the outputs of all sensors in the Sensor Suite making it possible for system-based basic control-task execution. Some of these include vital use cases for the continued functioning of the system, such as requirements to be able to keep running and control a vehicle in traffic and corporate or ad hoc modes. Additional decision logic that performs higher level decisions on the data from the Sensor Suite, in the Decision Logic package. This consists of use cases such as Check for Obstacle Colour; which has implications on future decisions like Park or Avoid Obstacle. These decisions determine how the system reacts to any obstacles it detects, by either pausing or bypassing and allowing navigation safely and continuously. These packages interact in an orderly manner and outline the flow of information as well as commands within the system. The data from the collection and processing of the environment as captured by The Sensor Suite is used by The Operational Logic to maintain main basic navigation roles. This data then feeds into the Decision Logic, which processes it and creates decisions that help in deciding how a system will adapt to its environment or move around. This diagram is well representative of the architecture in such a way that there are specific units and processes which have been compartmentalized yet at certain features they are all connected to ensure smooth functioning while making powerful decisions. This modularity not only makes it easy to learn how the system works but also represents its scalability and maintainability characteristics that aim at preparing this same navigation software to be able to adjust itself over time when new challenges (and technologies) come up.

C. *SEQUENCE DIAGRAM:*

Fig. 4 represents the connection of various subsystems within an autonomous vehicle. The diagram illustrates the sequential progress from initial detecting tasks to the execution of advanced driving sequences and parking.

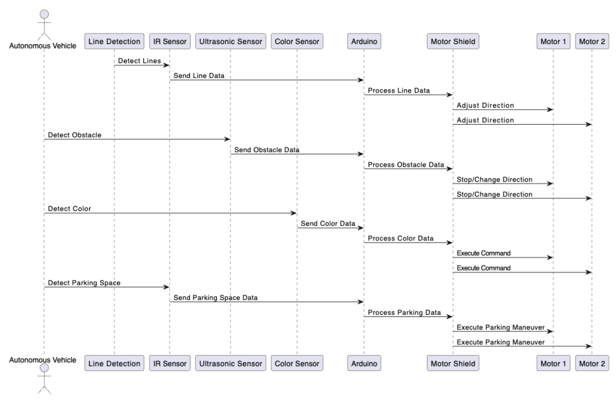


Fig. 4. SEQUENCE DIAGRAM

Actions and Roles: Every component of the autonomous vehicle has its own functions and responsibilities within the series of tasks it performs. Horizontal arrows represent the actions, and describe the sequence of operations and their interactions.

1) Detect Lines (IR Sensor - Arduino): The IR Sensor sends the detected line data to the Arduino. The Arduino processes the line data and sends motor commands to the Motor Shield. The Motor Shield adjusts the direction of Motor1 and Motor2 based on the commands.

2) Detect Obstacle (Ultrasonic Sensor - Arduino): The Autonomous Vehicle initiates obstacle detection by sending a command to the Ultrasonic Sensor. The Ultrasonic Sensor sends obstacle data to the Arduino. The Arduino processes the obstacle data and sends appropriate commands to the Motor Shield. The Motor Shield commands Motor1 and Motor2 to stop or change direction based on the obstacle data.

3) Detect Color (Color Sensor - Arduino): The Autonomous Vehicle sends a command to the Color Sensor to detect color. The Color Sensor sends the detected color data to the Arduino. The Arduino processes the color data and sends commands to the Motor Shield. The Motor Shield executes the command on Motor1 and Motor2 based on the color data.

4) Detect Parking Space (IR Sensor - Arduino): The AutonomousVehicle initiates parking space detection by sending a command to the IR Sensor. The IR Sensor sends parking space data to the Arduino. The Arduino processes the parking data and sends commands to the Motor Shield. The Motor Shield executes the parking maneuver on Motor1 and Motor2.

Sequence Flow: The sequence diagram shows the sequential connection among these components, demonstrating the autonomous vehicle's development from line detection to parking.

D. *REQUIREMENT DIAGRAM:*

Fig. 5 represents a requirement model for an autonomous vehicle system and shows the fundamental features and connections required for the vehicle to perform efficiently. Func-

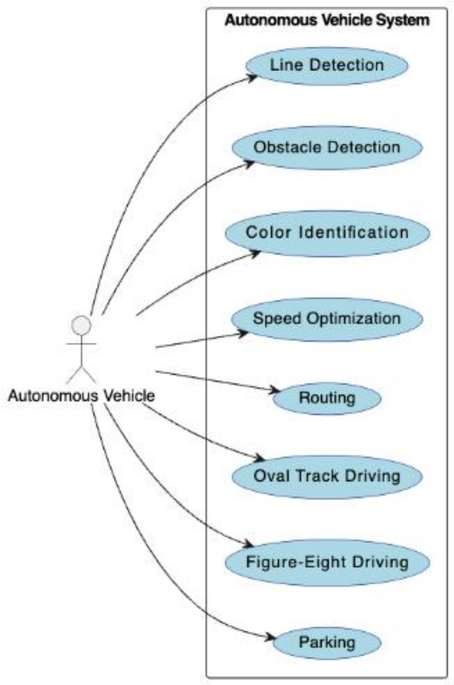


Fig. 5. REQUIREMENT DIAGRAM

tionalities:

1. Line Detection
2. Obstacle Detection
3. Color Identification
4. Speed Optimization
5. Parking

Roles: Each Functionality represents an individual action that the vehicle must execute. Autonomous vehicle plays the primary role. It will execute the required functionalities.

Summary: The diagram gives a overview of the main functions that an autonomous vehicle needs to be able to drive itself. It includes the key features a vehicle needs to perform such as detecting lines and obstacles, noticing colors and parking.

