

# **The University of Texas Rio Grande Valley**

College of Engineering and Computer Science

Department of Electrical and Computer Engineering

## **Senior Design II**

### **Final Report**

## **Project: An Educational Robot to Foster Student Engagement in K-12 Math**

Denisse Galvan, BSCE

Aaron Jackson, BSCE

Jesus E Ríos Jr, BSEE

Juan Vega, BSEE

**Advisor:** Dr. M. Ben Ghalia




May 2025

Edinburg, TX (USA)

## CERTIFICATE

This is to certify that, Denisse Galvan, Aaron Jackson, Jesus Rios, Juan Vega has been working under my supervision on the project titled “An Educational Robot to Foster Student Engagement in K-12 Math” towards the completion of his Senior Design project. The work satisfies the requirements of the Senior Design I/II (ELEE 4361/CMPE 4373, ELEE 4362/CMPE 4374) course.

Advisor Dr. M. Ben Ghalia Signature  Date: May 15, 2025

## **Technical Abstract**

This project develops a line-following autonomous educational robot to teach K-12 students in learning motion concepts such as average speed, distance, and time. This system will work by interacting with an Android mobile application that will use Bluetooth communication with an ESP32 microcontroller. The application will send speed commands which will control the robot's speed. The robot integrates sensors and motors to follow preset paths while allowing students to control speed parameters and collect real-time performance data. It encourages hands-on exploration of the relationship between speed, distance, time, and the robot create direct connections to math and science principles. Students can measure, analyze, and predict outcomes which foster critical thinking skills. This project bridges the gap between robotics and math/science education, offering an innovative and practical platform to engage students and enhance their comprehension of fundamental motion principles.

## **Non-Technical Abstract**

This project aims to create a user-friendly educational robot that helps students in K-12 learn Basic math/science concepts, such as average speed and distance, in a fun and interactive way. The line-following robot moves at speed which is controlled by the educator through a smartphone app, allowing students to calculate speed through hands-on experiments created by the educator. This innovative approach combines technology with education to improve classroom learning experiences, helping students grasp more complex topics. This robot would not only enhance classroom performance but build curiosity and excitement about STEM fields. Being able to inspire students to explore science and math in a hands-on and meaningful way.

## Table of Contents

1. Introduction .....	1
2. Problem Statement .....	2
2.1. Need Statement .....	2
2.2. Objective Statement .....	2
2.3. Background and Literature Search .....	3
3. System Requirements Specifications .....	4
3.1. Marketing & Engineering Requirements .....	4
3.2. Objective Tree .....	6
3.3. Realistic Design Constraints .....	7
4. Project Design .....	9
4.1. Hardware Research .....	9
4.1.1. Motor Selection .....	9
4.1.2. Motor Driver Selection .....	10
4.1.3. Sensor Selection .....	10
4.1.4. Power System .....	11
4.1.5. Display Selection .....	11
4.1.6. Microcontroller Selection .....	12
4.1.7. Mechanical Hardware .....	12
4.2. Prototype of Educational Robot .....	13
4.2.1. Motor Driver Evaluation and Replacement .....	14
4.2.2. QTR Sensor Application .....	17
4.3. App Development Process .....	21

4.3.1. Speed Data Exchange Protocol .....	23
4.3.2. Speed Transmission Test Protocol .....	24
4.3.3. Bluetooth Integration and Device Discovery .....	24
4.4. ESP32 Microcontroller Overview .....	27
4.4.1. System Initialization and Loop structure .....	29
4.4.1.1. Setup Phase .....	29
4.4.1.2. Main Loop Behavior .....	30
4.4.2. Encoder Feedback and Speed Measurement .....	33
4.4.2.1. Interrupt Handling .....	33
4.4.2.2. Speed Calculation .....	33
4.4.2.3. Average Speed Logging .....	34
4.4.3. PWM Motor Control and PI Feedback Loop .....	35
4.4.3.1. PWM Configuration .....	35
4.4.3.2. PI Controller Implementation .....	35
4.4.3.3. Output Adjustment and Application .....	36
4.4.4. Bluetooth Low Energy (BLE) Communication .....	37
4.4.4.1. Receiving Speed commands via BLE .....	37
4.4.4.2. Motor Control with PI Feedback .....	38
4.4.5. Line Following with Sensor Correction .....	38
4.4.5.1. Sensor Configuration and Calibration .....	39
4.4.5.2. Line Position Detection .....	39
4.4.5.3. Trajectory correction .....	41
4.4.6. Post-Run LCD Display Output .....	41

4.4.6.1.	LCD Integration .....	42
4.4.6.2.	Post-Run Data Display .....	42
4.4.6.3.	Educational Value .....	43
4.4.7.	App-Based Real-time Feedback and Telemetry .....	43
4.4.7.1.	Live Data Transmission .....	43
4.4.7.2.	Educational Value .....	44
4.5.	Block Diagram .....	45
4.6.	Flowchart .....	47
4.7.	Setbacks .....	49
5.	Gantt Chart – Project Timeline .....	50
5.1.	Initial Gantt Chart .....	50
5.2.	Prototype Final Gantt Chart .....	51
5.3.	Final Gantt Chart .....	52
5.4.	Financial Planning .....	53
5.5.	Original Budget Plan .....	54
5.6.	Final Budget Plan .....	55
5.7.	Project Cost for K-12 Educational Robot .....	55
6.	Final System Design and Test Results .....	56
6.1.	Prototype Final Design .....	56
6.2.	Improvement & Corrections .....	58
6.3.	Final Design .....	59
6.4.	Results .....	62
6.5.	Statistical Analysis .....	66

6.6. SpeediBot Manual .....	69
6.7. Completed Application .....	80
7. Project Criteria .....	81
7.1. Economic .....	81
7.2. Health & Safety .....	81
8. Project Summary .....	82
9. References .....	84
10. Appendix .....	86

## List of Figures

Figure 1: Objective Tree .....	6
Figure 2: Learning Math with SpeediBot .....	8
Figure 3: Fall Prototype .....	9
Figure 4: Voltage Output with L298N Motor Driver .....	15
Figure 5: Voltage Output with Dual VNH5019 Motor Driver .....	16
Figure 6: Error profile of QTR sensor array .....	19
Figure 7: QTR sensor .....	20
Figure 8: Initial App Layout Design .....	22
Figure 9: Initial Device Layout .....	22
Figure 10: Active App Session .....	26
Figure 11: Esp32-s3-wroom-1 Microcontroller .....	27
Figure 12: A Setup() initialization .....	29
Figure 13: B-Loop() Main Loop behavior .....	31
Figure 14: Interrupt Handling for encoder .....	32
Figure 15: A PWM settings .....	34
Figure 16: B PI control computation .....	35



Figure 17: C PWM adjusting with respect to PI .....	36
Figure 18: BLE Characteristics Class Instantiation .....	37
Figure 19: A QTR Sensor Configuration and Calibration .....	38
Figure 20: B line Position Detection Protocol .....	39
Figure 21: C Line Error Logic .....	40
Figure 22: D Corrections computation and PWM Adjustment .....	40
Figure 23: LCD Instantiation .....	41
Figure 24: Data Telemetry transmission to the App .....	43
Figure 25: Original Block Diagram .....	44
Figure 26: Final Block Diagram .....	45
Figure 27: Original Flow Chart .....	46
Figure 28: Final Flow Chart .....	47
Figure 29: Front Chassis .....	56
Figure 30: Rear Chassis .....	57
Figure 31: Mounting brackets chassis .....	58
Figure 32: Middle Chassis .....	59
Figure 33: Top Cover .....	60
Figure 34: QTR mount .....	60

Figure 35: 2 meters Trial .....	63
Figure 36: 3 meters Trial .....	64
Figure 37: Final Design of Splash Screen .....	80
Figure 38: Final Design of Control Screen .....	80
Figure 39: SpeediBot app control screen code .....	88
Figure 40: Bluetooth Communication code .....	92
Figure 41: Interior of SpeediBot .....	93
Figure 42: Exterior of SpeediBot .....	93

## **List of Tables**

Table 1: Pairwise Comparison Matrix .....	4
Table 2: Marketing & Engineering Requirements .....	5
Table 3: Voltage Output with L298N Motor Driver .....	15
Table 4: Voltage Output with Dual VNH5019 Motor Driver .....	16
Table 5: Measurements of the sensors moving to the right of the black line in increments of 2mm .....	17
Table 6: Measurements of the sensors moving to the left of the black line in increments of 2mm .....	18
Table 7: sensors and their assigned weight the sensors on the left are negative while the sensors on the right are positive .....	20
Table 8: Initial Gantt Chart .....	49
Table 9: Prototype Final Gantt .....	50
Table 10: Final Gantt Chart .....	51
Table 11: Original Budget Plan .....	54
Table 12: Final Budget Plan .....	54
Table 13: Project Cost for the K-12 Educational Robot .....	55
Table 14: Trial 1 Results .....	62
Table 15: Trial 2 Results .....	64

Table 16: trial 3 Results .....	65
---------------------------------	----

## 1. Introduction

According to the National Assessment of Educational (NAEP), many K-12 students struggle to understand basic motion concepts, such as average speed and distance traveled. To address this gap, educational robots are becoming increasingly introduced into classrooms as interactive learning tools. Research has shown that robotics engages students in the classroom and can enhance the quality of instruction and learning. Although many different education robot platforms exist such as LEGO Mindstorms, VEX robotics, and mbot. These platforms fall short of providing connections to specific physics concepts like speed, distance, and time.  $Speed = \frac{Distance}{Time}$

To address these limitations, we are designing and building a robot specifically tailored to accurately measure and demonstrate speed, thereby helping students and teachers explore fundamental physics concepts. Our robot will feature programmable functionality via a Bluetooth app, enabling users to input a target speed. Once the speed is set, the robot will travel between two fixed points, and its actual speed will be displayed on an integrated LCD screen. The inclusion of quad encoders in the robot's motors and an QTR Reflectance sensor for line tracking ensures precise measurements of time and distance, which are critical for accurately calculating speed.

By allowing students to visualize and compare input speed versus measured speed, the robot creates an interactive learning experience that makes abstract physics concepts tangible. Teachers can use the robot to demonstrate real-world applications of the speed formula ( $Speed = \frac{Distance}{Time}$ )

helping students understand the relationship between these variables more effectively. Additionally, the robot enables hands-on activities where students can calculate and predict speeds, enhancing engagement and reinforcing learning. This approach not only makes physics more accessible but also fosters critical thinking and problem-solving skills. The robot serves as an intuitive tool for bridging the gap between theoretical knowledge and practical application, ultimately improving students' understanding of motion.

## **2. Problem Statement**

### **2.1. Need Statement**

Students struggle to understand concepts like average speed when taught only through textbooks and lectures. Many students lack affordable, hands-on tools that make these concepts concrete and engaging. Our line-following robot fills this gap by letting students see speed calculations happen in real-time, experiment with different parameters, connect math formulas to physical movement and learn through direct observation. Our robot provides visual feedback through both its onboard display and a smartphone app, making the learning experience accessible and interactive. This approach helps students of various learning styles better grasp motion concepts that can cause confusion in K-12 science and math education.

### **2.2 Objective Statement**

Design and build a line-following autonomous educational robot to help K-12 students learn how to calculate average speed in a hands-on manner. This system will demonstrate real-time calculation of average speed as it navigates along predetermined paths using reflectance sensors. The robot can move autonomously using a 2-wheel differential drive chassis controlled by an ESP32 microcontroller, allowing wireless operations via Bluetooth communication with a

smartphone application. This design will enable educators to input their desired speed parameters, with the robot autonomously maintaining and measuring that speed while following line patterns. Real-time performance data will be displayed simultaneously on an integrated LCD screen and the mobile interface, providing immediate feedback and visualization of motion concepts.

## **2.3 Background and Literature Search**

Educational robots have proven to be effective tools in classrooms, as they foster student engagement, critical thinking, and problem-solving skills. Research highlights that robotics encourages active participation, enhances understanding of STEM topics, and improves the quality of learning experiences in K-12 settings (Johnson et al., 2020; Paper, 1980). Existing platforms like LEGO Mindstorms, VEX Robotics, and mBot are designed to introduce students to robotics and programming but do not focus explicitly on demonstrating or calculating physical motion parameters such as speed, distance, and time. For example, LEGO Mindstorms provides versatile programming capabilities and sensor integration, but its emphasis is on general robotics challenges rather than specific physics applications (Lego Education, 2022). Similarly, VEX Robotics and mBot encourage creativity and problem-solving in robotics but lack tailored functionality to demonstrate kinematics or dynamics principles effectively (VEX Robotics, 2022; Makeblock, 2022). These limitations indicate a clear gap in the market for a robot explicitly designed for teaching motion principles. Our design builds upon the strengths of these existing platforms while addressing their limitations. By integrating speed-measurement capabilities and linking them to physics concepts, our robot provides a more focused educational tool. The use of accurate measurement systems, such as quad encoders and IR sensors, ensures the reliability of

the data, which is critical for effective teaching and learning. Additionally, the Bluetooth-controlled app interface enhances the robot's accessibility, making it user-friendly for both teachers and students. This project contributes to the growing field of educational robotics by introducing a novel platform that bridges robotics and physics education, offering a solution to a well-documented gap in current teaching tools.

### 3. System Requirements Specifications

#### 3.1. Marketing & Engineering Requirements

The marketing requirements of the line-following robot are shown below:

- 1) **User Interface:** The system should have an intuitive and engaging user interface for the users.
- 2) **Accuracy and Reliability:** The robot should accurately report measured speeds and be reliable.
- 3) **Educational Value:** The robot should have the ability to demonstrate speed concepts to students.
- 4) **Hardware Functionality:** The robot's components should work together to achieve the desired performance.

**Table 1: Pairwise Comparison Matrix**

Criteria	UI	Accuracy	Educational	Hardware
User Interface	1	1/3	3	1/2
Accuracy and Reliability	3	1	5	2
Educational Value	1/3	1/5	1	1/4
Hardware Functionality	2	1/2	4	1



**Table 2: Marketing & Engineering Requirements**

<b>Marketing Requirements</b>	<b>Engineering requirements</b>	<b>Justification</b>
The system should have an intuitive and engaging user interface for the users.	The system should display data on an LCD screen. Interface should respond quickly to all inputs. The app should have 2 ways to visualize the speed data. Bluetooth connections shall be 99% reliable within 10m range.	Large text ensures classroom readability. Fast response prevents user frustration. Multiple visualization methods support varied learning approaches. Reliable wireless maintains engagement during demonstrations.
The robot should accurately report measured speeds and be reliable.	Sensors must measure accurate speed. Motors must be at a speed close to the target value.	Precise motor control ensures consistent demonstrations for clear learning outcomes. Sensor accuracy provides reliable data for student calculations.
The robot should have the ability to demonstrate speed concepts to students.	The speed calculation method must be visibly displayed screen. System must compare actual vs. theoretical speed in real-time. Data must be presented in both numerical and graphical formats.	Visual displays make abstract formulas concrete. Side-by-side comparisons show theory in practice. Adjustable parameters let students learn through hands-on testing.
The robot's components should work together to achieve the desired performance.	The system must integrate sensors, motors, and controls. Power distribution must support all components at full load simultaneously. Components must withstand typical classroom handling for 1+ years. Robot chassis must protect internal components.	Fast communication between components ensures a responsive line-following. A reliable power system prevents performance issues during demonstrations. Classroom- appropriate durability prevents maintenance disruptions during academic year.

### 3.2. Objective Tree

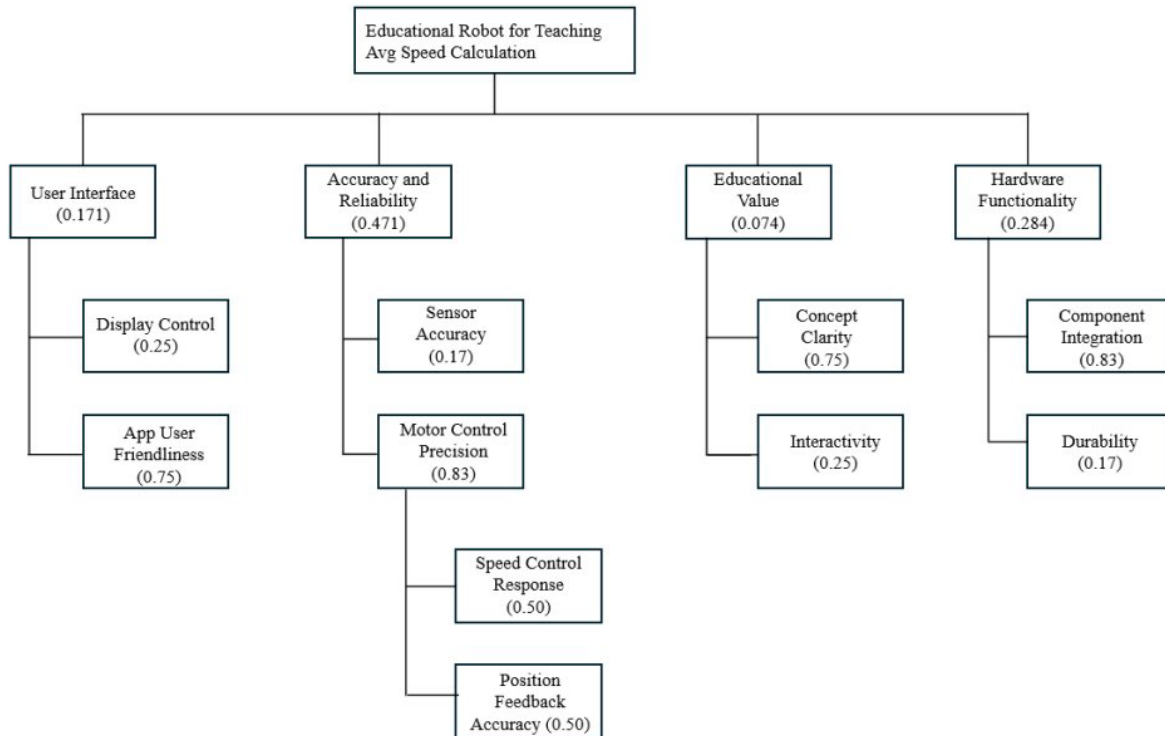


Figure 1 Objective Tree

This diagram represents a structured evaluation framework for an educational robot designed to teach average speed calculation. The framework is divided into four primary categories: User Interface, Accuracy and Reliability, Educational Value, and Hardware Functionality, each assigned a specific weight indicating its importance. The User Interface (0.171) focuses on display control and app user-friendliness, with more emphasis on ease of use. Accuracy and Reliability (0.471), the most significant category, evaluates sensor accuracy and motor control precision, further breaking down into speed control response and position feedback accuracy. Educational Value (0.074) emphasizes concept clarity and interactivity to ensure the robot effectively conveys learning objectives. Lastly, Hardware Functionality (0.284)

assesses component integration and durability, ensuring robust performance. This hierarchy highlights the critical factors in designing and assessing robots for educational purposes.

### **3.3. Realistic Design Constraints**

The educational robot project is subject to various constraints spanning technical, economic, environmental, social, political, ethical, health and safety, manufacturability, sustainability, and legal considerations. Technically, the robot must be compact, lightweight, and durable, with seamlessly integrated components and accurate real-time speed and RPM monitoring. Economic constraints demand affordability for educational institutions while ensuring minimal maintenance costs and market competitiveness. Environmentally, robots should minimize energy consumption, operate quietly, and use sustainable materials. Socially, it must be accessible to all students and culturally sensitive, while politically, it must comply with educational policies and international standards if exported. Ethical constraints include ensuring safety, avoiding surveillance features, and using non-toxic materials. Health and safety considerations focus on child-safe designs, electrical protection, and physical stability. Manufacturability requires readily available components, simple assembly, and realistic tolerances for cost-effective production. Sustainability means a long operational lifespan, repairability, and avoidance of obsolete parts quickly. Legally, the design must respect patent laws, adhere to educational product regulations, and ensure data privacy if any information is collected. These constraints collectively guide the project to create a safe, reliable, and impactful educational tool.