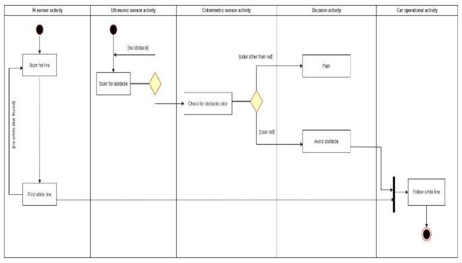


Fig. 6. ACTIVITY DIAGRAM

E. ACTIVITY DIAGRAM:

Fig. 6 illustrates the operation of a robotic system designed to follow a black line and handle obstacles.

1. Start the Power Source: The process begins with powering on the system using a 9V lithium battery.

2. Initial Actions:

Scan for Obstacles in Front: The system uses an ultrasonic sensor to detect if there's an obstacle directly ahead.

- If an obstacle is detected, the system proceeds to idle mode and then scans for the obstacle's color using a colorimetric sensor.
- If no obstacle is detected, it proceeds to scan for the black line.

Scan for Black Line: Using an IR sensor, the system checks for the presence of the black line.

- If the black line is detected, it goes idle.
- If no black line is detected, the system continuously searches for it.
-

3. Obstacle Handling:

After detecting an obstacle and identifying its color, the system scans for obstacles on either side (left and right) using the colorimetric sensor.

- If no obstacle is detected on either side, the system detracks from the black line, rotates the car until the black line is detected again, aligns with the line, and moves forward.
- If an obstacle is detected, the system determines the side (left or right) with the obstacle and uses a servo motor to rotate the wheel in the opposite direction until no obstacle is detected. It then detracks from the black line, rotates the car to find the black line, aligns with it, and moves forward.

4. Idle State: If the system detects the black line or after handling an obstacle, it goes idle for a moment before continuing with further actions.

5. Moving Forward: Once the black line is detected and obstacles are handled, the system moves forward along the black line.

6. End Process: After the task is completed, the system turns off the power.

Key Components and Their Roles:

- Ultrasonic Sensor: Detects obstacles in front.
- IR Sensor: Scans for the black line.
- Colorimetric Sensor: Scans for obstacle color and detects obstacles on the sides.

Flow Control:

- Decision Points: Represented by diamond shapes, where the system decides the next action based on sensor inputs.
- Actions: Represented by rectangles, describing the operations performed by the system.
- Connectors: Arrows indicating the flow from one action or decision point to another.
- Start and End: Represented by ovals, indicating the beginning and end of the process.

This activity diagram provides a clear overview of the robotic system's logic for navigating a path marked by a black line and managing any obstacles encountered.

F. BLOCK DIAGRAM:

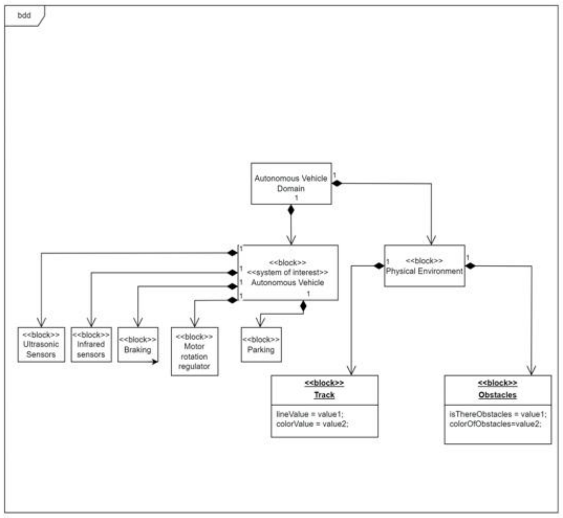


Fig. 7. BLOCK DIAGRAM

Fig. 7 is a Block Definition Diagram (BDD) in SysML (Systems Modeling Language), representing the structure of an autonomous vehicle system within a specified domain.

Main Components:

- **Autonomous Vehicle Domain:** This is the overarching domain for the system, encapsulating both the autonomous vehicle and its interaction with the physical environment.

- **System of Interest (Autonomous Vehicle)**: This block represents the autonomous vehicle itself, containing various subsystems that are crucial for its operation.

Subsystems within the Autonomous Vehicle:

1. Ultrasonic Sensors: These sensors likely measure distances to objects using ultrasonic waves, helping in obstacle detection.

2. Infrared Sensors: These sensors might be used for detecting heat signatures or for proximity sensing.

3. Braking: This subsystem handles the vehicle's braking mechanism, ensuring the vehicle can slow down or stop as needed.

4. Motor Rotation Regulator: This block is responsible for controlling the rotation of the vehicle's motors, influencing speed and direction.

5. Parking: This subsystem manages the vehicle's ability to park itself autonomously.

Interaction with Physical Environment:

1. Physical Environment: This block represents the external environment in which the autonomous vehicle operates. It includes:

- **Track**: Defined by attributes lineValue and colorValue, this likely represents the path or road the vehicle follows, with specific values for the line and its colitor.
- **Obstacles**: This block denotes obstacles the vehicle might encounter, with attributes isThereObstacles (indicating presence of obstacles) and colorOfObstacles (providing information about the color of obstacles).

Relationships and Multiplicity:

- The arrows with diamonds denote composition relationships. The black filled diamonds indicate that the part (e.g., subsystems) is an essential component of the whole (e.g., autonomous vehicle) and its lifecycle is tied to the whole.
- Multiplicity (indicated by '1' at each end of the relationship) shows that there is exactly one instance of each relationship within the context (e.g., one autonomous vehicle within the autonomous vehicle domain).

G. STATE MACHINE DIAGRAM:

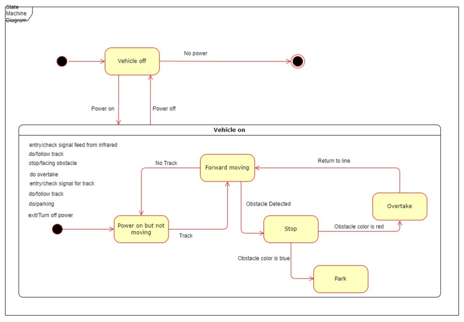


Fig. 8. STATE MACHINE DIAGRAM

States and Transitions:

1. Initial State:

- **Vehicle Off:** The vehicle in this state has no power.
- **Transition to "Vehicle On":** The stage where the vehicle is turned on.

2. Substates of "Vehicle On":

Power on but not moving:

- Entry Action: Check signal from infrared sensors.
- Activities: Follow track, stop when facing an obstacle, do overtaking or parking.
- Exit Action: Turn off power.
- Transition: If a track is detected, it shifts to state- "Forward moving". If no track is detected, it remains in the state.

- **Forward moving:**
- Activities: Follow track.
- Transition: If an obstacle is detected, it transitions to "Stop" state. If the vehicle needs to return to the line, it goes back to "Forward moving" with track scanning depending on the color of the obstacles.
- **Stop:**
- Activities: Avoiding the obstacle.
- Transition: If the obstacle color is blue, it transitions to "Park". If the obstacle color is red, the vehicle overtakes.
- **Overtake:**
- Activities: Perform overtaking the obstacle.
- Transition: Once the overtaking is done, it transitions back to tracking the line and "Forward moving".
- **Park:**
- Activities: The vehicle performs parking.
- Transition: The vehicle goes parking.

H. UPPAAL MODEL:

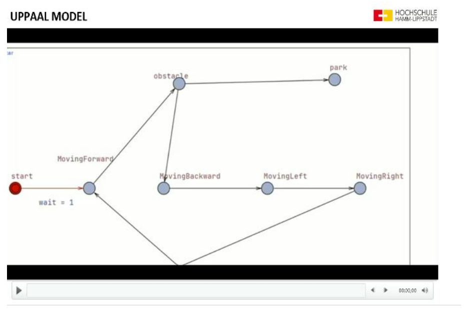


Fig. 9. UPPAAL MODEL

This is the state where The model kicks off as soon you push it to on ('start') - after that, the car recently stops for a slight while (it's around just allowing all of its systems to get themselves synchronized and stable prior then moving). This first wait cycles the vehicle on, during which it can adjust all sensors and functions for peak performance. All commands are first issued and then the initial state of operation is 'Moving Forward', in which case the car continues on a white line, using its built-in pathfinding strategy The system demonstrates its dynamic response capabilities when faced with an obstacle. Our model says the vehicle should first try to tackle this by going into 'Moving Backward'. This Virtual Uppercut specifically serves to build a divide upfront and open up a plethora of future movement opportunities. The vehicle then determines whether it could shift left or right (moves to the 'Moving Left/Right' state in the respective transitions) based on its surroundings and obstacle location. This thought is important for an uninterrupted movement of the navigation channel without breaking the borders of safe and efficient movements. This model comes with parking assistance too which is based on color detection. That is, the vehicle changes to a 'Park' state when it reads red colour from this pyrometer sensor. In areas where red may mean a stop sign of some sort or no entry, this feature is vital as it requires the vehicle to come to an immediate but safe halt. The UPPAAL model allows the project to not only simulate these conditions and transitions but also rank them in order of likelihood for testing purposes that when faced with real-world variables and scenarios, the prototype car's navigation system has been proven to handle nearly anything. By creating an integrated sensor model and embedding AI in the simulated environment, engineers can debug any potential flaws they find within the vehicle's decision logic - ensuring robust deployment performance. This detailed operational verification in the UPPAAL model is essential to be able to measure intermediate results of what has been reached and acquired by this project. This showcases that the model can mimic complex interactions in an autonomous navigation system and confirms there will be a potential for high-fidelity optofluidic actuation thereby making our end-effector bamboo-robotism directly applicable - with initial parameters provided by the corresponding ML-based LQR-synthesis. This work makes a significant contribution to the AV field, and more specifically, in improving reliability & safety using sophisticated simulation methods

IV. Hardware Components Used:

The prototype comprises several electronic components, each with distinct functions and unique properties. Here is a summary of each component:

A. *Microcontroller (Arduino)*

Fig. 10. Ardunio

The selected microcontroller for this project is the Arduino Uno, which serves as the central control unit. The Arduino Uno is an open-source electronics platform based on easy-to-use hardware and software. It reads inputs from various sensors and controls the motors based on programmed logic. The development board incorporates the ATmega328P microcontroller from the Atmel megaAVR family. It has 14 digital input/output pins, 6 of which support PWM (Pulse Width Modulation): pins 3, 5, 6, 9, 10, and 11. Communication with a PC is established through a USB connection. The Arduino Uno features an 8-bit RISC (Reduced Instruction Set Computer) processor core.

B. *IR Sensors*

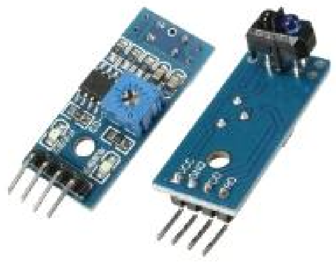


Fig. 11. IR sensor

Infrared (IR) sensors are used to detect the line on the ground. Two IR sensors are placed at the front of the car to detect the line and help the robot follow the path accurately. They emit infrared light onto the surface; when the return value is 1, it indicates black, and when it is 0, it indicates white. This allows the IR sensors to determine whether the car is positioned on or off the line. These sensors are crucial for ensuring the car follows the line precisely.

C. *Ultrasonic Sensor*

Fig. 12. Ultrasonic Sensor

The ultrasonic sensor is used to measure the distance to obstacles. It consists of a transmitter and receiver that send and receive ultrasonic waves. The time taken for the waves to return is used to calculate the distance to an obstacle. This sensor is essential for detecting obstacles in front of the car.

Distance = Speed * duration

Speed of sound in air and it is in meters/s so we have to convert it to cm/microsecond.

D. *Motor Driver*

Fig. 13. L298N motor driver

The L298N motor driver is used to manage the speed and direction of the DC motors in the project. This component can independently drive two DC motors, enabling movements in forward, backward, and stop directions. The motor driver receives control signals from the Arduino to operate effectively. Moreover, the L298N is adept at precise speed control through its enable A and enable B pins. Additionally, it utilizes Int 1 to 4 to determine and control the direction of each motor. This capability ensures accurate motor control and movement as per the project's requirements.