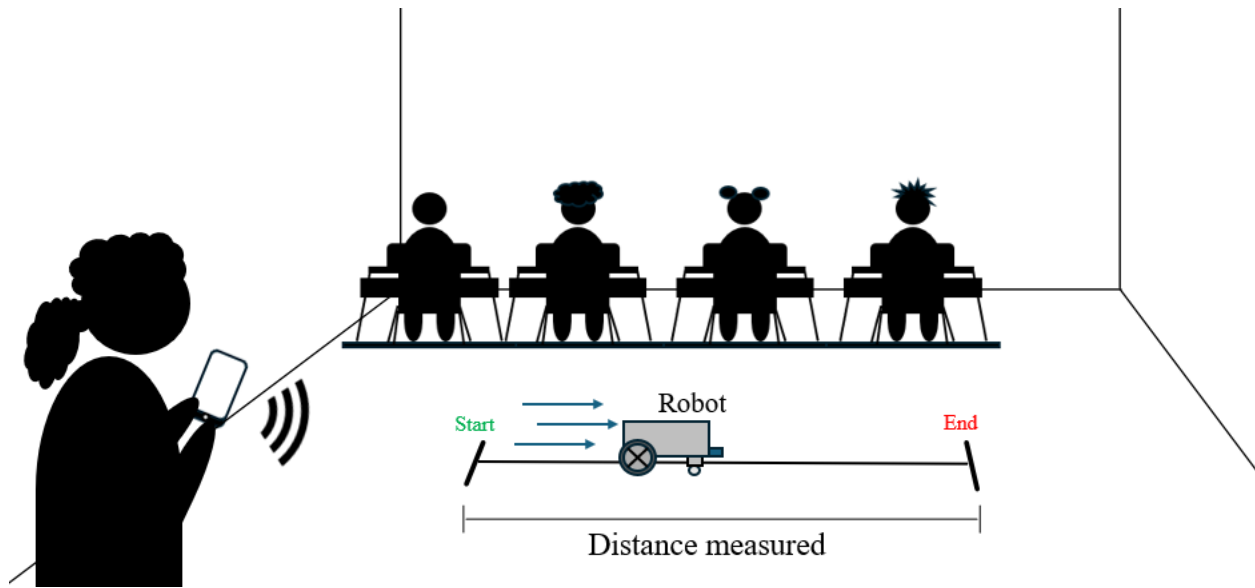
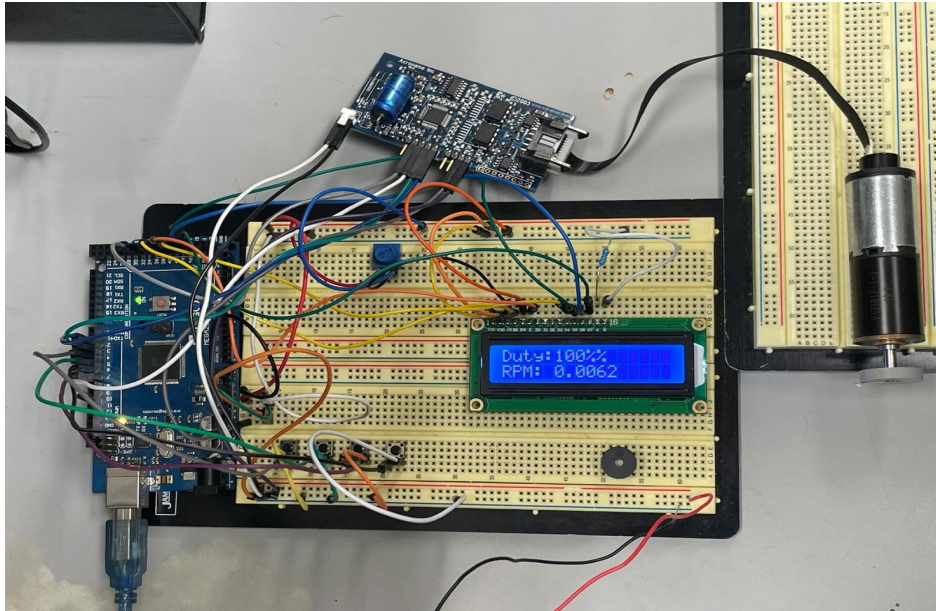


Figure 2: Learning Math with SpeediBot

In the figure above is a visual representation illustrates how SpeediBot is used as an educational tool within a classroom setting. The teacher (left) uses a smartphone to wirelessly send speed commands to the robot via Bluetooth. As SpeediBot travels across a measured distance, students watch closely, time the robot's movement, and calculate its average speed. This hands-on activity allows students to explore STEM concepts such as motion, distance, time and speed in an engaging and interactive way.

## Project Design

### 4.1 Hardware Research



*Figure 3: Fall Prototype*

Selection of hardware components was made to ensure precise control, reliable measurement, and ease of user interaction. This section details the rationale behind the selection of each major hardware component and how it contributes to the functionality of the system.

#### 4.1.1 Motor Selection

Component: 12V DC Motors with 10:1 Gear Ratio and Quadrature Encoders (64 CPR, dual channel)

Purpose and Contribution:

The selected 12V DC motors are geared for higher torque output and stable speed, making them ideal for controlled movement along a predefined path. The integrated quadrature encoders provide 64 counts per revolution when both channels are used, enabling precise measurement of

motor rotation. This high-resolution feedback is essential for calculating instantaneous RPM and, subsequently, the robot's average speed over a given distance. The ability to detect direction through the encoder channels also supports advanced control techniques such as PI (Proportional-Integral) control.

#### **4.1.2 Motor Driver Selection**

Component: Dual VNH5019 Motor Driver

Purpose and Contribution:

The Dual VNH5019 motor driver is responsible for delivering sufficient current to the 12V DC motors while enabling directional control and speed regulation through PWM signals. It supports logic-level interfacing at 3.3V, making it compatible with the ESP32-S3 microcontroller. Its integrated current sensing and robust output stage allow for reliable control of motor torque and speed, which is vital for maintaining consistent motion. This driver plays a central role in implementing closed-loop motor control, where real-time speed adjustments are made based on encoder feedback to match the user-specified speed input.

#### **4.1.3 Sensor Selection**

Component: QTR-MD-05A Reflectance Sensor Array (5-Channel, Analog Output)

Purpose and Contribution:

This five-channel analog reflectance sensor is used for detecting and following a black line on a white surface. Its analog output provides more precise reflectance readings compared to digital-

only sensors, allowing for smoother and more accurate line tracking. Accurate line following is critical to ensure that the robot travels along a known, consistent path, which directly influences the reliability of the distance measurement. Consistent path following is essential for accurately determining the average speed, as deviations or erratic motion could skew the results.

#### **4.1.4 Power System**

Components: 12V LiPo Battery with 3.3V and 5V Buck Converters

Purpose and Contribution:

The power system is based on a 12V LiPo battery, which serves as the primary power source for both the motors and electronic components. To ensure safe and efficient operation, two DC-DC buck converters are used to step down the voltage to appropriate levels. The first converter outputs 3.3V and powers the ESP32-S3 microcontroller, the motor encoders, the QTR sensor array, and also provides the logic-level supply for the VNH5019 motor driver. This ensures all logic-level components receive a clean and regulated voltage, promoting system stability and preventing damage to sensitive electronics. The second buck converter outputs 5V and is dedicated solely to powering the 16x2 LCD. The overall power system architecture ensures that each subsystem receives the appropriate voltage level, enhancing reliability and supporting uninterrupted operation of the robot.

#### **4.1.5 Display Selection**

Component: 16x2 Character LCD with I2C Interface

#### Purpose and Contribution:

The 16x2 character LCD with I2C interface is used to provide clear and immediate feedback to the user by displaying the robot's final average speed at the end of its run. Since it operates at 5V, it is powered independently through a dedicated 5V buck converter. The use of an I2C interface minimizes the number of GPIO pins required on the ESP32-S3, leaving more resources available for motor control, sensor input, and Bluetooth communication. Displaying only the average speed simplifies the interface and keeps the user focused on the educational objective of understanding motion and speed measurement. The LCD thus enhances user interaction and supports the system's role as a learning tool.

#### **4.1.6 Microcontroller Selection**

Component: ESP32-S3-DevKitC Microcontroller

#### Purpose and Contribution:

The ESP32-S3 serves as the central controller of the robot. It processes encoder signals to calculate motor speed and distance traveled, implements the PI control algorithm to regulate speed via PWM, and handles line-following logic using sensor data. The ESP32-S3 also manages Bluetooth Low Energy (BLE) communication, allowing users to input a target speed from a mobile device and receive performance feedback. Its ability to handle multiple tasks in real time, along with its 3.3V logic compatibility and built-in BLE, makes it a powerful and efficient choice for this application.

#### **4.1.7 Mechanical Hardware**



Components: Custom 3D-Printed Chassis with 63.5 mm Diameter Wheels and Front Castor Wheel

#### Purpose and Contribution:

The robot's mechanical structure consists of a custom 3D-printed chassis designed to securely mount all electronic and mechanical components, including the motors, sensor array, microcontroller, and battery. This customized design ensures optimal placement of each component, aiding in weight distribution, structural integrity, and overall balance. The drive system uses two wheels with a diameter of 63.5 mm, which directly affects the calculation of linear speed from motor RPM. Accurate wheel sizing is critical for converting rotational data from the encoders into meaningful distance and speed metrics. A front-mounted castor wheel provides additional stability without interfering with the robot's maneuverability, ensuring smooth movement and consistent contact with the surface. Together, the wheels and chassis support precise, stable travel along the predefined path, which is essential for reliable speed measurement.

#### **Prototype of Educational Robot**

The prototype of the Educational Robot, shown above in Error! Reference sources not found, which was designed throughout the previous semester (Fall 2024), consisted of a single motor, whose speed was controlled directly by 4 input button. The user will click on its desire

speed and will navigate through the LCD. The first message would be displayed at the speed that the user desired to click on. After the motor will start accelerating at the speed that was chosen. The second line of the LCD screen it will display the RPM of the desired speed that the user had chosen from the beginning.

#### 4.2.1 Motor Driver Evaluation and Replacement

One of the most significant design challenges in the development of our educational robot was achieving consistent and reliable motor control. At the beginning of the project, we selected the L298N motor driver due to its affordability and widespread use in basic robotics applications. However, as testing progressed, it became clear that the L298N had major performance limitations that directly impacted our ability to implement accurate speed measurement and control.

To evaluate the L298N, we conducted voltage measurements across a range of PWM duty cycles for both motors. The results are shown in Table 1 below. At low duty cycles (10%–30%), the output voltages were extremely low—well under 1V—causing the motors to remain stationary. As the duty cycle increased, voltage output improved, but the readings for Motor A and Motor B remained inconsistent, especially between 40% and 70%. These inconsistencies caused the robot to drift during motion and made precise encoder readings unreliable, which in turn made PID control implementation ineffective.

Duty Cycle	PWM value	Motor A Voltage	Motor B Voltage
10%	25.5	137.08mV	104.34mV
20%	51	131.7mV	144.1mV
30%	76.5	270mV	182mV

40%	102	2.8V	2.3V
50%	127.5	4.66V	4.16V
60%	153	5.6V	5.2V
70%	178.5	5.6V	5.2V
80%	204	6.3V	6.01V
90%	229	8.1V	7.79V
100%	254	9.36V	9.31V

Table 3: Voltage Output with L298N Motor Driver

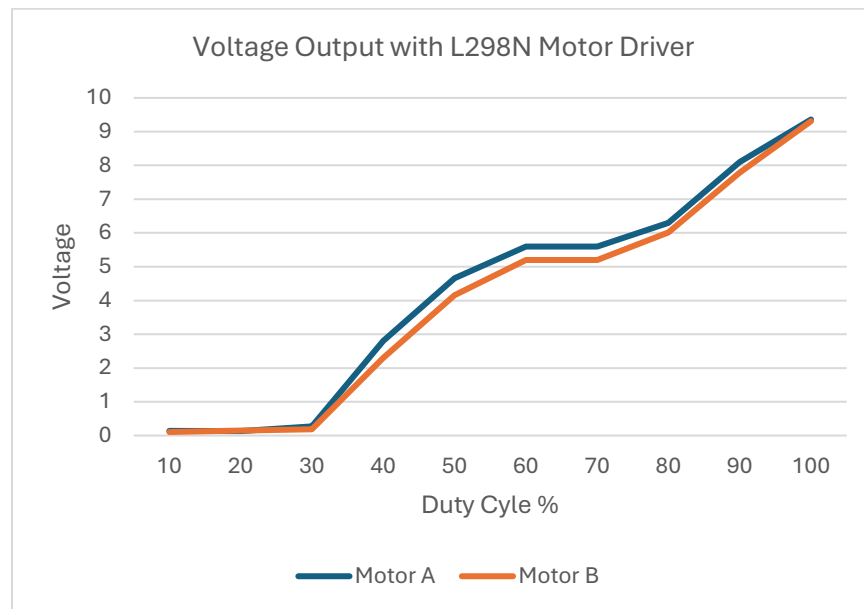


Figure 4 Voltage Output with L298N Motor Driver

To address this issue, we switched to the dual VN5019 motor driver, which provides significantly higher current handling capability, lower internal resistance, and more precise PWM control. After replacing the L298N, we ran the same voltage tests to measure performance. The results are shown in Table 2.

The VNH5019 driver delivered much more consistent and usable voltage across the full range of PWM duty cycles. At 10%, both motors received nearly 1V, which was sufficient to begin turning—an improvement over the L298N’s non-responsive low-range behavior. As duty cycles increased, the voltages rose smoothly and remained well-balanced between Motor A and Motor B. This level of precision is essential for maintaining straight-line motion, ensuring reliable encoder feedback, and enabling effective PID-based speed control.

Duty Cycle	PWM Value	Motor A Voltage	Motor B Voltage
10%	25.5	0.902V	0.88V
20%	51	2.78V	2.46V
30%	76.5	4.43	4.05V
40%	102	5.86V	5.45V
50%	127.5	6.75V	6.54V

Table 4: Voltage Output with Dual VNH5019 Motor Driver

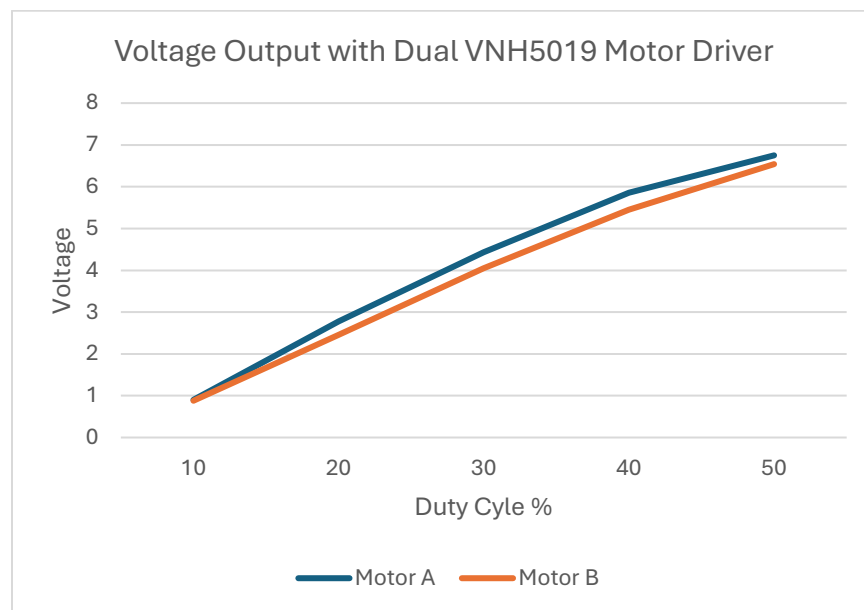


Figure 5 Voltage Output with Dual VNH5019 Motor Driver

The comparison between the two motor drivers clearly shows the advantage of the VNH5019. While the L298N struggled with low-voltage output and uneven power distribution, the VNH5019 consistently delivered higher and more balanced voltage levels. This directly improved motor responsiveness and allowed for reliable movement at all duty cycles. These improvements made it possible to fine-tune encoder readings, implement a stable PI/PID control system, and improve overall robot motion.

By resolving the voltage inconsistency and enabling better motor performance, this hardware change was instrumental in advancing the project's goal of delivering a functional and educational speed-measuring robot. The motor driver upgrade was not just a performance improvement, it was a necessary step to ensure the robot could meet its design requirements.