E. *Chassis and Wheels*

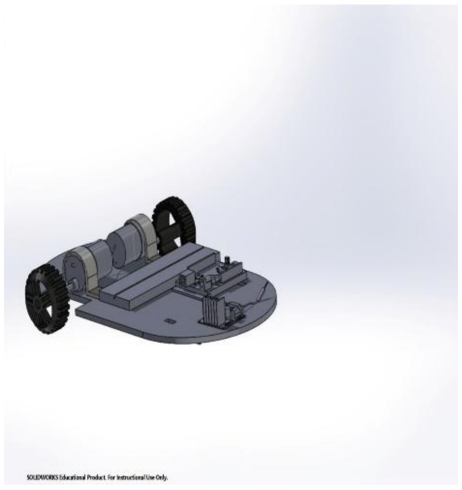


Fig. 14. 3D chasis

The wheels are crucial components connected directly to the motors, enabling the car to move forward. The vehicle features three wheels: two larger wheels at the back directly connected to the motors, which are essential for propulsion, and a smaller wheel positioned under the chassis at the front to maintain stability. The chassis, made from laser-cut plank material, serves as the structural foundation to house and support all components of the vehicle. It has been designed to accommodate all necessary elements, ensuring robustness and functionality throughout the project.

F. *Motors*

Fig. 15. Motors

DC motors provide the essential movement for the vehicle in this project. Two motors are employed, with each motor connected to the L293N motor driver. The Arduino sends PWM signals to the motor driver to regulate both the speed and direction of the motors. The motor driver is powered by a 12V battery supply, enabling it to provide sufficient power for the motors. The motors have a speed reduction ratio proportional to their torque output, ensuring effective performance. Specifically, a speed setting of 85 has been selected to accommodate the weight and movement requirements of the robot.

G. *Color Sensor*

The TCS3200 color sensor is employed to detect the colors of obstacles in the robot's environment. This sensor is capable

Fig. 16. Color Sensor

of distinguishing between different colors and plays a crucial role in identifying specific parking zones. When the sensor detects a blue color, the robot is programmed to move backward and then park. Conversely, when it detects a red color, the robot moves backward and maneuvers around the obstacle. This functionality allows the robot to effectively navigate and respond to its surroundings based on color detection.

H. *Power Supply*

Fig. 17. Battery

The source of power for the robot is a rechargeable Polymer Li-Ion Battery. This battery provides the required voltage and current to all components of the robot, ensuring stable operation of the Arduino, sensors, and motors. It plays a critical role in maintaining consistent power supply, which is essential for the proper functioning and performance of the robot throughout its operation.

I. *Breadboard and Connecting Wires*

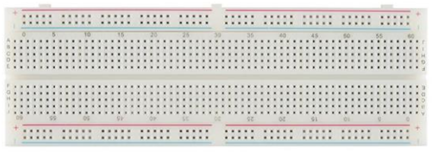


Fig. 18. Breadboard

The breadboard and cables are used to create connections between various electronic components in the robot's circuitry. The breadboard serves as a platform for prototyping and testing different configurations without soldering. It allows for easy insertion and removal of components, facilitating quick adjustments and modifications during the development process.

The cables, on the other hand, provide the necessary pathways for electrical signals to travel between components, ensuring proper communication and functionality throughout the robot's system. Together, the breadboard and cables enable the assembly and testing of the electronic circuitry with flexibility and ease of use.

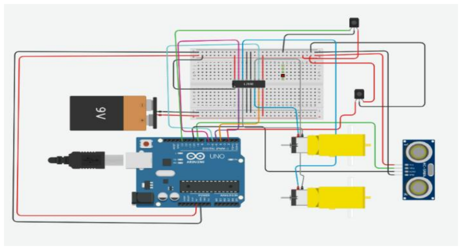


Fig. 19. Electronic Circuit

V. PROTOTYPE DESIGN

A. *Electronic Circuit:*

For the electronic circuit design phase of our project, we used Tinkercad, While designing our circuit, we encountered a challenge as the color sensor we intended to use wasn't available in Tinkercad's library. Despite this, Tinkercad proved invaluable as it allowed us to create and simulate our circuit design. We laid out each component on the virtual breadboard, ensuring correct wiring and connections between the Arduino, motors, and ultrasonic sensor. Through iterative testing and coding in Tinkercad's simulation environment, we observed how the motors and ultrasonic sensor behaved in response to programmed instructions. Tinkercad served as a guiding tool, providing us with a visual representation of our circuit design and helping us understand the necessary connections and configurations. This process enabled us to troubleshoot potential issues and refine our design before physical implementation. Furthermore, the simulation capabilities of Tinkercad allowed us to test various scenarios, ensuring that our circuit design met the project's requirements for functionality and performance. By using it, we gained valuable insights into the operational dynamics of our electronic system, paving the way for a well-informed and successful prototype implementation

B. *TinkerCAD*

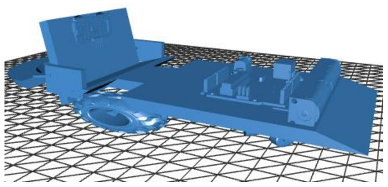


Fig. 20. Tinker Cad Design

Tinkercad proved to be an invaluable tool for designing our car. As a web-based software, it eliminated the need to download complex programs, offering a user-friendly environment accessible from any internet-connected device. We primarily utilized Tinkercad for prototyping, taking advantage of its intuitive interface and diverse set of tools. Initially, we began our design with the intention of incorporating obstacle-removal capabilities. However, as our design evolved, we adapted our approach to better suit the project's objectives and constraints. Tinkercad allowed us to navigate through various design iterations seamlessly, enabling us to refine and optimize our prototype efficiently.

C. SolidWorks

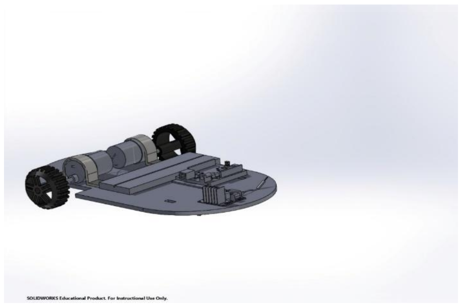


Fig. 21. Solid woks design

For the 3D SolidWorks design phase of our car project, we opted for SolidWorks, a robust software known for its advanced capabilities in creating detailed 3D models. This decision was driven by our need to design intricate components such as motor holders to secure the motors and prevent unwanted movement.

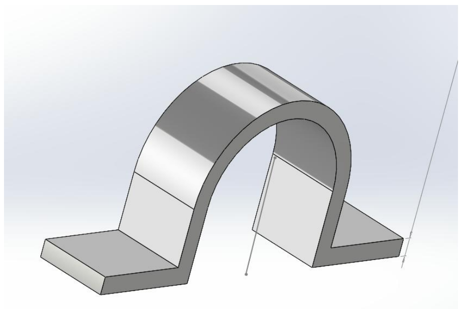


Fig. 22. Views on solid works

Each component, including the base, which was later modified, the front and rear sections which were precisely printed, and the support designed to house the ultrasonic sensor, was meticulously crafted and integrated into our overall design.

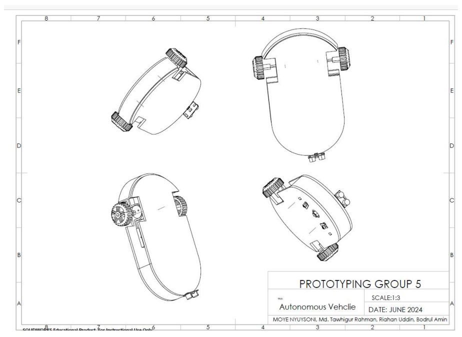

PROTOTYPING GROUP 5
Autonomous Vehclie
SCALE:1:3
DATE: JUNE 2024
MOYE NYUYSONI, Md. Tawhigur Rahman, Riahan Uddin, Bodrul Amin

Fig. 23. Final solid works design

One of the significant advantages of using SolidWorks was its ability to facilitate precise modeling and detailed assembly. Each part was designed with mounting holes, allowing us to securely fasten them to the chassis board using screws. This ensured stability and durability in our prototype. To validate our designs and ensure functionality, we conducted simulations within SolidWorks. These simulations allowed us to verify the fit and functionality of each component, ensuring they met our project's specific requirements before physical fabrication.

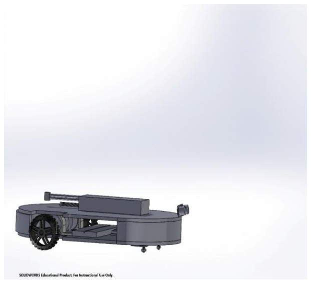


Fig. 24. Final solid works design

By leveraging SolidWorks for our 3D design phase, we achieved a streamlined process that enabled us to innovate and refine our car prototype efficiently, ensuring optimal performance and functionality.

D. Final Design:

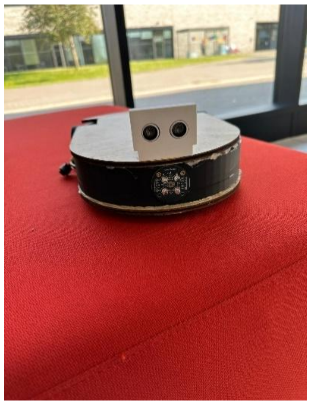


Fig. 25. Final Design

Our final design seamlessly integrates the detailed 3D components from SolidWorks with the designed electronic circuit from Tinkercad. This integration culminated in the creation of our real-life car prototype, showcasing both its visual appearance and functional capabilities. In our physical prototype, each 3D-printed part, carefully crafted using SolidWorks, complements the precision-cut components from laser cutting. We utilized glue and screws to securely assemble these parts, ensuring structural integrity and stability throughout the vehicle.

The collaborative effort of combining our designs and elements was instrumental in bringing our prototype to life. The 3D parts, printed to exact specifications, fit with the laser-cut pieces, creating a cohesive and robust chassis for our car. This cohesive design approach not only enhanced the aesthetic

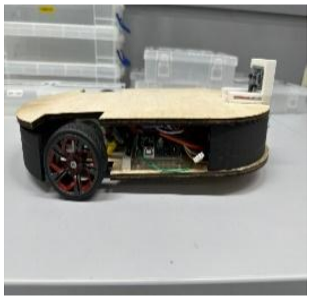


Fig. 26. Final design

appeal of our prototype but also ensured optimal performance and functionality. Furthermore, our physical prototype serves as a testament to the iterative design process we undertook, where insights gained from Tinkercad simulations informed refinements in our SolidWorks models, and vice versa. This iterative approach allowed us to iteratively improve the design and functionality of our prototype, ensuring it met the project's objectives effectively. Overall, the combination of SolidWorks and Tinkercad facilitated a comprehensive design and development process, resulting in a tangible prototype that embodies our team's collective effort and innovation.

E. Assembly Instructions of the vehicle:

1) Necessary Components:

- 1 Battery
- 1 Breadboard
- 1 Motor Shield
- 2 Ultrasonic Sensors
- 1 Front wheel
- 1 Arduino Uno
- 2 Tires
- 2 IR Sensors
- 2 Motors
- Wires
- Chassis

2) Tools:

- Screwdriver
- Hot Glue gun
- Wire cutters

Step 1: Prepare the chassis

1) Install the Front Wheel: We have screwed the Front Wheel to the bottom of the chassis. So it can spin effortlessly
2) Attach the Tires : Placed the tires along with the motors at the back of the chassis. The tires will touch the ground when they are attached to the chassis.

Step 2: Install the motors:

Assemble the Motors:

- Connected the motors to the wheels on the back of the chassis.
- Designed a holder for the motors that will keep them stable while the car is moving.