#### 4.2.2 QTR Sensor Application

Error Profile of the QTR sensor array:

Moving to the right	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5
0mm	0	182	651	0	0
2mm	0	487	615	0	0
4mm	0	631	404	0	0
6mm	0	690	40	0	0
8mm	292	664	0	0	0
10mm	605	594	0	0	0
12mm	674	392	0	0	0
14mm	699	56	0	0	0
16mm	639	0	0	0	0
18mm	572	0	0	0	0

20mm	370	0	0	0	0
22mm	91	0	0	0	0
24mm	0	0	0	0	0
26mm	0	0	0	0	0

Table 5:Measurements of the sensors moving to the right of the black line in increments of 2mm

Moving to the right	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5
0mm	0	222	717	0	0
2mm	0	0	733	192	0
4mm	0	0	643	453	0
6mm	0	0	542	610	0
8mm	0	0	136	733	99
10mm	0	0	0	734	613
12mm	0	0	0	614	811
14mm	0	0	0	475	844
16mm	0	0	0	41	833
18mm	0	0	0	0	777
20mm	0	0	0	0	719
22mm	0	0	0	0	622
24mm	0	0	0	0	367
26mm	0	0	0	0	0

Table 6: Measurements of the sensors moving to the left of the black line in increments of 2mm

Started from the center sensor in the middle of the line (the black line is 18.2mm wide) and moved the robot to the right in increments of 2mm and measuring the outputs and repeating on the left side. Yielding the following error profile chart:

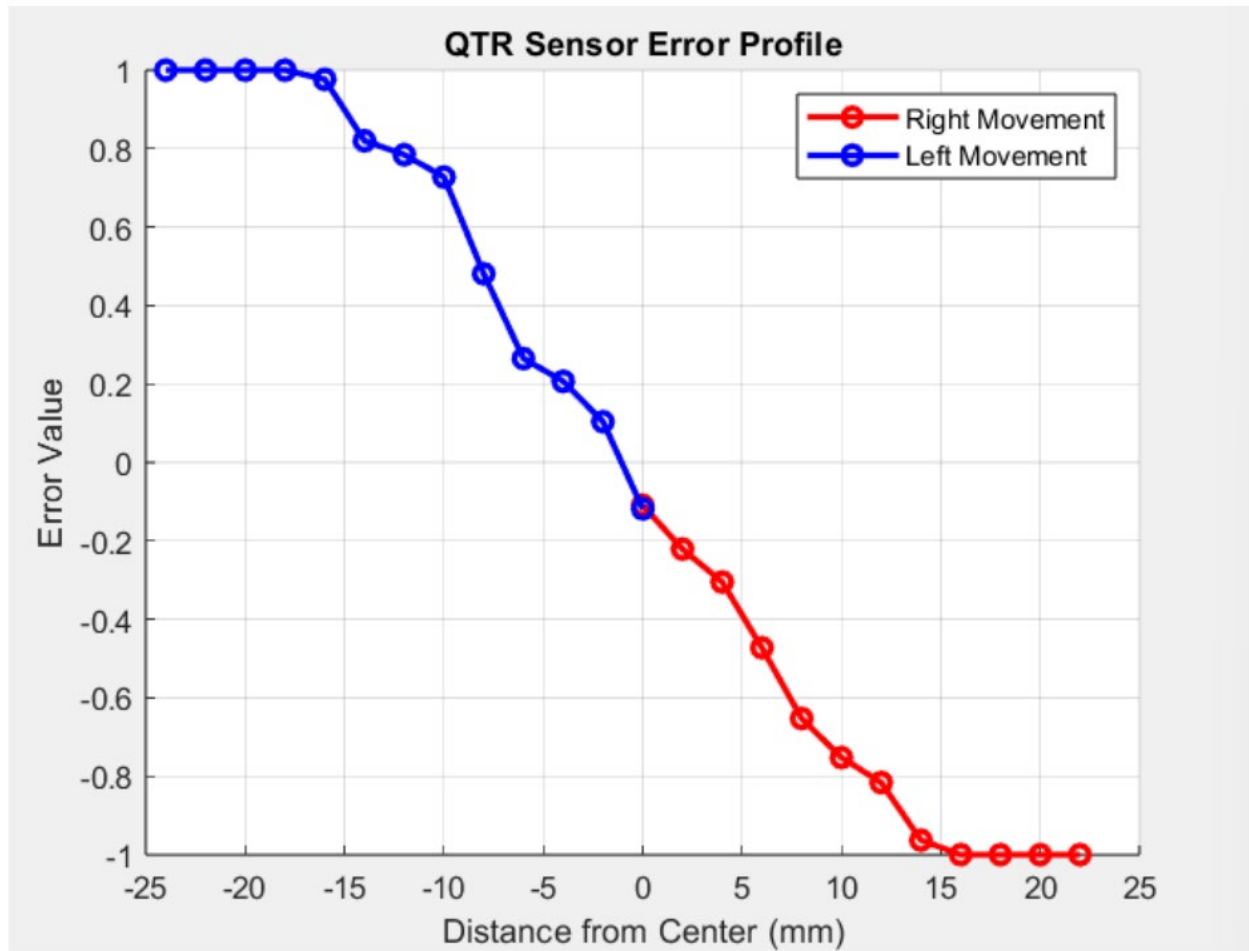


Figure 6 Error profile of QTR sensor array

$E = \frac{\sum(W_i \times S_i)}{\sum(S_i)}$  using weighted sum method to compute the error of the QTR sensor array of 5.

Where E= computed error (distance from center)

$W_i$ = weight assigned to each sensor

$S_i$ = sensor readings (0 to 1000)

Figure

Sensor Number	Weight $W_i$
Sensor 1	-2
Sensor 2	-1
Sensor 3	0
Sensor 4	+1
Sensor 5	+2

Table 7: sensors and their assigned weight the sensors on the left are negative while the sensors on the right are positive

As seen in Figure 6 error profile when the robot moves to the right of the line the sensors on the left side of the array activate yielding a negative error value which tells the robot to turn left and vice versa for when the robot moves to the left of the line.

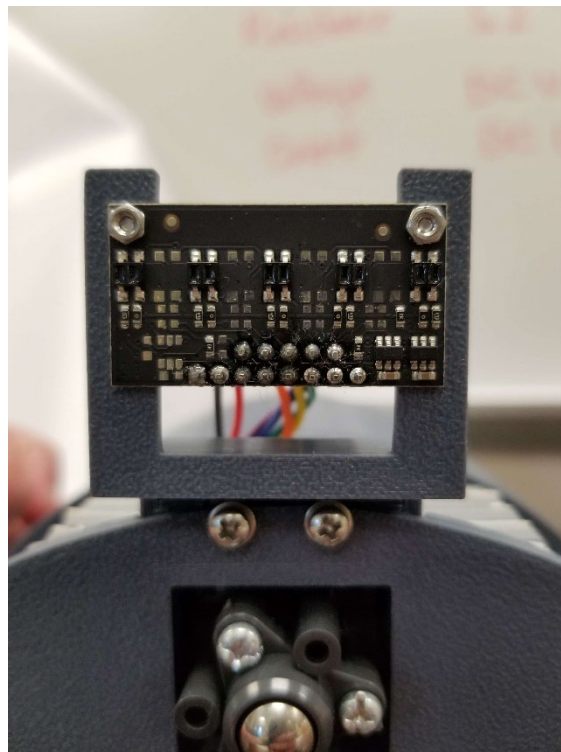


Figure 7 QTR sensor



The image shows the QTR reflectance sensor array mounted on the robot. It uses multiple infrared emitter-receiver pairs to detect surface reflectivity, helping the robot follow lines. The PCB is securely mounted, with soldered sensor connections and surface-mount components visible. Wires from the back connect the sensor outputs to the microcontroller for real-time line detection and navigation.

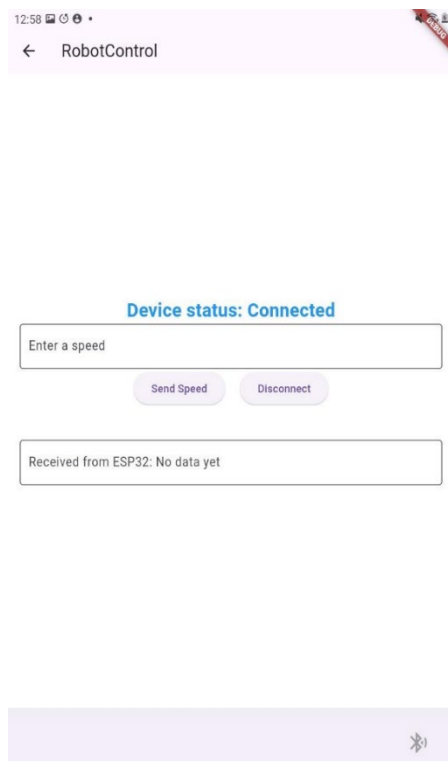
### **4.3. App Development Process:**

Before beginning our smartphone application, we researched about what framework and language was best for it. We chose Flutter as our framework because it enables cross-platform development with a single codebase, allowing our educational application to run on both Android and iOS devices without requiring separate development efforts for each platform. Dart, Flutter's programming language, offers strong typing and object-oriented features that helped us create a responsive and reliable essential for classroom use. For testing and implementation, we compiled the application in Android Studio and transferred the APK file directly to a physical Android smartphone via USB connection or WI-FI. This allowed us to test Bluetooth connectivity, interface responsiveness, and visualization instead of just relying on emulator-based testing.

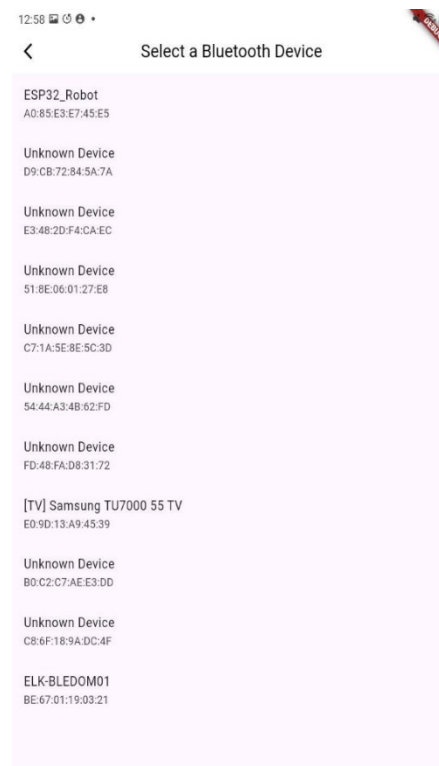
The application greets you with a Splash screen which has a duration of three seconds. After it would go to the speed control screen which consists of all the features needed. This screen includes the Bluetooth connection, the input text field and the text field that displays real-time feedback. For the Bluetooth connection, there is a scroll down menu which shows you if SpeediBot is available and filters out any other device, this menu is programmed to only look for SpeediBot and nothing else. It consists of a button that starts the scan again and would change if the device is connecting that will display "Device connected". Once the device is connected, there is the text field where the user can input the speed and send it. The application displays an

error message if the text-field is empty and the user is trying to click send or if the input is not a numerical value. It also includes a button to be able to stop the robot in case of a malfunction and a disconnect button to be able to disconnect the Bluetooth connectivity. Lastly, there is a text field, where it only shows you the real-time feedback that is coming from the microcontroller.

The backend programming for the application utilizes Bluetooth Low Energy (BLE) communication implemented through the Flutter library `flutter_blue_plus`. The library facilitates efficient wireless communication between the app and the ESP32 microcontroller embedded in the robot. The application is the replacement of physical control buttons with digital inputs through the mobile interface. Having a simple user interface facilitates the scanning and discovery of devices and communication between the microcontroller and the application. This allowed us to test the application and the microcontroller with ease, which you can see in the figures below.



*Figure 8. Initial App Layout Design*



*Figure 9. Initial Device Layout*

#### 4.3.1 Speed Data Exchange Protocol:

When the app's prototype was developed, it sent speed commands to the ESP32 microcontroller, but the inputs were limited. First, the application will retrieve the text from the text field controller, and it will attempt to parse the text as an integer value. It will validate the speed text field is not empty then the text can be successfully parsed as a numeric value. If the validation passes, then it will call a method from the Bluetooth Service to send the speed and if the validation fails, then it will display an error message using Snackbar. The way the application was set up to send the data to the microcontroller was to check if the target Characteristic which represents a Bluetooth characteristic is not null, this meant that a Bluetooth connection has been established. It will then convert the speed value into a string. The string is then converted to a list of byte codes using a command called codeUnits which converts each character to its UTF-8 code unit. The transmission happens with target Characteristic!. write () method this sends the bytes to the connected device. It then uses withoutResponse: true parameter, this sends the data without waiting for an acknowledgement. The speed data is sent as ASCII text instead of a binary value, the ESP32 receives these bytes and parses them back to numeric value.

However, even though the application successfully communicated with the microcontroller, and could send data. We encountered a limitation with the input values. Since the input was parsed as an integer value then the system could only accept whole numbers, this prevented us from inputting decimal values for more precise speed control. To ensure precise speed control, instead of parsing the text as an integer value, it was changed to be parsed a

double with a format of two decimal places. Once this change was made, we were successfully able to send different inputs, for example, 0.2, 0.4.

After finishing the previous implementation, we focused on receiving data from the microcontroller. To be able to do this implementation, we created a text field that will display real-life feedback. In our Bluetooth service, we set up a notification listener for Bluetooth characteristics. It will check if the Bluetooth connection supports notifications, then enables this feature to listen for incoming data. When the ESP32 sends information, the system will decode the received bytes into a readable text using UTF-8 encoding, logs the receive data and updates the app's display with the latest information. This implementation successfully sets up a two-way data protocol, where the application can receive live data from the microcontroller without asking for information from the microcontroller.

#### **4.3.2 Speed Transmission Test Protocol:**

The minimalist application interface was developed to conduct tests on the microcontroller's capability to receive speed commands and accurately adjust the motors to match the specified input speed. The microcontroller can receive these commands and parse them back to a numerical value.

#### **4.3.3 Bluetooth Integration and Device Discovery**

Setting up wireless Bluetooth communication came with complications during the development. There were complications at both ends, the application and the microcontroller.

When we were setting up the Bluetooth service for both and testing them, the microcontroller was not receiving anything, and we thought the issue came from the application. After some research, we noticed that an error was happening in the microcontroller. When creating the BLE characteristics we had to use multiple parameters, and we were missing the `PROPERTY_WRITE_NR`. Its function is for the application to be able to write values without requiring acknowledgement. We also implemented a function that had the same task in the application backend which was previously discussed. After adding this property, we were able to write to the microcontroller and receive data from it as well. Another complication was the display of available devices for the app, it will scan for any available devices and print them all, which made it difficult to find the robot. To fix this issue we filtered out any other devices that were not named SpeediBot.

The other issue was how we were going to display the available devices. At first, we would navigate to another page and look for SpeediBot, but we wanted to make it simpler. To do that we decided to get rid of that approach and add a dropdown menu to the control screen that will only scan for SpeediBot.



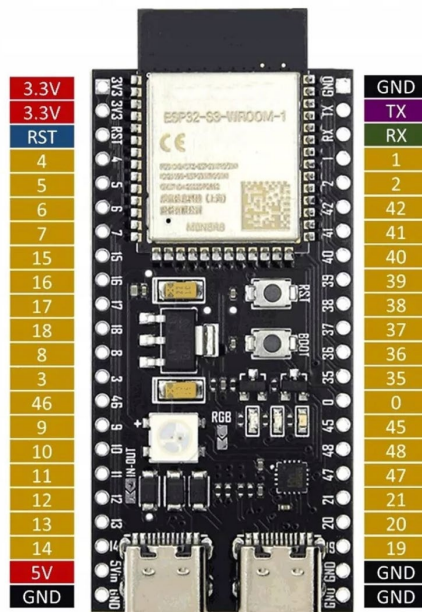
*Figure 10 Active App Session*

#### 4.4. ESP32 Microcontroller Overview

The Esp32-S3-DevKitC-1 microcontroller serves as the central processing unit for the robot, enabling real-time control, sensor data collection, and Bluetooth communication. Its powerful dual-core architecture, extensive GPIO options, and integrated wireless connectivity makes it ideal for educational robotics applications that require responsive interaction and flexibility.

The Esp32 was chosen for its ability to handle concurrent tasks such as motor speed regulation, sensor feedback processing, and bidirectional communication with a smartphone app via Bluetooth Low Energy (BLE). The hardware specifications that support these functions are outlined below:

- Processor: 32-bit Xtensa Dual-core @240MHz
- RAM: 512 KB SRAM
- Flash Memory: 8 MB
- Wireless Connectivity: Integrated Wi-Fi and Bluetooth (Classic + BLE)
- GPIO Pins: 45 GPIO pins
- PWM Channels: 8 independent PWM channels
- Communication Interfaces: UART, SPI, I2C, I2S, CAN
- Operating Voltage: 3.3V



*Figure 11 Esp32-s3-wroom-1 Microcontroller*

The microcontroller also supports advanced power management and interrupt handling, allowing it to manage real-time tasks such as motor adjustments and sensor reading without latency. Its BLE functionality is essential for connecting with the mobile app, which sends speed commands and receives data. There is a wide array of hardware peripherals and libraries available for Esp32

development. This with a combination of development using the Arduino framework accelerated our firmware development and fast testing.

Overall, the Esp32's processing power and integrated communication capabilities made it an ideal fit for this project's needs in terms of performance, cost-effectiveness, and scalability.

#### **4.4.1 System Initialization and Loop structure**

The firmware for the robot is written using the Arduino framework, targeting the Esp32s3-DevModule board configuration. To ensure compatibility with LCD display library (LiquidCrystal\_I2C), we used esp32 board package version 2.0.9, as later versions introduced changes that led to compilation and driver conflicts.

The firmware follows the standard Arduino structure, beginning with the setup() function, where all hardware and communication peripherals are initialized. The Loop() function then runs continuously to maintain real-time control of the robot's motion, speed regulation, and Bluetooth communication.

##### **4.4.1.1 Setup phase:**

In setup(), the following function performs all one-time initializations on startup:

- **Serial communication:** initialized at 115200 baud for debugging and serial monitoring.
- **QTR Sensor Calibration:** The 5-channel reflectance array is calibrated over 400 cycles to determine the white/black threshold levels for line following.
- **Motor and PWM initialization:** The two motors are controlled via the Esp32's hardware PWM using ledcSetup() and ledcAttachPin(). Directional pins and PWM outputs are set accordingly.



- **Encoder initialization:** The encoders are configured with internal pull-ups and assigned interrupts on signal changes to track wheel rotations.
- **BLE Setup:** The Esp32 starts a BLE server advertising as “SpeediBot”, with custom read/write/notify characteristics to communicate with the mobile application.

```
// ----- Setup -----
void setup() {

  Serial.begin(115200);
  // Initialize I2C with specified pins
  Wire.begin(SDA_PIN, SCL_PIN);
  // Pass Wire object to LCD (some versions do this automatically)
  lcd.init();          // Initialize the LCD
  lcd.backlight();
  Serial.println("Calibrating QTR sensors...");
  for (uint16_t i = 0; i < 400; i++) {
    qtr.calibrate();
    delay(5);
  }
  Serial.println("Calibration complete.");

  qtr.setTypeAnalog();
  qtr.setSensorPins((const uint8_t[]){4, 5, 6, 7, 8}, 5);
  qtr.setEmitterPin(2);

  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(STOP_LED, OUTPUT); digitalWrite(STOP_LED, LOW);

  ledcSetup(PWM_CHANNEL_A, PWM_FREQ, PWM_RESOLUTION);
  ledcSetup(PWM_CHANNEL_B, PWM_FREQ, PWM_RESOLUTION);
  ledcAttachPin(ENA, PWM_CHANNEL_A);
  ledcAttachPin(ENB, PWM_CHANNEL_B);

  pinMode(ENC1_A, INPUT_PULLUP); pinMode(ENC1_B, INPUT_PULLUP);
  pinMode(ENC2_A, INPUT_PULLUP); pinMode(ENC2_B, INPUT_PULLUP);
  for (uint8_t i = 0; i < NUM_SENSORS; i++) pinMode(sensorPins[i], INPUT);

  attachInterrupt(digitalPinToInterrupt(ENC1_A), encoder1ISR, CHANGE);
  attachInterrupt(digitalPinToInterrupt(ENC2_A), encoder2ISR, CHANGE);
  setupBLE();
  lastTime = millis();
  robotStartTime = millis();
}
```

Figure 12 A Setup() initialization

#### 4.4.1.2 Main Loop Behavior:

The loop() function executes continuously and handles:

4. **Stop-Line Detection:** if all five QTR sensors detect black, the robot stops, reports the average speed over the run, and exits early. A status LED is turned on to indicate termination.
5. **Safety Checks:** if Bluetooth is disconnected or a zero/negative speed is received, the motors stop immediately.
6. **Timed Control Loop (20ms):** every 20 milliseconds, the following actions occur:
  - a. **Encoder Sampling:** Calculates the RPM of both motors based on the number of encoder ticks in the last interval.
  - b. **Speed Conversion:** Converts RPM to linear speed in m/s.
  - c. **Average Speed Logging:** Accumulates speed data if the robot is in motion.
  - d. **QTR Reading:** Captures the current line position using both digital thresholding and analog line position estimation.
  - e. **Data Logging and Transmission:** Constructs a telemetry string containing target speed, motor speeds, RPMs, and line error, which is sent via BLE to the app.
  - f. **PI Speed Control:** A proportional-integral controller computes new PWM values for both motors to match target RPM.
  - g. **Line Correction:** A line error correction factor is applied to adjust motor outputs using a  $k_{line}$  gain multiplier.
  - h. **Motor Update:** Final PWM values are sent to the motor driver, and the last timestamp is updated.