Connect Motors to Motor Shield:

1) For example Motor 1
2) Enable pin (enA): Arduino pin 10
3) Input pin (in1): Arduino pin 6
4) Input pin (in2): Arduino pin 7
5) For Motor 2:
6) Enable pin (enB): Arduino pin 9
7) Input pin (in3): Arduino pin 12
8) Input pin (in4): Arduino pin 8

Position the Arduino Uno:

- We placed the Arduino Uno in the centre of the chassis.

Attach the Motor Shield to the Arduino:

- Next, we positioned the motor shield alongside the Arduino Uno and ensured that the pins were correctly aligned.

Step 3: Connect the IR Sensors Position the IR Sensors

- We installed the IR Sensors in front of the vehicle. There is a space between two sensors that detect the track line on both sides.

Connect IR Sensors to the Arduino:

- We linked the right infrared sensor to Arduino pin 5.
- The left infrared sensor is connected to Arduino 4.

Step 4: Install the Ultrasonic Sensors

Position the Ultrasonic Sensors:

- We installed the Ultrasonic Sensors in front of the vehicle
- It will face the forward track to detect the obstacle.

Connect Ultrasonic Sensors to the Arduino:

- The trig pin is connected to Arduino pin 2.
- The echo pin is connected to Arduino pin 3.

Step 5: Delivering the System Power

Connect the battery:

- Next, we connected the battery to the Arduino's power jack.
- Connected the battery to the breadboard, then use it to power the Arduino and motor shield.

F. Testing Procedures and instructions:

Hardware Testing Procedures: Before doing hardware component testing, we did an initial check to verify that all components were properly installed and all wires were properly connected. We also confirmed the battery connection and voltage to guarantee a consistent power supply. The initial procedure included activating the system and checking for unusual activity. We started sensor testing with the IR sensors. We uploaded a test code to the Arduino micocontroller in order to the gather data from the IR sensors. After that, we positioned the vehicle on the selected track and monitored the data outputs. This helped us confirm the sensors can detect the tracks correctly. Next we did a series of tests on the Ultrasonic sensors. We did an experiment by uploading a test code to calculate the distance. In this experiment, we positioned objects at certain distances from the sensors and verified the accuracy of the results. Then, we tested the motors to see that they ran forward, backward and stopped correctly.

Software Testing Procedures: During the test, we first made sure that IR and Ultrasonic sensors were reading important inputs correctly by uploading code. It was crucial for figuring out how well those sensors could provide accurate data when they connected to the vehicle's control system. Next we put the system's motor control and sensor parts together. Line following. Obstacle detection, speed optimization and parking tests were part of functional testing.

Test Results: During the hardware testing process, we have seen several problems. Initially, we have noticed that IR sensors were giving inconsistent readings due to varying lighting conditions. We tried the sensors in different lighting conditions to solve this problem. In the case of ultrasonic sensors, It was not accurate to measure the distance from the obstacles. So we have worked on our test code to solve this problem. We have also faced some challenges in software testing. Line and Obstacle detection provided inconsistent results. We have reviewed our test code to solve this problem.

Calibration Instructions: We calibrated the IR sensors by uploading a calibration code to the Arduino main board. We also fixed the height and angle of the IR sensors to increase their ability to detect line. When it comes to Ultrasonic sensors, we placed obstacles at certain distances to test their accuracy and functionality. We have refined our code to achieve accurate obstacle detection.

Operating Instructions: To use the autonomous vehicle, we should check that the battery is fully charged and properly connected. To start the vehicle, turn on the power. Turning on the vehicle starts an initial setup phase. During the testing phase, the vehicle must be on track so that we can see that all components of the vehicle are working accurately.

Safety Instructions: When driving a Autonomous vehicle, It is very crucial to make sure the testing area is safe and free of any dangers. No unnecessary obstacles should interfere with the track. Take care when handling the battery, and ensure its connected properly to avoid any short circuits. Do not overcharge the battery. We should follow the guidelines for charging and handling the battery. We should be aware when handling the sensors and motors. During the project, we used different kinds of hardware to develop the vehicle that could pose potential risks to our personal safety. We must take care when it comes to our personal safety.

G. Programming the assembly:

Motor Control Functions

The motor control functions are essential for moving the car. These functions control the L298N motor driver by setting the appropriate pins high or low, which in turn drives the motors in the desired direction.

1) The forward function makes the robot move forward by setting the right motor to spin forward (in1 LOW, in2 HIGH) and the left motor to spin forward as well (in3 HIGH, in4 LOW).
2) The backward function makes the robot move backward by reversing the polarity of the motor pins (in1 HIGH, in2 LOW for the right motor and in3 LOW, in4 HIGH for the left motor).
3) The TurnRight function turns the robot to the right by stopping the left motor and moving the right motor forward (in1 LOW, in2 HIGH). This creates a pivot point on the left wheel, making the robot turn.
4) The TurnLeft function turns the robot to the left by stopping the right motor and moving the left motor forward (in3 HIGH, in4 LOW). This creates a pivot point on the right wheel, making the robot turn.
5) The Stop function stops the robot by setting all motor control pins to LOW, effectively cutting off the power to the motors.

Speed Control Functions

These functions adjust the speed of the motors to make the robot move smoothly.

- The accelerator function helps the motor speed gradually until it reaches the maxSpeed.
- The decelerateMotor functon Gradually decreases the motorspeed until it reaches the minSpeed.
- The Ultrasonic sensor function Measures the distance to an obstacle using an ultrasonic sensor.
- The Function avoidObstacle Implements a sequence to avoid obstacles detected by the ultrasonic sensor.

  Steps:
  - Move backward for 1000 milliseconds.
  - Turn left for 1200 milliseconds.
  - Move forward for 700 milliseconds.
  - Turn right for 1700 milliseconds and move
  - forward again for 300 milliseconds to realign with the path.
- The park function helps the robot park in a designated spot.

  Steps:
  - Move backward for 1000 milliseconds.
  - Align correctly by turning left for 1500 milliseconds.
  - Move forward for 1000 milliseconds to reach the parking spot.
  - Stop for an extended period (100000 milliseconds) to keep the robot parked.

  The function followLine Keeps the robot following a line using IR sensors.

  It does that by controlling the system's movement

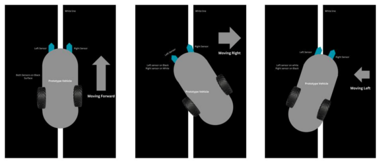


Fig. 27. IR Sensor movement demonstration

based on the inputs from two sensors (rightSensor and leftSensor). The sensors' readings are combined using a bitwise operation (rightSensor ¡¡ 1 — leftSensor) to determine the system's state:
When both sensors are on the line (0b11), the system stops for 1000 milliseconds (Stop(); delay(1000);). This is typically a condition to handle intersections or endpoints. When the right sensor is on the line and the left sensor is off (0b10), the system turns right (turnRight();) and decelerates the motor (decelerateMotor();). When the left sensor is on the line and the right sensor is off (0b01), the system turns left (turnLeft();) and decelerates the motor (decelerateMotor();). When both sensors are off the line (0b00), the system moves forward (forward();) and accelerates the motor (accelerateMotor();). For any other sensor readings (handled by default), the system moves forward (forward();) and decelerates the motor (decelerateMotor();). This acts as a fallback to ensure the system continues moving even with unexpected sensor inputs.

Various states of the system:
Enum definition for the various states the system can be in

Enum State

FOLLOW LINE, // State for line following behavior

AVOID OBSTACLE, // State for obstacle avoidance behavior

PARK, // State for parking behavior

STOPPED, // State for stopping behavior

The State enumeration defines the different modes the system can operate in:

FOLLOW LINE: This state makes the system follow a line by utilizing inputs from the right and left sensors.

AVOID OBSTACLE: This state engages the system's obstacle avoidance mechanism.

PARK: This state activates the parking procedure.

STOPPED: This state brings the system to a halt.

A switch statement is used to handle the behavior of the system based on the current state (currentState):
In the FOLLOW LINE state, the follow line function is called with right Sensor and left Sensor as parameters, enabling the system to follow a line using these sensor inputs.

In the AVOID OBSTACLE state, the avoidObstacle function is called to navigate around any detected obstacles. After successfully avoiding an obstacle, the system transitions back to the FOLLOW LINE state.

In the PARK state, the Park function is invoked to park the vehicle.

In the STOPPED state, the Stop function is executed to stop all movements of the system.

The process red value, process green value, and process blue value functions read color values from a color sensor by configuring specific pins and measuring the pulse length:
process red value:
digitalWrite(S2 PIN, LOW); and digitalWrite(S3 PIN, LOW); set the sensor to read the red color component. A delay of 100 milliseconds (delay(100)) allows the sensor to stabilize. The pulseIn(OUT PIN, LOW); function measures the duration of a pulse in microseconds while the signal is LOW, which corresponds to the intensity of the red color. The measured pulse length is returned as the red color value.

process green value:
digitalWrite(S2 PIN, HIGH); and digitalWrite(S3 PIN, HIGH); set the sensor to read the green color component. A delay of 100 milliseconds (delay(100)) allows the sensor to stabilize. The pulseIn(OUT PIN, LOW); function measures the duration of a pulse in microseconds while the signal is LOW, which corresponds to the intensity of the green color. The measured pulse length is returned as the green color value.

process blue value:
digitalWrite(S2 PIN, LOW); and digitalWrite(S3 PIN, HIGH); set the sensor to read the blue color component. A delay of 100 milliseconds (delay(100)) allows the sensor to stabilize. The pulseIn(OUT PIN, LOW); function measures the duration of a pulse in microseconds while the signal is LOW, which corresponds to the intensity of the blue color. The measured pulse length is returned as the blue color value.

Each function configures the sensor to measure a specific color by setting the appropriate pins and then reads the pulse length to determine the color intensity. The delay ensures the sensor provides accurate readings after configuration.

The Park function handles the procedure for parking the system when it finds blue:
Step 1: Process the color sensor values.
The function process blue value() is called to get the blue component (b). The function process green value() is called to get the green component (g). The function process red value() is called to get the red component (r).

Step 2: The system moves backward slightly to start the parking maneuver.
The backward() function is called to move the system backward. A delay of 1000 milliseconds (delay(1000)) ensures the system moves back a bit. The system stops (Stop()) and pauses for 500 milliseconds (delay(500)). The state is set to FOLLOW LINE to prepare for the next maneuver.

Step 3: The system aligns perfectly for parking.
The aturnLeft() function is called to turn the system left. A delay of 1500 milliseconds (delay(1500)) ensures correct alignment. The system stops (Stop()) and pauses for 500 milliseconds (delay(500)). The state is set to FOLLOW LINE to continue the parking process.

Step 4: The system moves forward into the parking spot.
The forward() function is called to move the system forward. A delay of 1000 milliseconds (delay(1000)) adjusts the movement into the perfect spot. The system stops (Stop()) and pauses for 500 milliseconds (delay(500)). The state is set to FOLLOW LINE to finalize the parking process.

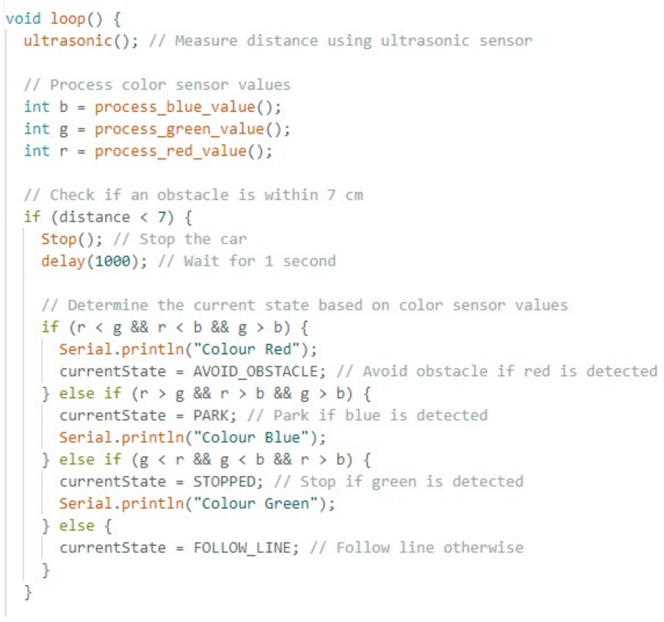
Step 5: The system ensures it remains stopped.
The Stop() function is called to keep the system stationary. A long delay of 100000 milliseconds (delay(100000)) ensures the system stays stopped for an extended period.