This time-slice structure ensures reliable real-time operation, for smooth motor control, and responsive user feedback, all within a tight, repeatable control loop.

```

// ----- Loop -----
void loop() {

    bool allOnBlack = false;
    int dummyError;

    readLineSensors(dummyError, allOnBlack);

    if (allOnBlack) {
        Serial.println("STOP LINE DETECTED - Stopping motors.");
        stopMotors();
        digitalWrite(STOP_LED, HIGH);

        if (sampleCount > 0) {
            Serial.printf("Average Speed: %.2f m/s\n", speedSum / sampleCount);
        }

        targetSpeed = 0;
        targetRPM = 0;
        return; // Exit early to avoid processing the rest of the loop
    }

    if (!deviceConnected || targetSpeed <= 0) {
        stopMotors();
        digitalWrite(STOP_LED, LOW);
        return;
    }

    unsigned long currentTime = millis();
    if (currentTime - lastTime >= TIMER_INTERVAL) {
        float dt = (currentTime - lastTime) / 1000.0;
        rpm1 = abs((encoderCount1 / (float)CPR) * (60.0 / dt));
        rpm2 = abs((encoderCount2 / (float)CPR) * (60.0 / dt));
        encoderCount1 = encoderCount2 = 0;

        float speed1 = (rpm1 / 60.0) * (PI_F * WHEEL_DIAMETER);
        float speed2 = (rpm2 / 60.0) * (PI_F * WHEEL_DIAMETER);
        float avgSpeed = (speed1 + speed2) / 2.0;
        if (isMoving) { speedSum += avgSpeed; sampleCount++; }

        int lineError = 0;
        bool allOnBlack = false;
        readLineSensors(lineError, allOnBlack);

        // Read QTR line position and print sensor values
        uint16_t position = qtr.readLineBlack(sensorValues);
        Serial.print("Line Position: ");
        Serial.println(position);

        Serial.print("Sensor Values: ");
        for (uint8_t i = 0; i < 5; i++) {
            Serial.print(sensorValues[i]);
            Serial.print(' ');
        }
        Serial.println();
        //TIMER
        unsigned long elapsedTime = (millis() - robotStartTime); // in seconds
        Serial.print("Run Time: ");
        Serial.print(elapsedTime);
        Serial.println(" sec");

        String data = "Target: " + String(targetSpeed) + " m/s | M1: " +
            String(speed1) + " m/s, " + String(rpm1) + " RPM | M2: " +
            String(speed2) + " m/s, " + String(rpm2) + " RPM | LineError: " +
            String(lineError) + " Avg Speed:" + String(avgSpeed);
        Serial.println(data);
        if (deviceConnected) {
            pCharacteristic->setValue(data.c_str());
            pCharacteristic->notify();
        }

        float error1 = targetRPM - rpm1;
        float error2 = targetRPM - rpm2;
        integral_error1 += error1 * dt;
        integral_error2 += error2 * dt;
        //clamp integral to prevent windup
        if (rpm1 < 150 && rpm2 < 150) {
            integral_error1 = constrain(integral_error1, -600, 600);
            integral_error2 = constrain(integral_error2, -600, 600);
        }
        else {
            integral_error1 = constrain(integral_error1, -1000, 1000);
            integral_error2 = constrain(integral_error2, -1000, 1000);
        }
        pwm1 = Kp * error1 + Ki * integral_error1;
        pwm2 = Kp * error2 + Ki * integral_error2;
        float correction = k_line * lineError;
        updateMotorOutputs(correction);
        lastTime = currentTime;
    }
}

```

Figure 13 B Loop() Main Loop behavior

## 4.4.2 Encoder Feedback and Speed Measurement

To accurately regulate and monitor the robot's movement, the firmware uses quadrature encoders mounted on each motor shaft. These encoders produce two square wave signals (A and B) with a phase shift, allowing the system to detect both rotation direction and pulse count. The firmware uses interrupt-driven counting on the primary signal lines (ENC1\_A and ENC2\_A) and compares the state of the secondary lines (ENC1\_B and ENC2\_B) to determine direction.

### 4.4.2.1 Interrupt Handling

The encoder signals are processed using hardware interrupts configured on the rising/falling edge of channel A. Within each interrupt service routine (ISR), the direction of rotation is determined by comparing the A and B signal levels. This provides accurate counting regardless of motor direction:

```
// ----- Interrupts -----  
void IRAM_ATTR encoder1ISR() {  
    if (digitalRead(ENC1_A) == digitalRead(ENC1_B)) encoderCount1++;  
    else encoderCount1--;  
}
```

*Figure 14 interrupt Handling for encoder*

Each Motor's tick count is stored in a volatile variable to ensure safe access between interrupt and main loop contexts.

### 4.4.2.2 Speed Calculation

At regular intervals (every 20 ms), the encoder tick counts are used to calculate the motors rotational speed (RPM). The encoder counts per revolution (CPR) is set at 320 (initially set to 160 before reconfiguration for better accuracy), and the RPM is computed using the formula:

$$RPM = \left(\frac{Ticks}{CPR}\right) \times \left(\frac{60}{\Delta t}\right)$$

Where:

- Ticks = number of encoder counts in the last interval
- CPR = 320
- $\Delta t$  = time interval in seconds

The RPM is then converted to linear speed (m/s) using:

$$Speed \left(\frac{m}{s}\right) = \left(\frac{RPM}{60}\right) \times \pi \times D$$

Where D = 0.0635 m is the wheel diameter.

#### **4.4.2.3 Average Speed Logging**

During each interval, the firmware also logs the average linear speed for both motors for later reporting. This value is accumulated over time and used to compute the total average speed when the robot completes its run and encounters the stop line. This helps demonstrate the accuracy of speed regulation and provides real-time insight to students via the app and LCD.

#### **4.4.3 PWM Motor Control and PI Feedback Loop**

To control the speed of the robot's two DC motors, the firmware employs a Proportional-integral(PI) feedback loop that dynamically adjusts the motor output via hardware PWM (Pulse width Modulation) signals. This control strategy enables the robot to match its actual speed to the user-defined target speed in real-time, compensating for load changes, friction, or battery variation.

#### 4.4.3.1 PWM Configuration

The Esp32 supports multiple high-resolution PWM channels. Two channels are configured using the `ledcSetup()` and `ledcAttachPin()` functions:

```
// PWM Settings
#define PWM_CHANNEL_A 0
#define PWM_CHANNEL_B 1
#define PWM_FREQ 5000
#define PWM_RESOLUTION 8
```

*Figure 15 A PWM settings*

- PWM\_CHANNEL\_A and PWM\_CHANNEL\_B are assigned to ENA and ENB pins respectively.
- An 8-bit resolution allows PWM duty cycles from 0 to 255.
- A frequency of 5kHz provides smooth motor behavior.

#### 4.4.3.2 PI Controller Implementation

The firmware calculates the difference between the target RPM (from desired speed) and the measured RPM (from encoder data). This difference, or error, is used to compute the PWM output using a PI control equation:

$$PWM = K_p * error + K_i * \int error * dt$$

Where:

- $K_p = 0.45$  is the proportional gain
- $K_i = 0.13$  is the integral gain

- The integral term is accumulated over time to eliminate steady-state error

This computation is done independently for each motor every 20 milli seconds to prevent integral wind-up, the accumulated error is clamped depending on the motor's current speed:

```
// PI Controller
float pwm1 = 0.0;
float pwm2 = 0.0;
float integral_error1 = 0.0;
float integral_error2 = 0.0;
float Kp = 0.45;
float Ki;

if(rpm1 < 135 && rpm2 < 135){ //
Ki = 0.104;
float error1 = targetRPM - rpm1;
float error2 = targetRPM - rpm2;
integral_error1 += error1 * dt;
integral_error2 += error2 * dt;
integral_error1 = constrain(integral_error1, -600, 600);
integral_error2 = constrain(integral_error2, -600, 600);
pwm1 = Kp * error1 + Ki * integral_error1;
pwm2 = Kp * error2 + Ki * integral_error2;
}
else if(rpm1 > 181 && rpm2 > 181){
Kp = 0.6;
Ki = 0.16;
float error1 = targetRPM - rpm1;
float error2 = targetRPM - rpm2;
integral_error1 += error1 * dt;
integral_error2 += error2 * dt;
integral_error1 = constrain(integral_error1, -600,600);
integral_error2 = constrain(integral_error2, -600,600);
pwm1 = Kp * error1 + Ki * integral_error1;
pwm2 = Kp * error2 + Ki * integral_error2;
}
else{
Ki = 0.13;
float error1 = targetRPM - rpm1;
float error2 = targetRPM - rpm2;
integral_error1 += error1 * dt;
integral_error2 += error2 * dt;
integral_error1 = constrain(integral_error1, -1000,1000);
integral_error2 = constrain(integral_error2, -1000,1000);
pwm1 = Kp * error1 + Ki * integral_error1;
pwm2 = Kp * error2 + Ki * integral_error2;
}
```

Figure 16 B PI control computation

This ensures control stability at both low and high speeds.

#### 4.4.3.3 Output Adjustment and Application

The computed PWM values are modified by a line-following correction factor (4.3.5) and then written to the motor driver using `ledWrite()`:

```
// ----- Sensor & Motor Functions -----  
void updateMotorOutputs(float correction) {  
    int pwmVal1 = constrain((int)(pwm1 - correction), 0, 255);  
    int pwmVal2 = constrain((int)(pwm2 + correction), 0, 255);  
    ledcWrite(PWM_CHANNEL_A, pwmVal1);  
    ledcWrite(PWM_CHANNEL_B, pwmVal2);  
}
```

*Figure 17 C PWM Adjusting with respect to PI*

The direction pins (IN1-IN4) are set to ensure forward motion during normal operation. If the target speed is invalid, or the robot reaches the end of the line, the controller stops both motors and resets integral terms.

This feedback mechanism enables the robot to accurately follow a target speed while adapting in real-time to dynamic conditions—a core requirement for demonstrating average speed to students.

#### **4.4.4 Bluetooth Low Energy (BLE) Communication**

To enable real-time interaction between the robot and the user, the system uses Bluetooth Low Energy (BLE) communication between the ESP32 microcontroller and the mobile application. This Communication channel supports two-way data exchange, allowing speed commands to be sent to the robot and sensor feedback to be received on the mobile device.

##### **4.4.4.1 Receiving Speed commands via BLE**

The mobile application sends speed values as ASCII text strings to the ESP32. These values are received through a custom BLE characteristic defined using the BLE Characteristic class. Upon receiving new data, the system converts the string of characters into a floating-point number,



representing the target speed in meters per second. This value is then used to calculate the corresponding target RPM using the formula:

$$RPM = \frac{Speed \cdot 60}{\pi \cdot D}$$

Where  $D = 0.0635$  m is the wheel diameter. The target RPM is then used as the reference for the motor control system. Once a valid speed is received, the robot begins motion and initializes sampling for calculating average speed.

```
class MyCallbacks : public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        std::string value = std::string(pCharacteristic->getValue().c_str());
        if (!value.empty()) {
            targetSpeed = atof(value.c_str());
            targetRPM = (targetSpeed > 0) ? (targetSpeed * 60.0) / (PI_F * WHEEL_DIAMETER) : 0.0;
        }
    }
};
```

Figure 18 BLE Characteristics Class Instantiation

#### 4.4.4.2 Motor Control with PI Feedback:

The system uses a Proportional-Integral (PI) controller to regulate motor speed and maintain the target RPM. Quadrature encoders mounted on the motors provide feedback on rotational speed, which is converted to linear speed using wheel geometry. The controller updates PWM outputs every 100 milliseconds, adjusting for speed errors using the following control equation:

$$PWM = K_p \cdot error + K_i \cdot \int error \cdot dt$$

Where  $K_p = 0.45$  and  $K_i = 0.1$  are the proportional and integral gains, respectively.

#### 4.4.5 Line Following with Sensor Correction:

The robot uses a QTR-MD-05A reflectance sensor array to detect and follow a black line on a white background. This sensor array enables the robot to autonomously stay on a designated path, correct its trajectory in real time, and detect a stop line at the end of the course.

#### 4.4.5.1 Sensor Configuration and Calibration

The array consists of five digital IR sensors, connected to GPIO pins 4 through 8. Each sensor outputs HIGH or LOW depending on whether it sees white or black, respectively. The system uses the QTRSensors library to manage calibration, readings, and emitter control:

```
// ----- QTR Sensor Array Setup -----  
const uint8_t NUM_SENSORS = 5;  
const uint8_t sensorPins[NUM_SENSORS] = {4, 5, 6, 7, 8};  
const int sensorWeights[NUM_SENSORS] = {-2, -1, 0, 1, 2};  
const int BLACK_THRESHOLD = HIGH;  
  
qtr.setTypeAnalog();  
qtr.setSensorPins((const uint8_t[]){4, 5, 6, 7, 8}, 5);  
qtr.setEmitterPin(2);
```

*Figure 19 A QTR Sensor Configuration and Calibration*

During setup(), the sensors are calibrated over 400 cycles to establish reflectance thresholds for the specific surface used in testing. This ensures robust and repeatable behavior across different lighting or material conditions.

#### 4.4.5.2 Line Position Detection

The robot uses two methods for line tracking:

- Digital logic for stop-line detection
- Analog weighted average for line correction

In each control loop, the firmware checks if all five sensors detect black. If so, it interprets this as a stop line, halts the robot, and reports the average speed:

```
bool allOnBlack = false;
int dummyError;

readLineSensors(dummyError, allOnBlack);

if (allOnBlack) {
    Serial.println("STOP LINE DETECTED - Stopping motors.");
    stopMotors();
    digitalWrite(STOP_LED, HIGH);
}
```

*Figure 20 -B Line Position Detection Protocol*

Separately, the analog line position is computed using the weighted sum method:

$$Line\ Error = \frac{\sum_{i=1}^5 W_i * A_i}{\sum_{i=1}^5 A_i}$$

Where:

- $A_i$  is the active state (1 or 0) of each sensor
- $W_i$  is the assigned weight: -2, -1, 0, +1, +2

This formula provides a numerical “error” from the center, where:

- A negative value means the robot is veering left
- A positive value means it’s veering right
- Zero means it’s centered on the line

```

void readLineSensors(int &linePos, bool &allOnBlack) {
    int weightedSum = 0;
    int sum = 0;
    allOnBlack = true;
    Serial.print("QTR: ");
    for (uint8_t i = 0; i < NUM_SENSORS; i++) {
        int sensorVal = digitalRead(sensorPins[i]);
        Serial.print(sensorVal); Serial.print(" ");
        if (sensorVal != BLACK_THRESHOLD) allOnBlack = false;
        int active = (sensorVal == BLACK_THRESHOLD) ? 1 : 0;
        weightedSum += sensorWeights[i] * active;
        sum += active;
    }
    Serial.println();
    linePos = (sum > 0) ? (weightedSum / sum) : 0;
}

```

*Figure 21 C Line Error Logic*

#### 4.4.5.3 Trajectory correction

The calculated line error is scaled by a gain constant ( $k_{line} = 10.0$ ) and used to offset the PWM values for the left and right motors:

```

float correction = k_line * lineError;
updateMotorOutputs(correction);

```

*Figure 22 -D Correction computation and PWM Adjustment*

This allows the robot to continuously correct its trajectory in real-time and remain aligned with the track. If the robot loses the line entirely, it defaults to neutral control behavior.

This implementation of QTR sensor feedback provides reliable, proportional line-following performance while also enabling controlled stopping behavior—both of which are critical to demonstrating distance and speed concepts accurately.

#### 4.6.6 Post-Run LCD Display Output

The robot features a 16x2 I2C LCD display, which provides post-run feedback to students and observers. Rather than displaying continuously during motion, the LCD is updated after the robot completes its run, ensuring the data is readable and contextually meaningful once the robot has stopped.

#### 4.4.6.1 LCD Integration

The LCD module is connected via I2C using the LiquidCrystal\_I2C library, which simplifies to communication over just two GPIO pins (SDA and SCL). Due to compatibility issues with newer Esp32 board packages, the firmware uses Esp32 Arduino core version 2.0.9 to ensure stable operation of the LCD display.

The LCD is initialized in setup()

```
// Set the LCD address and dimensions
LiquidCrystal_I2C lcd(0x27, 16, 2);
lcd.init();           // Initialize the LCD
lcd.backlight();
lcd.setCursor(0, 0);
```

*Figure 23 LCD Instantiation*

This ensures the display is prepared and responsive for use when the robot concludes its motion.

#### 4.4.6.2 Post-Run Data Display

Once the robot reaches the end of the track (detected via all five QTR sensors detecting black, it stops and calculates the average speed over the run. This value is then displayed on the LCD for students to observe.

This approach avoids trying to show rapidly changing data while the robot is in motion – something impractical for classroom use –and instead provides a clear, final result that supports student analysis after each run.

#### **4.4.6.3 Educational Value**

Displaying the final average speed on the LCD offers a tangible, real-world confirmation of the robot's behavior that students can compare against their own predictions or calculations. It acts as an immediate feedback tool that reinforces the connection between input speed, motion, and measurement.

#### **4.4.7 App-Based Real-time Feedback and Telemetry**

While the robot's LCD provides post-run data for physical observation, the robots mobile application offers continuous real-time feedback during motion. This live telemetry allows educators and students to monitor the robot's speed, RPM, and line-following status as the robot moves.

##### **4.4.7.1 Live Data Transmission**

Every 20 milliseconds, the firmware constructs a data string containing key operation values:

- Target speed(m/s)
- Left and right motor speeds(m/s)
- Left and right motor RPMS
- Line error (QTR sensor deviation)
- Running average speed

```

String data = "Target: " + String(targetSpeed) + " m/s | M1: " +
  String(speed1) + " m/s, " + String(rpm1) + " RPM | M2: " +
  String(speed2) + " m/s, " + String(rpm2) + " RPM | LineError: " +
  String(lineError) + "Avg Speed:" + String(avgSpeed);
Serial.println(data);
if (deviceConnected) {
  pCharacteristic->setValue(data.c_str());
  pCharacteristic->notify();
}

```

*Figure 24 Data Telemetry transmission to the App*

## **Educational Value**

This real-time feedback system plays a crucial role in enhancing the learning experience. By visualizing live speed data, RPM, and line-following behavior as the robot moves, students can directly observe how their input affects the robot's performance. This immediate cause – and- effect relationship reinforces core STEM concepts such as motion, speed calculation, control systems, and feedback loops. Unlike the post-run LCD output, the app allows continuous monitoring, encouraging experimentation and critical thinking as students adjust speed settings and analyze the robot's response. This interactivity transforms the learning process from passive observation to active engagement, making abstract principles more tangible and easier to understand.

## 4.5 Block Diagram

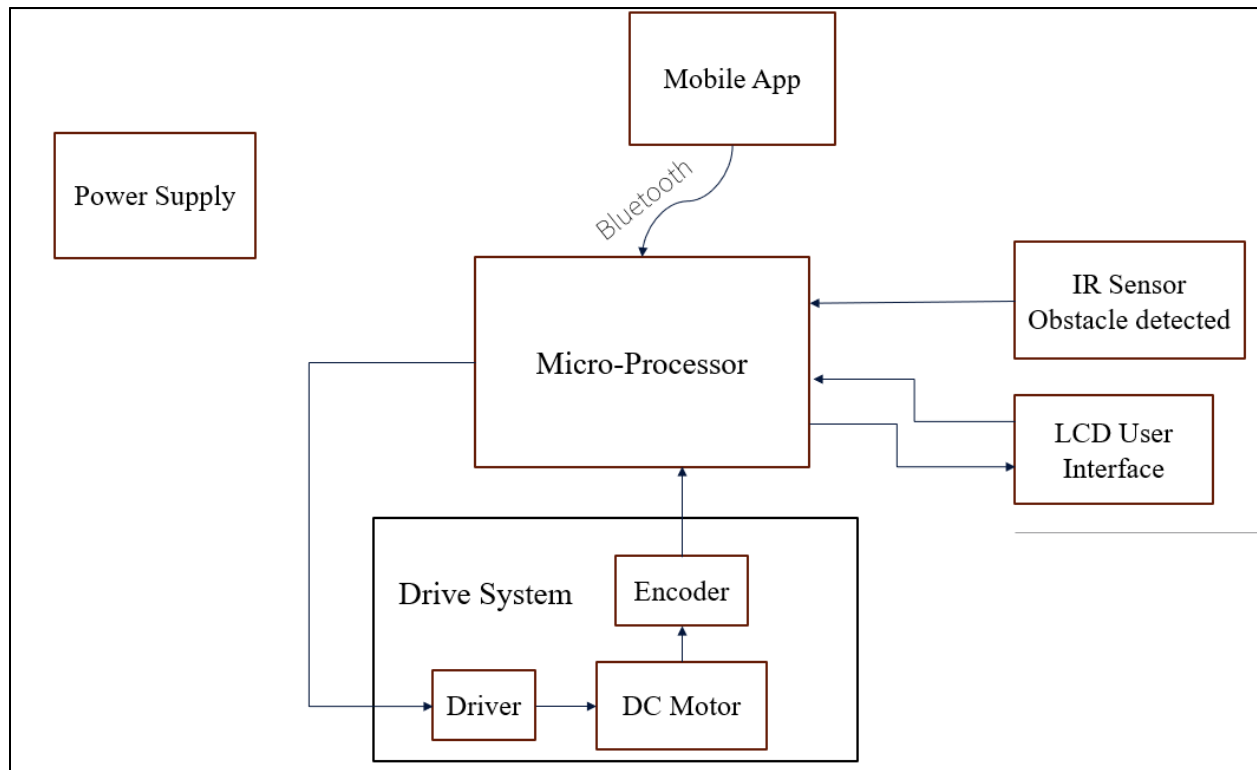


Figure 25 Original Block Diagram

The figure above demonstrates the original block diagram and outlines the early-stage architecture for the robot system. The microprocessor receives commands wirelessly from a mobile app via Bluetooth. It controls the drive system, which includes the driver, DC motor, and encoder for motor speed feedback. The IR sensor is used to detect obstacles, triggering the robot to stop, or adjust movement accordingly. The LCD used interface is used to display real-time data, such as speed or obstacle detection status. Power is distributed across the system from a centralized supply.



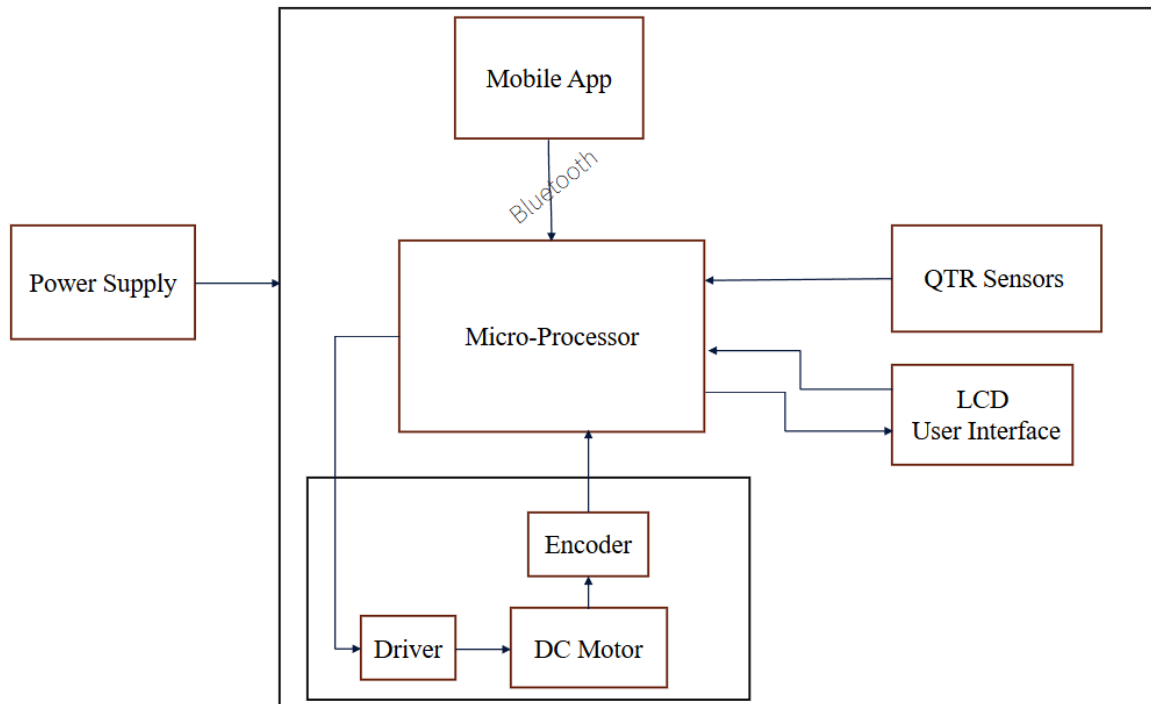


Figure 26 Final Block Diagram

The figure above presents the final block diagram and showcases the updated system architecture for the robot. The microprocessor remains the central control unit and continues to receive commands wirelessly from the mobile app via Bluetooth. In this version, QTR reflectance sensors are integrated for line-tracking and stopping capabilities, enhancing movement accuracy. The LCD user interface continues to display real-time data, such as speed and sensor readings. The drive system, consisting of the driver, DC motor, and encoder, remains in place to provide feedback for motor speed control. Power is supplied through a centralized source. These additions reflect the transition from basic obstacle detection to precise line-following and improved speed regulation.

#### 4.6 Flowchart

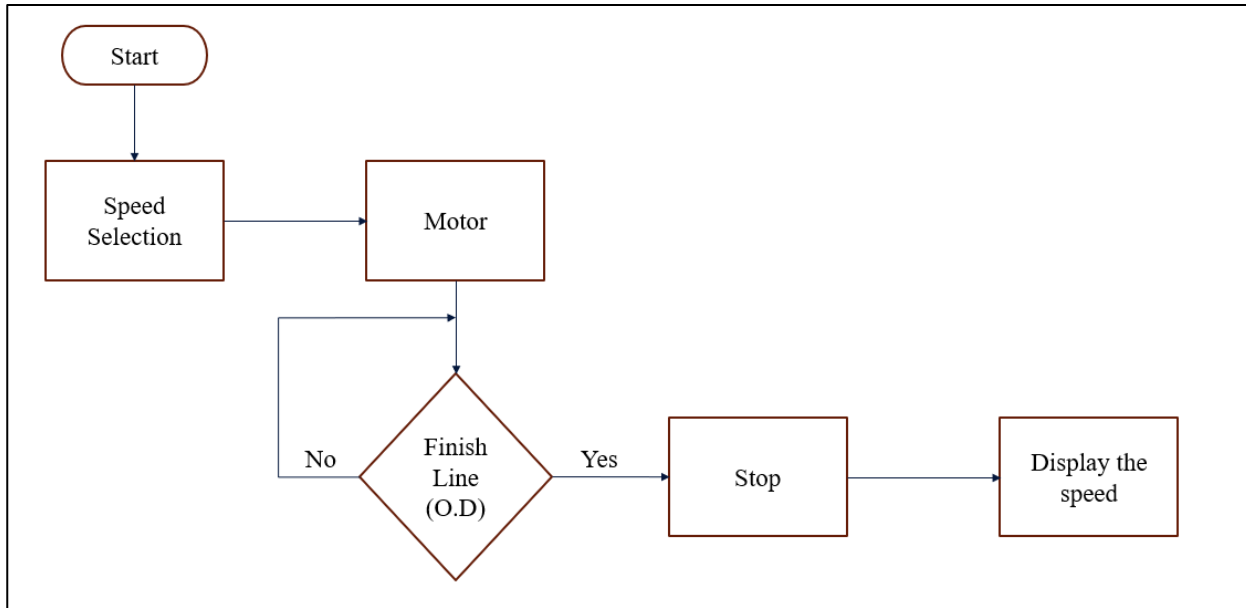


Figure 27 Original Flow Chart

The original flowchart represents a basic logic structure for controlling the robot. It begins with speed selection, followed by activating the motor. The system then continuously checks whether the robot has reached the finish line using an optical detection (O.D.) sensor. If the finish line is not reached, the motor continues to run. Once the finish line is detected, the robot stops and the speed is displayed. This flow emphasizes a straightforward linear process, with limited feedback or control adjustments.

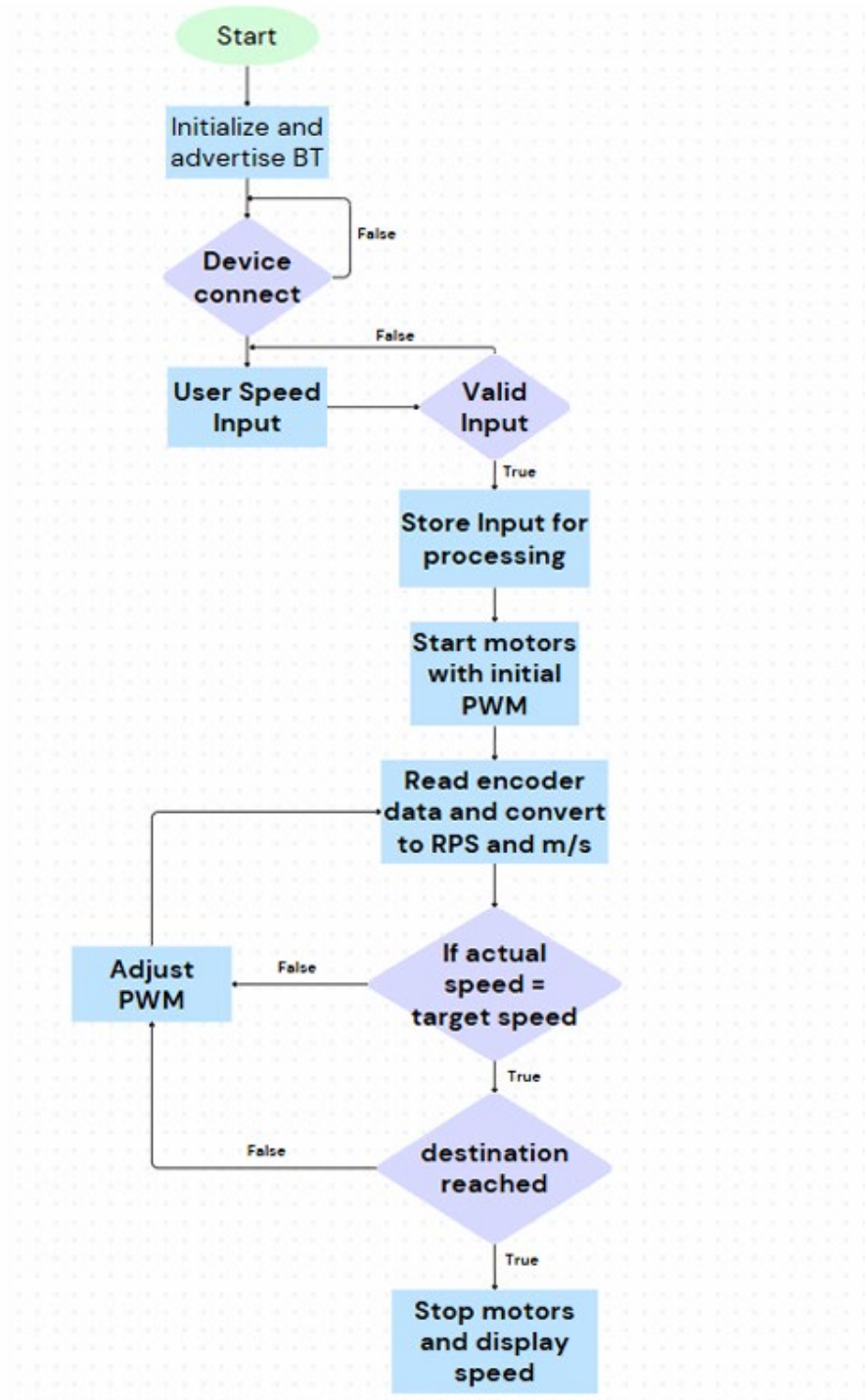


Figure 28 Final Flow Chart

The final flowchart outlines a more advanced logic design that incorporates feedback and dynamic control. The process begins by initializing Bluetooth communication and waiting for a device to connect. After receiving valid speed input from the user, the system stores the input and

starts the motors with an initial PWM value. It then reads encoder data, converts it to RPS and m/s, and continuously checks whether the actual speed matches the target. If not, it adjusts the PWM accordingly. Once the correct speed is achieved and the destination is reached, the system stops the motors and displays the final speed. This flowchart reflects a closed-loop control system with enhanced accuracy and user interaction

#### **4.7 Setbacks**

Throughout the development of our educational robot, we encountered several challenges that delayed progress and required design adjustments. One of the earliest setbacks involved our initial motor driver choice. We started with the L298N, but after multiple tests, we realized it couldn't deliver consistent speed control at higher duty cycles, especially under load. This resulted in erratic robot movement and caused inaccurate RPM readings. Eventually, we transitioned to the VNH5019 motor driver, which provided the higher current capacity and reliability we needed.

Another significant issue was related to motor synchronization. Although both motors were identical, we noticed the robot veering to one side during forward motion. After extensive debugging, we determined the discrepancy was due to variations in the encoder feedback and motor response. To resolve this, we implemented a PI control algorithm to balance the speeds of both motors, which required weeks of tuning and testing.

Integration of the IR sensors also presented challenges. We initially placed the sensors too far from the centerline of the robot, causing false detections and unexpected stops during wall-approach testing. Repositioning and recalibrating the sensors fixed the issue but set us back on our development timeline.

Bluetooth control was another hurdle. While the ESP32 provided built-in BLE capabilities, intermittent disconnections and pairing issues between the microcontroller and the Flutter app made remote control unreliable. We revised our BLE handling code and streamlined the command structure to ensure consistent communication between the app and the robot.

Finally, when we first integrated all the features into one program, the robot would freeze or behave unpredictably. This was traced back to blocking code in sensor readings and PWM updates. Converting the structure into a non-blocking, interrupt-based system allowed smoother multitasking and real-time response.

Despite these technical and integration setbacks, each issue helped us better understand the system and led to a more robust final design. The successful demonstration of the robot's mobility, sensor feedback, and wireless control highlighted the team's perseverance and problem-solving throughout the project

## 5. Gantt Chart – Project Timeline

### 5.1 Initial Gantt Chart

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13
<b>Research</b>													
<b>Prototyping</b>													
<b>Hardware Development</b>													
<b>Software Development</b>													
<b>Testing</b>													

Table 8: Initial Gantt Chart

The original Gantt chart outlined a 13-week plan dividing the project into five key phases: Research, Prototyping, Hardware Development, Software Development, and Testing. Research was scheduled for the first three weeks to establish design goals and explore technical requirements. Prototyping overlapped with the end of the research phase and continued until Week 6, allowing time to build and evaluate initial models. Hardware development was planned for Weeks 5 through 8, followed by software development from Weeks 8 to 11. The final phase, Testing, was set for Weeks 11 to 13 to validate system performance and integration. This timeline provided a structured approach, although actual progress required adjustments due to unforeseen setbacks during implementation.

## 5.2 Prototype Final Gantt Chart

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13
<b>Research</b>													
<b>Prototyping</b>													
<b>Hardware Development</b>													
<b>Software Development</b>													
<b>Testing</b>													

Table 9: Prototype Final Gantt

The Prototype Final Gantt chart reflects adjustments made to the original project timeline in response to development setbacks. While the research phase remained on schedule during Weeks 1 to 3, delays in hardware component sourcing and motor driver issues pushed hardware development to start later, extending it from Week 5 through Week 10. Prototyping was condensed and completed in Weeks 3 and 4 to allow more time for hardware refinement. Software development, initially planned earlier, was rescheduled to start in Week 8 and continue

through Week 11 due to integration challenges with the motor control system and BLE communication. Testing was also delayed, beginning in Week 10 and running through the end of the project in Week 13, allowing for troubleshooting and validation of the final robot functionality. These shifts ensured that despite initial delays, all core project objectives were still achieved.

5.3 Final Gantt Chart



Table 10: Final Gantt Chart

The final Gantt chart provides a detailed view of task responsibilities and timeline progression for the Educational Robot K-12 project from early February through the first week of May 2025. The project spans multiple overlapping phases, with research efforts divided among team members: Jesus Rios and Denisse Galvan handled application research, and component research was conducted by Juan Vega and Aaron Jackson. Development tasks followed the research phase—application development was led by Denisse and Jesus, software development was a team effort, and hardware development was carried out by Jesus and Juan. Each phase was strategically overlapped to maximize time efficiency and collaborative progress.

The final testing phase, involving all team members, occurred near the end of the timeline, ensuring the robot met functionality and performance expectations before the project's conclusion. This chart captures the finalized structure, showing clear ownership, task dependencies, and realistic deadlines refined through prior iterations.

## **5.4 Financial Planning**

### Microcontroller

#### Arduino Mega 2560 Microcontroller

- Must be capable of handling multiple input/output devices including motor drivers, sensors, and display modules
- Provides sufficient digital and analog pins for encoder feedback and sensor integration.

### Motors

#### 2 DC Motor with Encoders

- Must provide adequate torque to move the robot on classroom surfaces.
- Encoders must enable real-time speed and distance feedback for accurate movement tracking.

### Motor Driver

#### Generic Motor Driver

- Must support bi-directional control of two DC motors.
- Must be compatible with the 12 V motors and Arduino PWM signals.



## Sensor

### Infrared (IR) Sensor

- Must be able to display real-time speed (RPM), status messages, or sensor data..

## Display

### LCD Display Module

- Must be able to display real-time speed (RPM), Status Messages, or sensor data

## Chassis

### Wheel Set

- Must be compatible with the motor shaft
- Must provide sufficient grip for smooth movement across flat surface.

## Mounting Hardware

### L-Bracket Pair

- Must be compatible with the motor shaft.
- Must provide sufficient grip for smooth movement across flat surface.

### 6mm Shaft Mount

- Must ensure stable attachment between motor shaft and wheel.
- Must be compatible with the selected motor diameter.

## 5.5 Original Budget Plan

Item	Quantity	Unit cost (USD)	Total Cost (USD)
<b>Fall 2024</b>			
Wheels	2	\$5.75	\$11.50
DC Motors with Encoders	2	\$51.50	\$103.00
Arduino Mega 2560	1	\$41.00	\$41.00
L298N Motor Driver	1	\$9.99	\$9.99
IR Sensor	1	\$10.00	\$10.00
Chassis	1	\$15.00	\$15.00
Display	1	\$10.00	\$10.00
L-Bracket Pair	1	\$9.95	\$9.95
Mounting for 6mm Shaft	1	\$9.95	\$9.95
Total Fall 2024			<b>\$220.39</b>
<b>Spring 2025</b>			
ESP32	1	\$32.99	\$32.99
Battery Pack	1	\$20.99	\$20.99
DC-DC Buck converter	1	\$12.99	\$12.99
Total Spring 2025			<b>\$66.97</b>
<b>PROJECT TOTAL</b>			<b>\$287.36</b>

Table 11: Original Budget Plan

## 5.6 Final Budget Plan

Item	Quantity	Unit cost (USD)	Total Cost (USD)
<b>Spring 2025</b>			
QTR-MD-05A Reflectance Sensor Array	1	\$5.82	\$5.82
Ball Caster Kit	2	\$7.00	\$14.00
Dual VNH5019 Motor Diver	1	\$69.95	\$69.95
Adhesive Steel Wheel Weights	72	\$0.37	\$26.99
Jumper Wires	240	\$0.04	\$9.19
Solder Flux	2	\$2.99	\$5.99
Total Spring 2025			<b>\$131.94</b>
<b>PROJECT TOTAL</b>			<b>\$131.94</b>

Table 12: Final Budget Plan

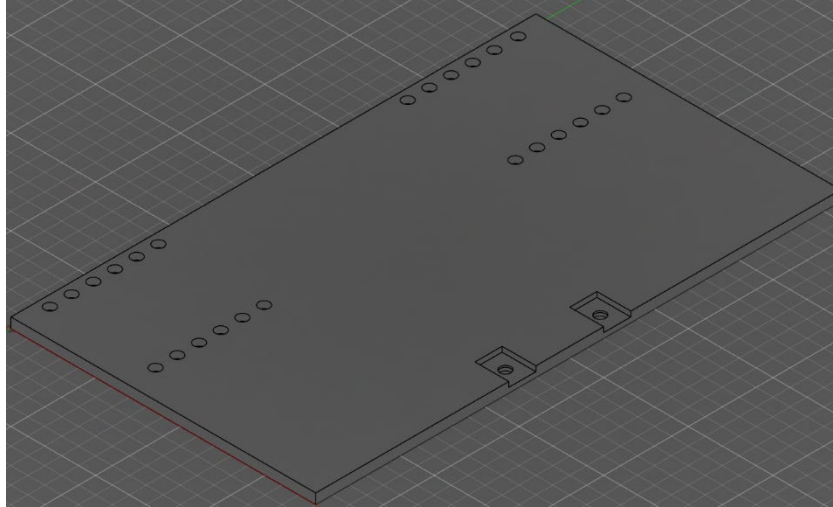
## 5.7 Project Cost for the K-12 Educational Robot

Item	Quantity	Unit cost (USD)	Total Cost (USD)
<b>Fall 2024</b>			
Wheels	2	\$5.75	\$11.50
DC Motors with Encoders	2	\$51.50	\$103.00
Arduino Mega 2560	1	\$41.00	\$41.00
L298N Motor Driver	1	\$9.99	\$9.99
IR Sensor	1	\$10.00	\$10.00
Chassis	1	\$15.00	\$15.00
Display	1	\$10.00	\$10.00
L-Bracket Pair	1	\$9.95	\$9.95
Mounting for 6mm Shaft	1	\$9.95	\$9.95
Total Fall 2024			<b>\$220.39</b>
<b>Spring 2025</b>			
ESP32	1	\$32.99	\$32.99
Battery Pack	1	\$20.99	\$20.99
DC-DC Buck converter	1	\$12.99	\$12.99
QTR-MD-05A Reflectance Sensor Array	1	\$5.82	\$5.82
Ball Caster Kit	2	\$7.00	\$14.00
Dual VNH5019 Motor Diver	1	\$69.95	\$69.95
Adhesive Steel Wheel Weights	72	\$0.37	\$26.99
Jumper Wires	240	\$0.04	\$9.19
Solder Flux	2	\$2.99	\$5.99
Total Spring 2025			<b>\$198.91</b>
<b>PROJECT TOTAL</b>			<b>\$419.30</b>

Table 13: Project Cost for the K-12 Educational Robot

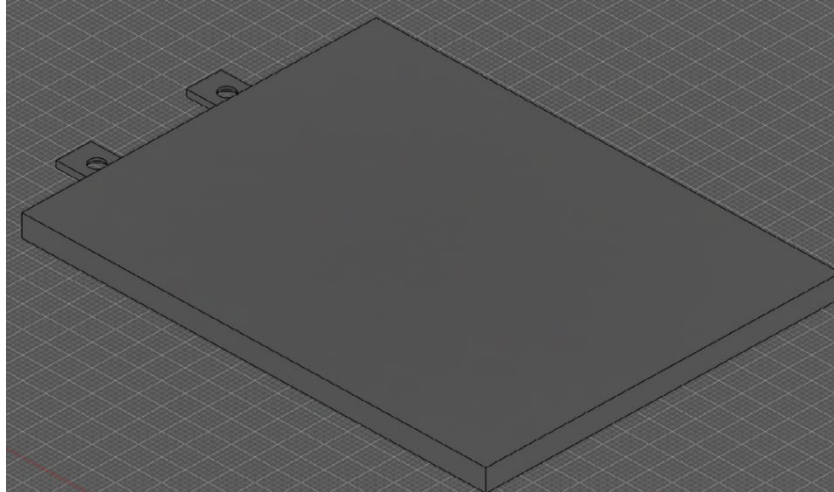
## 6. Final System Design and Test Results

### 6.1 Prototype Final Design



*Figure 29 Front Chassis*

The figure above shows the initial design of SpeediBot front chassis, which serves as the mounting base for the two DC motors. This rectangular platform features symmetrically placed mounting holes to ensure balanced motor alignment, enabling smooth and accurate forward motion. Reinforced tabs near the front edge provide added structural support for mounting or securing additional components. The flat surface offers a stable foundation for attaching electronics modules such as the motor driver or IR sensor. Designed for both function and simplicity, this chassis supports the core drive system of the robot and can be further improved by adding cable management slots, sensor mounts or reducing excess materials for weight optimization.



*Figure 30 Rear Chassis*

The image above shows the rear chassis design of SpeediBot, which is intended to hold the motor driver and breadboard components. This rectangular platform provides a solid, flat surface for mounting electronic modules securely during operation. The two mounting tabs at the back offer additional support or attachment points to keep the structure stable and organized. Unlike the front chassis, this section does not require motor mounting holes, allowing for a cleaner layout focused on electrical connectivity. This design helps separate power components from the drive system, improving wire management and ease of maintenance.

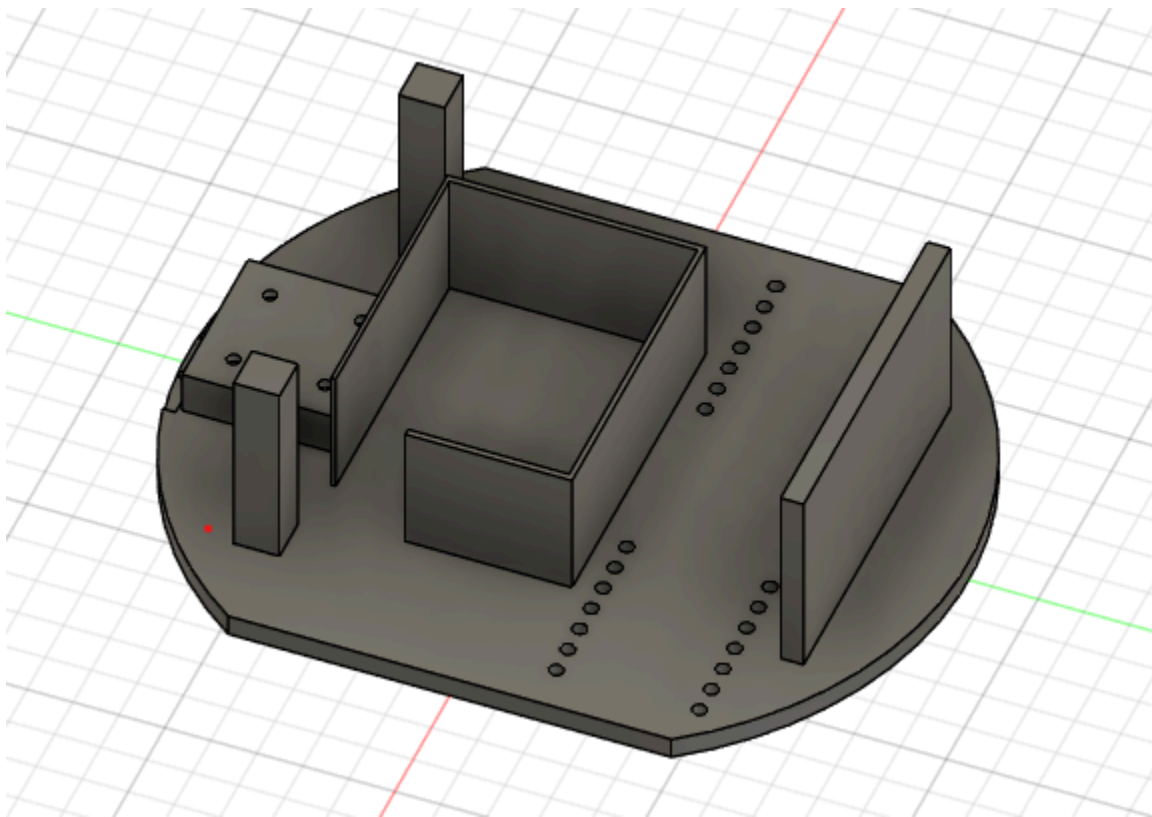
## **6.2 Improvements and Corrections**

We have made improvements and corrections to the robot since the past semester. One of the main ones would be we got rid of the buttons for the speed inputs and replaced it with the smartphone application, which was a major improvement since we are not limited to four user inputs anymore. We changed the motor driver to the Dual VNH5019 Motor Driver since the last motor driver did not let us input lower speeds, if it had a duty cycle lower than 45% the motors would not move. Changing the chassis to a stronger to improve durability and stability, upgrading the IR sensor to the QTR sensor for line-following to provide more reliable tracking

performance, and added a battery and a power button to improve user experience by making the robot more self-contained and easier to operate. Improved the robot's software by adding calculations from m/s to RPM, line-tracking, stopping mechanism, PI control for the motors, and P control for the QTR sensors.

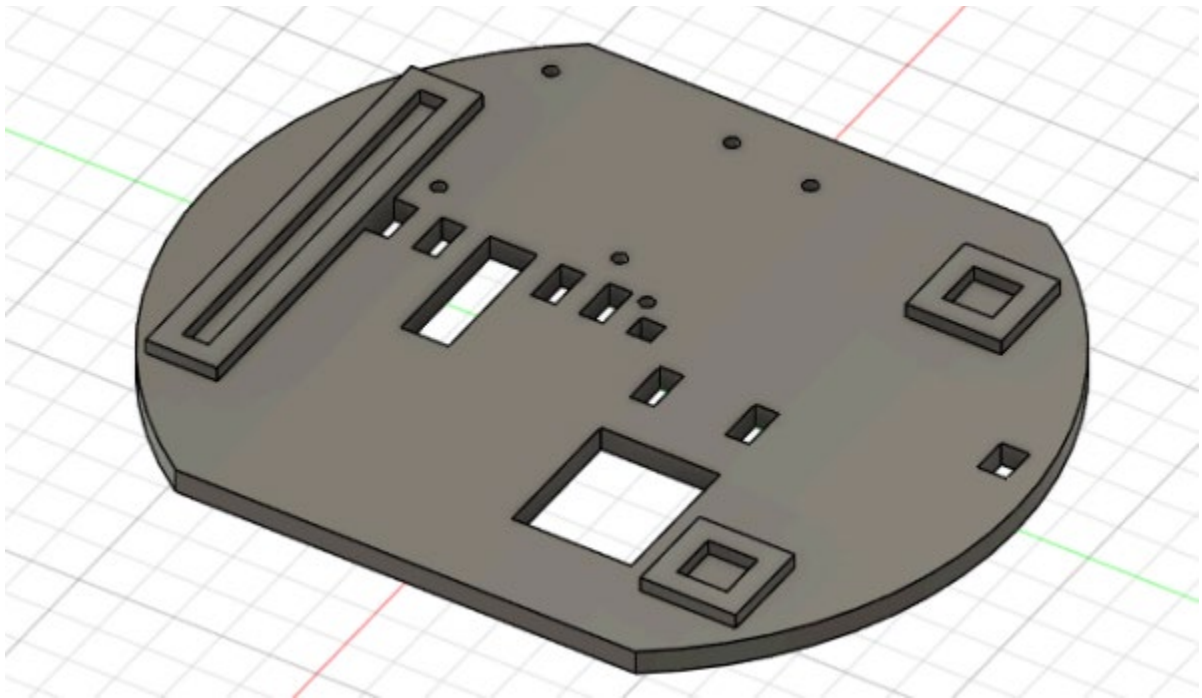
Overall, these modifications address functional limitations, improve user experience and enhance the robot's capabilities.

### 6.3 Final Design



*Figure 31 Mounting bracket chassis*

The figure above demonstrates how the mounting brackets allow us to adjust the position of the motors by extending them from the base. This layer also adds a shell that can secure the battery to the chassis without having to use adhesives or screws. The pillar frames protruding from the base allow for this layer to connect to the second layer in the figure below without having to use adhesive or screws. Additionally, this layer also allows us to mount the QTR sensor.



*Figure 32 Middle Chassis*

The figure above shows how the base has openings that allow circuitry from the first layer, in figure-#, through. The bores in the base allow for the mounting of 2 different types of motor drivers (L298N or Dual VN5019 Motor Driver). Additionally, the cavities seen are for mounting to the first layer without adhesives or screws.

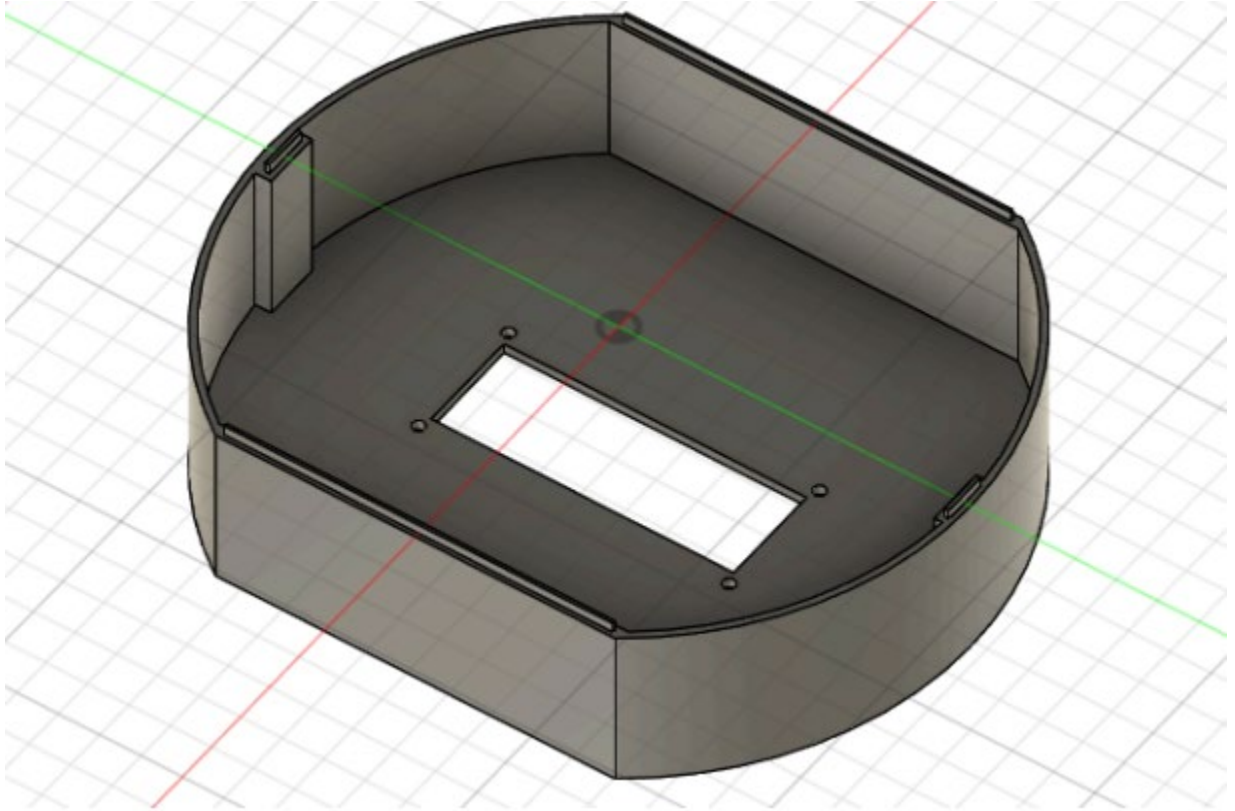


Figure 33 Top Cover

The figure form above is the top cover of the robot that hides the circuitry form layer 2. The cover allows for an LCD screen to be mounted with screws. Also, the cover can also be mounted to layer 2 without adhesives or screws.

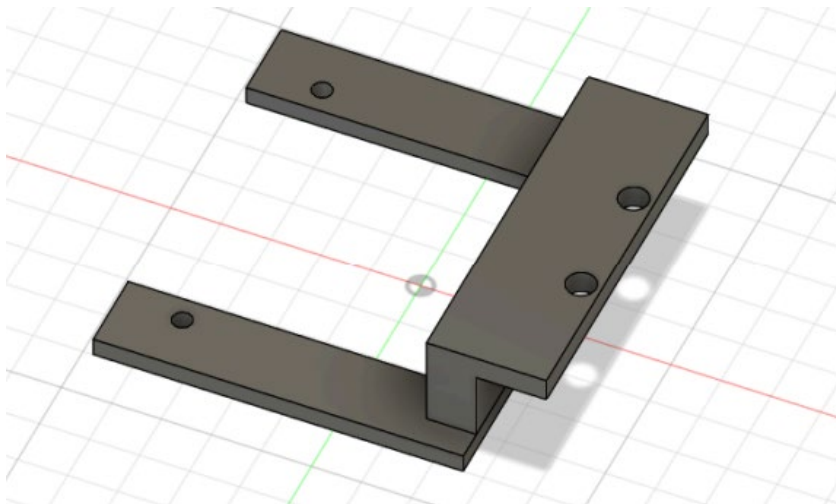


Figure 34 QTR mount



The figure above shows the QTR mount that allows us to mount the QTR sensor array to the first layer of the robot. When mounted to the first layer the QTR sensors are approximately 5mm (about 0.2 in) off the ground which is the optimal is the optimal sensing distance for the module.

The image below shows the internal components of the robot, including the main control system and wiring layout. An ESP32 microcontroller is mounted on the top platform, connected to various sensors and motor drivers through color-coded jumper wires. The middle section houses the stacked DC-DC converters that regulate power distribution to different components. At the front of the robot, the QTR reflectance sensor is mounted and wired for surface detection during navigation. The organized internal layout ensures stable mechanical structure, efficient power management, and reliable sensor and motor operation.

## **6.4 Results**

After finalizing the hardware and software integration, including the encoder setup, BLE communication, and motor control logic, we conducted comprehensive testing to evaluate the robot's real-world speed tracking performance. The goal was to assess how accurately the robot could achieve commanded speeds and how long it took to stabilize at those speeds. This is critical for both educational feedback and for validating our PI control implementation.

Three separate test trials were conducted using speed inputs of 0.2 m/s, 0.4 m/s, and 0.6 m/s. For each input speed, we recorded three key parameters: the settling time, the encoder-reported speed (E.S), and the calculated speed (C.S) based on distance traveled over time. These trials helped us understand both response time and accuracy across different conditions and robot start states.

Each trial began with the robot stationary and placed at the same starting location. A speed command was sent via the mobile app using Bluetooth. The robot was allowed to reach the target speed while encoder data was continuously logged. The **settling time** refers to the duration it took the robot to reach and maintain a stable velocity within  $\pm 0.02$  m/s of the target.

- **Encoder Speed (E.S)** was derived from pulse counts over a known interval, calibrated based on wheel diameter and encoder resolution.
- **Calculated Speed (C.S)** was obtained by manually measuring the distance the robot traveled during a fixed time window and dividing distance by time.

This dual-speed comparison enabled us to verify the accuracy of the encoder measurements and ensure that our control logic was performing consistently.

### Trail 1

Input	Test	Time(s)	E.S	C.S
0.2	1	8.8	0.22 m/s	0.22 m/s
	2	8.92	0.22 m/s	0.23 m/s
	3	8.94	0.22 m/s	0.23 m/s
0.4	1	5.2	0.38 m/s	0.39 m/s
	2	5.66	0.35 m/s	0.39 m/s
	3	5.55	0.36 m/s	0.37 m/s
0.6	1	3.26	0.58 m/s	0.58 m/s
	2	3.64	0.55 m/s	0.58 m/s
	3	3.58	0.55 m/s	0.56 m/s

Table 14: Trial 1 Results

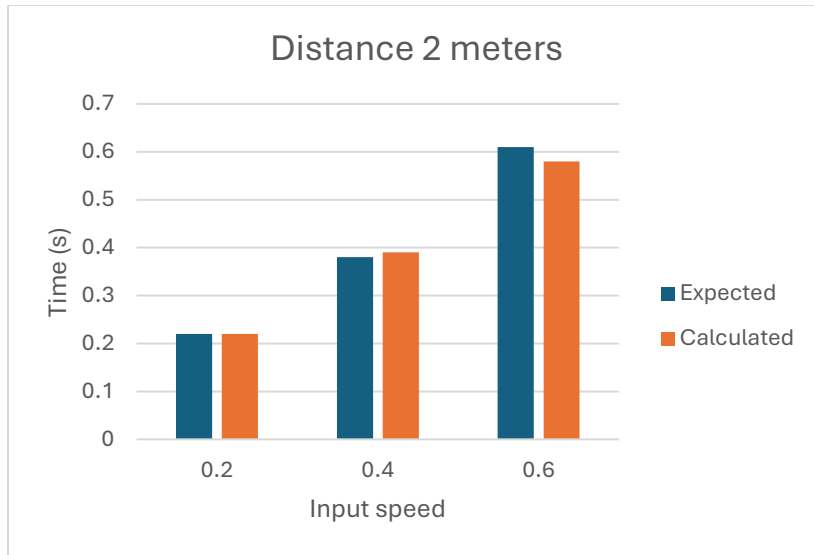


Figure 35 2 meters Trial

In Trial 1, the robot responded quickly to each speed command. The settling times were shortest in this trial, particularly at higher inputs, where the robot stabilized at 0.6 m/s in just 3.26 seconds. The encoder and calculated speed values were virtually identical, demonstrating excellent calibration and effective response from the PI controller. For each test recorded in the table, multiple runs were conducted, and the results were averaged to ensure consistency and minimize the impact of any minor fluctuations in performance.

## Trial 2

Input	Test	Time (s)	E.S	C.S
0.2	1	13.65	0.21 m/s	0.2 m/s
	2	13.47	0.22 m/s	0.22 m/s
	3	13.58	0.22 m/s	0.2 m/s
0.4	1	7.88	0.37 m/s	0.37 m/s
	2	8.02	0.37 m/s	0.37 m/s

	3	7.77	0.38 m/s	0.41 m/s
0.6	1	5.08	0.59 m/s	0.61 m/s
	2	5.7	0.59 m/s	0.62 m/s
	3	5.14	0.58 m/s	0.61 m/s

Table 15: Trial 2 Results

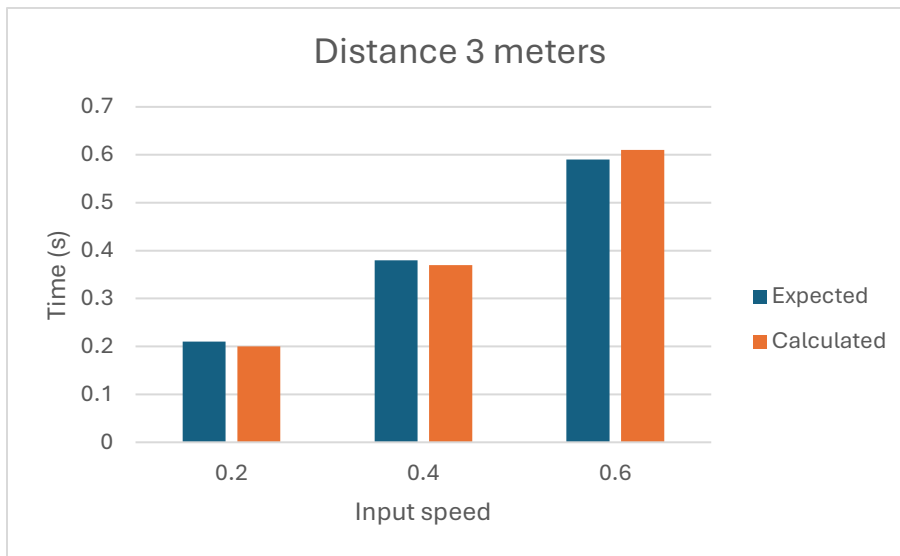


Figure 36 3 meter trial

Trial 2 showed a slightly slower response time, particularly for the 0.2 m/s input, which took 13.65 seconds to stabilize. However, once stabilized, the encoder and calculated values remained consistent and within acceptable tolerance limits. The slight delays may have resulted from variations in motor warm-up behavior or minor differences in initial encoder alignment. For each test recorded in the table, multiple runs were conducted, and the results were averaged to ensure consistency and to account for any minor fluctuations in motor performance.

### Trial 3

Input	Tests	Time (s)	E.S	C.S
-------	-------	----------	-----	-----

0.2	1	19.37	0.21 m/s	0.2 m/s
	2	19.24	0.2 m/s	0.22 m/s
	3	19.35	0.2 m/s	0.19 m/s
0.4	1	11.21	0.38 m/s	0.36 m/s
	2	11.21	0.38 m/s	0.36 m/s
	3	11.29	0.35 m/s	0.36 m/s
0.6	1	6.95	0.57 m/s	0.59 m/s
	2	6.97	0.57 m/s	0.59 m/s
	3	7.03	0.56 m/s	0.61 m/s

Table 16: trial 3 Results

In Trial 3, the robot exhibited the longest response times, particularly for the lowest speed input. The 0.2 m/s command required nearly 20 seconds to stabilize, which may be attributed to motor startup torque limitations or slight variations in battery voltage. Despite these delays, the final speeds achieved closely matched the commanded values, with encoder readings and measured speed staying within a 0.02 m/s margin. For each test recorded in the table, multiple runs were performed, and the results were averaged to provide a more accurate representation of the robot's performance and to minimize the effects of any individual anomalies.

Across all three trials, the robot demonstrated strong performance in speed tracking accuracy. The encoder speed (E.S) and calculated speed (C.S) values closely aligned at all input levels, validating the encoder calibration and confirming the effectiveness of the PI control algorithm. Although variations in settling time were observed—especially at lower speeds—these are expected in real-world conditions and are likely influenced by initial load conditions, surface friction, and minor differences in motor response.

The consistency in final speed measurements across trials shows that the robot is capable of maintaining stable, accurate velocity over time. This is critical for delivering an educational experience that is both interactive and reliable. Furthermore, the data reinforces that the system is ready for future integration with dynamic speed control via mobile app input in units of m/s.

Future improvements could focus on optimizing low-speed response time through more refined PI tuning or exploring closed-loop startup routines. Nonetheless, the results clearly demonstrate that the robot is performing within acceptable parameters and ready for real-world demonstration and use in classroom environments.

## **6.2 Statistical Analysis**

To evaluate the accuracy and consistency of the robot's speed based on input speeds, internally calculated speeds (expected), and manually measured speeds (calculated) using stopwatch and distance.

### **Definitions:**

- **Input:** Target speed sent to the robot (in m/s).
- **Expected:** Speed calculated by the robot upon stopping.
- **Calculated:** Speed derived manually using the formula:  $\text{Speed} = \text{Distance} / \text{Time}$ .

### **Statistical Metrics Used:**

For each input speed:

- **Mean Manual Speed (Calculated)**

- **Mean Robot Speed (Expected)**
- **Standard Deviation (SD) of Manual Speeds** – indicates consistency of manual measurements.
- **Mean Absolute Error (MAE):**  $|\text{Expected} - \text{Calculated}|$  – measures robot's speed accuracy relative to manual timing.
- **Mean Absolute Input Error (MAIE):**  $|\text{Input} - \text{Calculated}|$  – measures how closely manual speed matched the input command.

#### Analysis Summary:

Input (m/s)	Mean Manual (m/s)	Mean Robot (Expected)(m/s)	SD (Manual)(m/s)	MAE (Robot vs Manual)(m/s)	MAIE (Input vs Manual)(m/s)
0.2	0.212	0.212	0.015	0.009	0.014
0.4	0.377	0.363	0.017	0.016	0.026
0.6	0.594	0.574	0.019	0.020	0.017

#### Interpretation:

##### Accuracy:

- **0.2 m/s input:** Robot's reported speed matches manual measurements closely (MAE ~0.009 m/s).

- **0.4 m/s input:** Robot slightly **undershoots** the target (mean expected: 0.363), while manual timing shows 0.377 (MAE  $\sim$ 0.016 m/s).
- **0.6 m/s input:** Robot slightly **undershoots** (expected average 0.574 vs manual 0.594), giving an MAE of  $\sim$ 0.020 m/s.

### **Precision (Consistency):**

- Manual timing consistency (SD):
  - 0.015 m/s at 0.2 input
  - 0.017 m/s at 0.4 input
  - 0.019 m/s at 0.6 input
- Indicates reliable repeatability across trials.

### **Deviation from Input:**

- Maximum manual deviation from input (MAIE) is  $\sim$ 0.026 m/s at 0.4 input, and less for other inputs.
- The robot's performance remains within  $\pm 0.03$  m/s of commanded speeds.

The robot demonstrates:

- **High accuracy**, with speed errors below 0.02–0.03 m/s in all cases.



- **Strong precision**, with low variability in manual measurements.
- A **slight tendency to undershoot** target speeds at higher inputs (0.4 and 0.6), which could be refined with better PI tuning.

Overall, the speed measurement and control system is highly reliable for educational and demonstration purposes.

## **6.6 SpeediBot Manual**

We created a manual for the teacher to have a step-by-step process on how speediBot works. If they have any issue, error, or malfunction they can always open the manual and have there issues resolve.

# SPEEDIBOT MANUAL

Juan Vega | Denisse Galvan | Aaron Jackson | Jesus Rios

UTRGV | EECE

## Table of Contents

- Introduction
  - Purpose of SpeediBot
  - Educational Objectives
- Safety Guidelines
  - General Usage Precautions
  - Electrical Safety
  - Safe operating Environment
- What's Included
  - Package Contents
- Setup Instructions
  - Charging
  - Powering the robot
- Bluetooth App Setup
  - How to connect to SpeediBot
  - App Controls
- Step-by-Step Operation Guide
  - Choosing a speed
  - Starting a run
  - Reading display values
- Troubleshooting Guide
  - Common issues and Fixes
- Maintenance & Care
  - Cleaning and Storage
  - Battery Tips
  - Contact Info

## **Introduction**

### **Purpose of SpeediBot**

The purpose of SpeediBot is to provide an engaging, hands-on learning tool that helps K–12 students understand key physics and math concepts—specifically speed, distance, and time. Designed to operate at multiple speeds and display real-time motion data, SpeediBot allows students to observe and measure how changes in input affect movement. With its user-friendly interface, Bluetooth control, and on-board LCD display, the robot transforms

abstract formulas like  $Speed = \frac{Distance}{Time}$  into interactive, visual experiences. By experimenting with SpeediBot, students gain practical insights into motion, measurement, and data analysis, reinforcing classroom lessons through physical experimentation. The robot is also a platform for building curiosity in STEM fields by connecting science concepts to real-world applications in a fun and accessible way.

### **Educational Objective**

SpeediBot is designed with the primary educational objective of helping K–12 students explore and understand fundamental principles of motion through hands-on experimentation. By using SpeediBot, students will learn how to calculate average speed using real-world data, compare theoretical and measured values, and analyze how variables like time, distance, and power affect movement. The robot promotes critical thinking and data literacy by encouraging students to make predictions, record observations, and draw conclusions from their experiments. Additionally, SpeediBot supports cross-disciplinary learning by integrating physics, mathematics, and basic programming concepts, aligning with national STEM education goals and helping students build practical skills that extend beyond the classroom.

## **Safely Guidelines**

### **General Usage Precautions**

- Always place the robot on a clean, flat surface before powering it on to avoid tipping or erratic movement.
- Keep hands and objects away from moving wheels while the robot is running to prevent injury or obstruction.
- Turn off the robot when not in use to conserve battery and avoid overheating.

- Do not operate the robot near water or on wet surfaces, as the electronics are not waterproof.
- Avoid sudden impacts or drops, which may damage internal components or affect sensor alignment.
- Use only the recommended power supply or battery, as over-voltage can damage the microcontroller or motor driver.

#### **Electrical Safety**

- Do not connect or disconnect wires while the robot is powered on. Always turn off the robot before making any hardware changes.
- Use the correct voltage as specified (e.g., 12V DC input). Supplying too much voltage can permanently damage the electronics.
- Check for loose wires or exposed terminals before each use to prevent short circuits or accidental contact.
- Only charge batteries under supervision, using the recommended charger. Never leave charging batteries unattended.
- If the robot stops responding or overheating, immediately disconnect the power source and inspect for faults.

#### **Safe operating Environment**

- Operate SpeediBot indoors, on smooth, clean surfaces such as tile, hardwood, or classroom floors. Avoid carpet or uneven terrain.
- Keep the robot away from the edge of tables or desks to prevent accidental falls.
- Clear the robot's path of obstacles before each test run to avoid crashes or inaccurate measurements.
- Do not use the robot near strong magnets or high-frequency devices, as these may interfere with the sensors or Bluetooth connection.
- Always supervise younger users, especially during wiring or programming activities.

## What's Included

### Package Contents

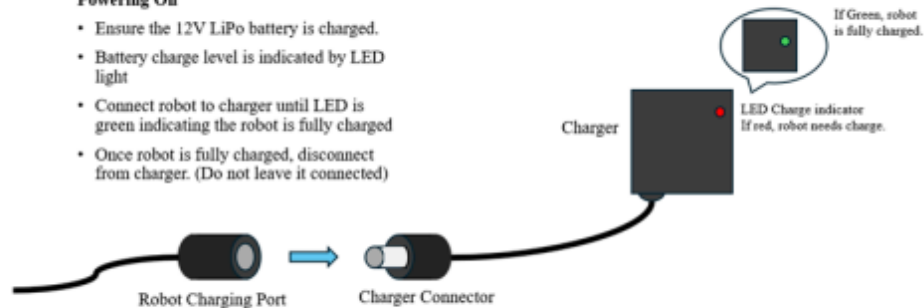
Items	Quantity	Descriptions
ESP32-S3 Microcontroller	1	Pre-installed and programmed for Bluetooth control
VMH5019 Motor Driver	1	Dual-channel driver for precise motor control
16X2 LCD Display	1	Display real-time speed information
QTR Sensor	1	Detect the black line to guide the robot and make it stop
Power Button (On/Off Switch)	1	Turns the robot on and off safely
Rechargeable Battery Pack	1	12 V power supply
Charging Cable	1	Used to recharge the battery pack
Wheels	2	3D-Printed wheels
Wheel mounting component	2	Full guide with safety, setup and activity instructions
User Manual	1	Full guide with safety, setup, and activity instructions

· [Setup Instructions](#)

## Getting Started

### Powering On

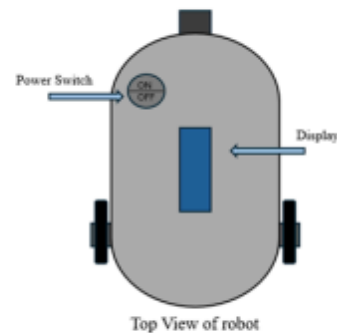
- Ensure the 12V LiPo battery is charged.
- Battery charge level is indicated by LED light
- Connect robot to charger until LED is green indicating the robot is fully charged
- Once robot is fully charged, disconnect from charger. (Do not leave it connected)



## Getting Started

### Powering On

- Press the power button located on top of the robot.
- The Display will turn on.
- The robot is ready to be paired with Bluetooth device.

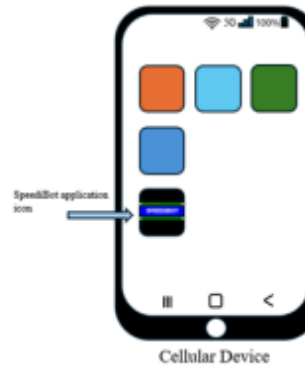


· [Bluetooth App Setup](#)

## Pairing with Bluetooth device

### Bluetooth pairing with robot

- Enable Bluetooth on cellular device.
- Ensure the “SpeediBot” application is installed on android/ios device.
- Once installed open the application, after a few seconds the user interface will be displayed.



## Pairing with Bluetooth device

### Bluetooth pairing with robot

- On the user interface tap on “scan for devices” after a few seconds the device will pair with the robot automatically. This can be confirmed when the interface displays “Speedibot found” in the top right. (Only one device can be connected at a time to the robot)
- Once the robot and device are paired speed commands can now be sent.



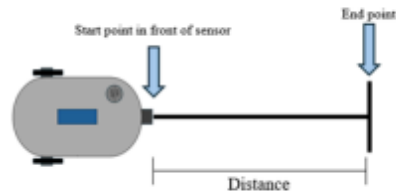
· [Step-by-Step Operation Guide](#)



## Operating the robot

### Starting a run

- Place a straight black line on the ground (greater than 1 meter)
- Place a horizontal black line on the original straight line, this indicates the end point of the track and where the robot will stop. (horizontal black line should be greater than the width of the sensor module)
- Place the robot on the line.
- Ensure the sensors are centered on the black line.



Top-down view of robot on the line

## Operating the robot

### Starting a run

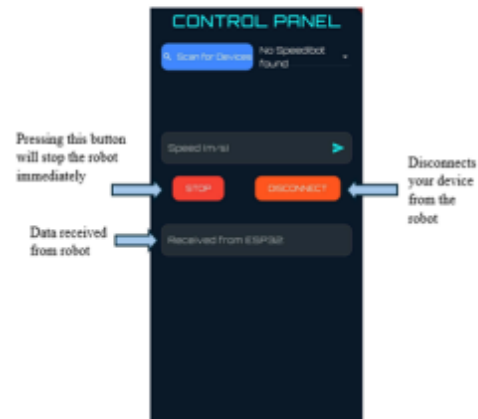
- On the user interface tap the location on the screen that reads "Speed (m/s)"
- Input a speed between 0.1 m/s and 1 m/s
- Then tap the blue arrow to the right, this will send the value to the robot
- The robot will then start to move until it reaches the horizontal black line.
- Once the robot stops it will calculate and display the average speed



## Operating the robot

### Starting a run

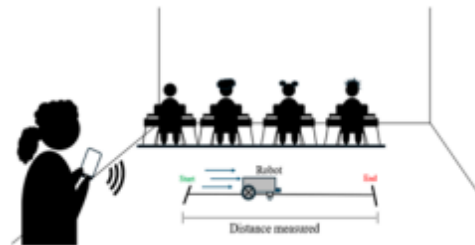
- The robot will display its average speed calculation on the user interface and on the robot display.
- Pressing the “STOP” button will stop the robot if needed.
- Pressing the “DISCONNECT” button will unpair the connected device from the robot



## Operating the robot

### Starting a run

- To do another run pick up the robot and place at desired distance and input another speed.
- For best results run the robot on level surfaces and environments with good lighting.



- Troubleshooting Guide
  - Common issues and Fixes

If the robot does not move after powering on, ensure that it has been properly charged and that a speed value has been sent via the Bluetooth app. If it fails to follow the black line, check that the sensor array is clean and unobstructed, and that the surface is well-lit and clearly marked.

- Maintenance & Care

- Cleaning and Storage

To maintain optimal performance, regularly inspect and clean the robot. Wipe the wheels and line sensors with a soft, dry cloth to remove dust or debris that could interfere with motion or detection. Avoid using water or harsh cleaning chemicals, especially near electronic components. When not in use, store the robot in a dry, cool location away from direct sunlight, moisture, and magnetic interference. Keeping the robot protected will extend the lifespan of both mechanical and electronic parts.

- Battery Tips

Charge the 12V LiPo battery only with a compatible charger and never leave it unattended while charging. Always check the charger's LED status to confirm charging and completion. Avoid discharging the battery completely, as this may reduce its capacity or damage it. When storing the robot for extended periods, disconnect the battery and store it at around 50% charge in a fireproof container, if possible.

- Contact Info / Support

If you encounter issues not covered in this manual or need technical assistance, feel free to reach out to our support team.

Support Contact:

Email: [jesus.rios08@utrgv.edu](mailto:jesus.rios08@utrgv.edu)

[juan.vega02@utrgv.edu](mailto:juan.vega02@utrgv.edu)

[aaron.jackson01@utrgv.edu](mailto:aaron.jackson01@utrgv.edu)

[denisse.galvan01@utrgv.edu](mailto:denisse.galvan01@utrgv.edu)

## 6.7 Completed Application

The application for the line-following robot reached completion after integrating all planned features and undergoing testing. The user is first greeted by a splash screen which welcomes the user and displays the name of the robot. The splash screen will transition to the control panel screen which displays all the needed features. The control panel screen consists of scanning and discovery of devices by using a button to scan and display the device on the dropdown menu. The app would then be connected to the robot and be ready to transmission user inputs. The app would let the user send speed commands to the robot and the user would be able to see the real-time data feedback being displayed on the control panel screen. The user would also be allowed to stop the robot if any malfunction happens and be able to disconnect the Bluetooth connection from the robot.

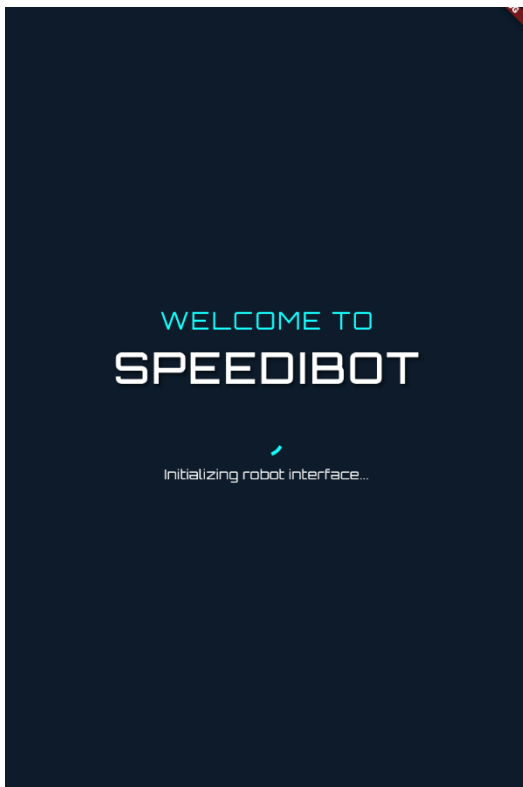


Figure 37. Final Design of Splash Screen

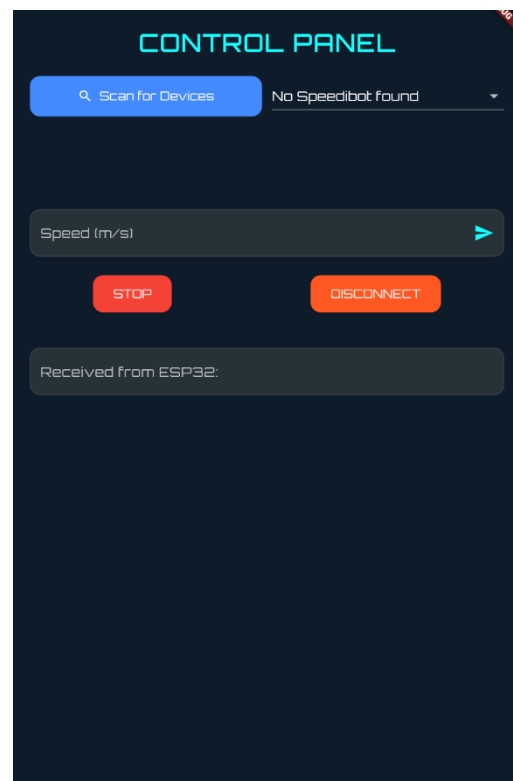


Figure 38. Final Design of Control Screen

## **7. Project Criteria**

### **7.1 Economic**

The economics of this project were carefully considered to ensure that the educational robot would be an affordable and sustainable solution for K-12 classrooms. The cost effectiveness of the design was achieved by selecting readily available and budget-friendly components, such as the VNH5019 Motor Driver, 12 V DC motors, and the ESP32 microcontroller. These components strike a balance between functionality and affordability, making the project accessible to schools with limited funding for STEM initiatives. Additionally, the robot's modular design and use of durable materials ensure minimal maintenance costs and longevity, reducing the need for frequent repairs or replacements. By leveraging Bluetooth connectivity and an LCD screen for data display, we avoided the expense of more complex hardware solutions, keeping the total project cost within a manageable range. This economic feasibility not only makes the robot more attractive to educators but also encourages widespread adoption, providing students from diverse backgrounds the opportunity to engage in hands-on learning with an innovative and practical tool.

### **7.2 Health & Safety**

Health and safety were central considerations in the design of this educational robot to ensure it is suitable for use in K-12 environments. The robot's compact and lightweight design

minimizes risks of physical injury, while rounded edges and a stable chassis prevent accidental tipping or sharp-edge hazards. Electrical safety was prioritized by properly insulating all wiring and incorporating components with built-in protection, such as the VNH5019 Motor Driver, which includes safeguards against overcurrent and overheating. The system operates at a safe voltage level (12V), reducing the risk of electrical shock. Additionally, the robot's motors and sensors are securely mounted to avoid loose parts that could pose choking hazards for younger students. Software safety was also addressed, as the robot's programming includes fail-safes to stop motion automatically upon detecting obstacles, preventing unintended collisions. Overall, these measures ensure that the robot is safe for hands-on use by students of all ages, fostering worry-free learning environment while encouraging exploration and creativity

## **8. Project Summary**

The Educational Robot K–12 project was developed to create an interactive, Bluetooth-controlled robotic platform aimed at enhancing STEM education for K–12 students. The robot was designed to demonstrate motion concepts such as speed, acceleration, and control systems through real-time feedback and user interaction. The system integrates an ESP32-S3 microcontroller, QTR line sensors, motor encoders, an LCD screen, and Bluetooth connectivity via a custom mobile application. This allows users to send speed commands, observe movement, and receive immediate feedback on performance metrics like speed and distance.

Throughout the design and development process, the team faced and overcame several technical challenges, including motor synchronization issues, unreliable speed readings due to a low-

performance L298N motor driver, and encoder calibration discrepancies. By upgrading to a dual VNH5019 motor driver and refining the encoder feedback system, the robot achieved significantly improved speed accuracy and stability. A PI control algorithm was implemented to ensure consistent motor performance, and voltage tests confirmed balanced outputs across various duty cycles.

Final testing confirmed that the robot could accurately track commanded speeds across multiple trials, with minimal deviation between encoder-based speed measurements and calculated physical speeds. Settling times varied slightly between trials, particularly at low-speed inputs, but overall accuracy remained within a 0.02 m/s tolerance.

The project culminated in a fully functional educational robot that supports real-time speed monitoring, mobile app communication, and stable motion control—delivering a hands-on learning tool for students and a successful capstone experience for the design team. Future work may focus on optimizing speed transitions, refining Bluetooth feedback, and expanding the user interface to include more detailed performance metrics.

## 9. References

- [1] A. M. Research, "NASA GISS Research Paper," Rutgers University, 2019. [Online]. Available: <https://njsgc.rutgers.edu/sites/default/files/nasa-giss-research-paper-2019.pdf>.
- [2] Engineering and Robotics Club, "Exploring Robotics in STEM Education," YouTube, 2023. [Online]. Available: <https://www.youtube.com/watch?v=-PCuDnpgiew>.
- [3] Education Insights, "Classroom Robotics for Learning," YouTube, 2024. [Online]. Available: <https://youtu.be/gwFE-NkzR-k?si=VKZhHkMg8vSP7ZH>.
- [4] A. Smith and B. Johnson, "The Effects of Robotics Professional Development on Science and Mathematics Teaching Performance and Student Achievement in Underserved Middle Schools," *Contemporary Issues in Technology and Teacher Education*, vol. 21, no. 4, pp. 123 140, 2021. [Online]. Available: <https://citejournal.org/volume-21/issue-4-21/mathematics/the-effects-of-robotics-professional-development-on-science-and-mathematics-teaching-performance-and-student-achievement-in-underserved-middle-schools/>.
- [5] National Center for Science and Engineering Statistics, "Student Learning in Mathematics and Science," 2023. [Online]. Available: <https://nces.nsf.gov/pubs/nsb202331/student-learning-in-mathematics-and-science>.
- [6] National Science Board, "Instructional Technology and Digital Learning," 2018. [Online]. Available: <https://www.nsf.gov/statistics/2018/nsb20181/report/sections/elementary-and-secondary-mathematics-and-science-education/instructional-technology-and-digital-learning>.
- [7] S. Johnson et al., "The Role of Robotics in STEM Education: A Review of Applications in K-12 Learning," 2020.



[8] S. Papert, Mindstorms: Children, Computers, and Powerful Ideas. New York, NY: Basic Books, 1980.

[9] LEGO Education, "LEGO Mindstorms EV3," 2022. [Online]. Available: [www.legoeducation.com](http://www.legoeducation.com).

[10] VEX Robotics, "VEX Robotics Platforms," 2022. [Online]. Available: [www.vexrobotics.com](http://www.vexrobotics.com).

[11] Makeblock, "mBot Educational Robotics Kit," 2022. [Online]. Available: [www.makeblock.com](http://www.makeblock.com).

## 10. Appendix

### SPEEDIBOT APP CONTROL SCREEN

```
import 'package:flutter/material.dart';
import '../services/bluetooth_service.dart';
import 'package:flutter_blue_plus/flutter_blue_plus.dart' as blue;

class ControlScreen extends StatefulWidget {
  const ControlScreen({super.key});

  @override
  ControlScreenState createState() => ControlScreenState();
}

class ControlScreenState extends State<ControlScreen> {
  final BluetoothService bluetoothService = BluetoothService();
  final TextEditingController speedController = TextEditingController();
  //String receivedMessage = " ";
  blue.BluetoothDevice? _selectedDevice;

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) async {
      await bluetoothService.requestBluetoothPermissions();
      bluetoothService.scanForDevices();
    });
  }

  void sendSpeed() {
    String speedText = speedController.text.trim();
    double? parsedSpeed = double.tryParse(speedText);
    if (speedText.isNotEmpty && parsedSpeed != null) {
      bluetoothService.sendSpeed(parsedSpeed.toStringAsFixed(2));
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Enter a valid speed value!")),
      );
    }
  }

  void stopSpeed(){
    bluetoothService.sendSpeed(0); // Send 0 to stop the motor
  }
}
```

```

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text("Robot has stopped")),
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFF0D1B2A),
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Column(
            children: [
              Text(
                "CONTROL PANEL",
                style: TextStyle(
                  fontFamily: 'Orbitron',
                  fontSize: 28,
                  color: Colors.cyanAccent,
                  fontWeight: FontWeight.bold,
                  letterSpacing: 2,
                ),
              ),
              const SizedBox(height: 20),
              Row(
                children: [
                  Expanded(
                    //flex: 1,
                    child: ValueListenableBuilder<bool>(
                      valueListenable: bluetoothService.isConnected,
                      builder: (context, isConnected, child) {
                        return ElevatedButton.icon(
                          onPressed: () => bluetoothService.scanForDevices(),
                          icon: Icon(isConnected ? Icons.bluetooth_connected : Icons.search, color: Colors.white),
                          label: Text(
                            isConnected ? 'Connected' : 'Scan for Devices',
                            style: const TextStyle(fontFamily: 'Orbitron', color: Colors.white,)),
                        ),
                      ),
                    ),
                  ),
                  const SizedBox(width: 12),
                  Expanded(
                    //flex: 2,
                    child: ValueListenableBuilder<List<ValueNotifier<blue.BluetoothDevice>>>(
                      valueListenable: bluetoothService.bluetoothList,
                      builder: (context, deviceList, child) {
                        // Filter the list to only include devices named "Speedibot"
                        final filteredDevices = deviceList.where((notifier) {
                          String deviceName = notifier.value.platformName;
                          return deviceName.toLowerCase().contains("speedibot");
                        }).toList();

                        if (filteredDevices.isEmpty) {
                          return DropdownButton<blue.BluetoothDevice>(
                            hint: Text('No Speedibot found', style: TextStyle(color: Colors.white, fontFamily: 'Orbitron')),
                            value: null,
                            isExpanded: true,
                            items: [],
                            onChanged: null,
                          );
                        }
                      ),
                    ),
                    child: DropdownButtonFormField<blue.BluetoothDevice>(
                      dropdownColor: Colors.black87,
                      style: const TextStyle(color: Colors.white, fontFamily: 'Orbitron'),
                      decoration: InputDecoration(
                        filled: true,

```

```

        items: [],
        onChanged: null,
      );
    }
    return DropdownButtonFormField<blue.BluetoothDevice>(
      dropdownColor: Colors.black87,
      style: const TextStyle(color: Colors.white, fontFamily: 'Orbitron'),
      decoration: InputDecoration(
        filled: true,
        fillColor: Colors.blueGrey.shade900,
        border: OutlineInputBorder(borderRadius: BorderRadius.circular(12)),
      ),
      hint: Text('Select Speedibot', style: TextStyle(color: Colors.white70)),
      value: _selectedDevice,
      isExpanded: true,
      items: filteredDevices.map((notifier) {
        blue.BluetoothDevice device = notifier.value;
        return DropdownMenuItem<blue.BluetoothDevice>(
          value: device,
          child: Text(device.platformName),
        );
      }).toList(),
      onChanged: (blue.BluetoothDevice? device) {
        setState(() {
          _selectedDevice = device;
        });
        if (device != null) {
          bluetoothService.connectToDevice(device);
        }
      },
    );
  },
),
),
],

```

Figure 39 SpeeddiBot app control screen code

## Bluetooth Communication

```
import 'dart:convert';
import 'package:flutter_blue_plus/flutter_blue_plus.dart' as blue;
import 'package:flutter/foundation.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:logger/logger.dart';
import 'package:flutter/material.dart';
import '../screens/control_screen.dart';

final logger = Logger();

class BluetoothService {
  blue.FlutterBluePlus flutterBlue = blue.FlutterBluePlus();
  blue.BluetoothDevice? targetDevice;
  blue.BluetoothCharacteristic? targetCharacteristic;

  final String serviceUUID = "12345678-1234-5678-1234-56789abcdef0";
  final String characteristicUUID = "87654321-4321-6789-4321-fedcba987654";
  /// final String serviceUUID = "12345678-1234-1234-1234-123456789012";
  ///final String characteristicUUID = "87654321-4321-4321-4321-210987654321";

  final ValueNotifier<bool> isConnected = ValueNotifier(false);
  final ValueNotifier<List<ValueNotifier<blue.BluetoothDevice>>> bluetoothList = ValueNotifier([]);
  final ValueNotifier<blue.BluetoothDevice?> selectedDevice = ValueNotifier(null);
  final ValueNotifier<String> receivedData = ValueNotifier("");

  // Request necessary Bluetooth permissions
  Future<void> requestBluetoothPermissions() async {
    if (await Permission.bluetoothScan.request().isGranted &&
        await Permission.bluetoothConnect.request().isGranted &&
        await Permission.locationWhenInUse.request().isGranted) {
      logger.w("Bluetooth and Location permissions granted!");
    } else {
      logger.e("Bluetooth or Location permissions denied!");
    }
  }

  // Scan for all available Bluetooth devices
  Future<void> scanForDevices() async {
    bluetoothList.value.clear();
  }
}
```

```

// Final String serviceUUID = "12345678-1234-1234-1234-123456789012" ;
// Scan for all available Bluetooth devices
Future<void> scanForDevices() async {
  bluetoothList.value.clear();
  bluetoothList.notifyListeners(); // Force UI update

  await blue.FlutterBluePlus.startScan();

  blue.FlutterBluePlus.scanResults.listen((results) {
    for (var result in results) {
      if(!bluetoothList.value.any((device) => device.value.remoteId == result.device.remoteId)) {
        bluetoothList.value.add(ValueNotifier(result.device));
        bluetoothList.notifyListeners();
        logger.i("Found Speedibot: ${result.device.platformName}");
      }
    }
  });

  // Optional: Stop scan after 10 seconds
  Future.delayed(Duration(seconds: 25), () {
    blue.FlutterBluePlus.stopScan();
    logger.w("Stopped scanning after 25 seconds.");
  });
}

// Connect to a selected device
Future<void> connectToDevice(blue.BluetoothDevice device) async {
  try {
    targetDevice = device;
    await targetDevice!.connect();
    targetDevice!.connectionState.listen((state) {
      if (state == blue.BluetoothConnectionState.disconnected) {
        logger.w("Device disconnected unexpectedly");
        isConnected.value = false;
      }
    });
  }

  List<blue.BluetoothService> services = await device.discoverServices();
  for (blue.BluetoothService service in services) {

```

```

// Final String serviceUuid = "12345678-1234-1234-1234-123456789012";
// Scan for all available Bluetooth devices
Future<void> scanForDevices() async {
  bluetoothList.value.clear();
  bluetoothList.notifyListeners(); // Force UI update

  await blue.FlutterBluePlus.startScan();

  blue.FlutterBluePlus.scanResults.listen((results) {
    for (var result in results) {
      if(!bluetoothList.value.any((device) => device.value.remoteId == result.device.remoteId)) {
        bluetoothList.value.add(ValueNotifier(result.device));
        bluetoothList.notifyListeners();
        logger.i("Found Speedibot: ${result.device.platformName}");
      }
    }
  });

  // Optional: Stop scan after 10 seconds
  Future.delayed(Duration(seconds: 25), () {
    blue.FlutterBluePlus.stopScan();
    logger.w("Stopped scanning after 25 seconds.");
  });
}

// Connect to a selected device
Future<void> connectToDevice(blue.BluetoothDevice device) async {
  try {
    targetDevice = device;
    await targetDevice!.connect();
    targetDevice!.connectionState.listen((state) {
      if (state == blue.BluetoothConnectionState.disconnected) {
        logger.w("Device disconnected unexpectedly");
        isConnected.value = false;
      }
    });
  });

  List<blue.BluetoothService> services = await device.discoverServices();
  for (blue.BluetoothService service in services) {

```

```

    } else {
        logger.w("Bluetooth not connected!");
    }
}

void sendStop() {
    sendSpeed(0); // Call sendSpeed with 0 to stop the motor
}