Conditional Check: If the red component is the highest among the color values and greater than both green and blue components(means it is red), the system will invoke the avoidObstacle() function to handle the detected condition.

The avoidObstacle function manages the system's behavior when an obstacle is detected:
Step 1: The system stops and moves backward to create distance from the obstacle.
The backward() function is called to move the system backward. A delay of 1000 milliseconds (delay(1000)) is introduced to adjust the distance moved backward. The system stops (Stop()) and pauses for 500 milliseconds (delay(500)).

Step 2: The system turns left to begin maneuvering around the obstacle.
The aturnLeft() function is called to turn the system left. A delay of 1200 milliseconds (delay(1200)) controls the angle of the turn. The state is set to FOLLOW LINE to prepare for resuming line following. Step 3: The system moves forward to pass the obstacle.
The forward() function is called to move the system forward. A delay of 700 milliseconds (delay(700)) adjusts for the size of the obstacle. Step 4: The system turns right to realign with the path.
The aturnRight() function is called to turn the system right. A delay of 1700 milliseconds (delay(1700)) controls the angle of the turn. The state is set to FOLLOW LINE to continue with line following. The system moves forward again (forward()) with a delay of 300 milliseconds (delay(300)). The state is again set to FOLLOW LINE to finalize the realignment. Each step includes delays that can be adjusted based on the specific requirements of the obstacle avoidance maneuver and the size of the obstacle.

Code:



```
void loop() {
  ultrasonic(); // Measure distance using ultrasonic sensor

  // Process color sensor values
  int b = process_blue_value();
  int g = process_green_value();
  int r = process_red_value();

  // Check if an obstacle is within 7 cm
  if (distance < 7) {
    Stop(); // Stop the car
    delay(1000); // Wait for 1 second

    // Determine the current state based on color sensor values
    if (r < g && r < b && g > b) {
      Serial.println("Colour Red");
      currentState = AVOID_OBSTACLE; // Avoid obstacle if red is detected
    } else if (r > g && r > b && g > b) {
      currentState = PARK; // Park if blue is detected
      Serial.println("Colour Blue");
    } else if (g < r && g < b && r > b) {
      currentState = STOPPED; // Stop if green is detected
      Serial.println("Colour Green");
    } else {
      currentState = FOLLOW_LINE; // Follow line otherwise
    }
  }
```

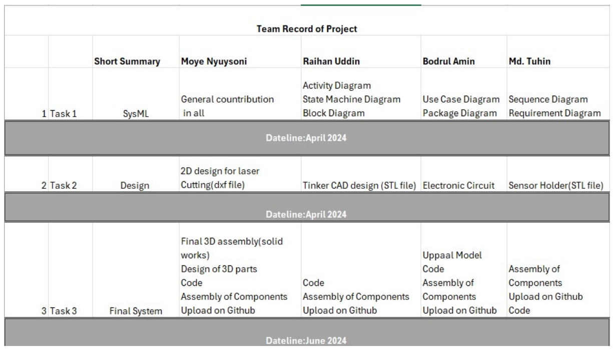
Fig. 28. CODE

Summary: Delays are used to control the timing of movements and sensor readings, ensuring smooth transitions and avoiding abrupt changes in speed or direction. The switch statement is utilized in the follow Line function to handle different sensor states efficiently and concisely. The binary representation of the sensor states allows for easy pattern matching and decision-making. Speed control is managed by the function analogWrite, which sets the speed of the motors to a precise number, either by stating it directly, as in the backward function, or using motorSpeed, a variable that can be adjusted dynamically. In conclusion, the robot successfully navigates along a predetermined path using infrared sensors to detect and follow the line. Additionally, equipped with an ultrasonic sensor, the robot can accurately measure distances and detect obstacles in its path.

Conclusion: In conclusion, the robot successfully navigates along a predetermined path using infrared sensors to detect and follow the line. Additionally, equipped with an ultrasonic sensor, the robot can accurately measure distances and detect obstacles in its path.

Avoid Red Obstacles and Park When It Sees Blue: Utilizing the TCS3200 color sensor, the robot distinguishes between red and blue obstacles. When encountering a red obstacle, the robot skillfully maneuvers to avoid it. Conversely, when it detects a blue obstacle, the robot efficiently parks itself. This final prototype demonstrates the integration and functionality of all designed components, achieving the project's objectives effectively.

Individual Contribution:



Team Record of Project

		Short Summary	Moye Nyuysoni	Raihan Uddin	Bodrul Amin	Md. Tuhin
1	Task 1	SysML	General countribution in all	Activity Diagram State Machine Diagram Block Diagram	Use Case Diagram Package Diagram	Sequence Diagram Requirement Diagram
Dateline:April 2024						
2	Task 2	Design	2D design for laser Cutting(dxf file)	Tinker CAD design (STL file)	Electronic Circuit	Sensor Holder(STL file)
Dateline:April 2024						
3	Task 3	Final System	Final 3D assembly(solid works) Design of 3D parts Code Assembly of Components Upload on Github	Code Assembly of Components Upload on Github	Uppaal Model Code Assembly of Components Upload on Github	Assembly of Components Upload on Github Code
Dateline:June 2024						

Fig. 29. Task Distribution

REFERENCES

[1] https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf

[2] https://uppaal.org/

[3] https://components101.com/sensors/ir-sensor-module

[4] https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf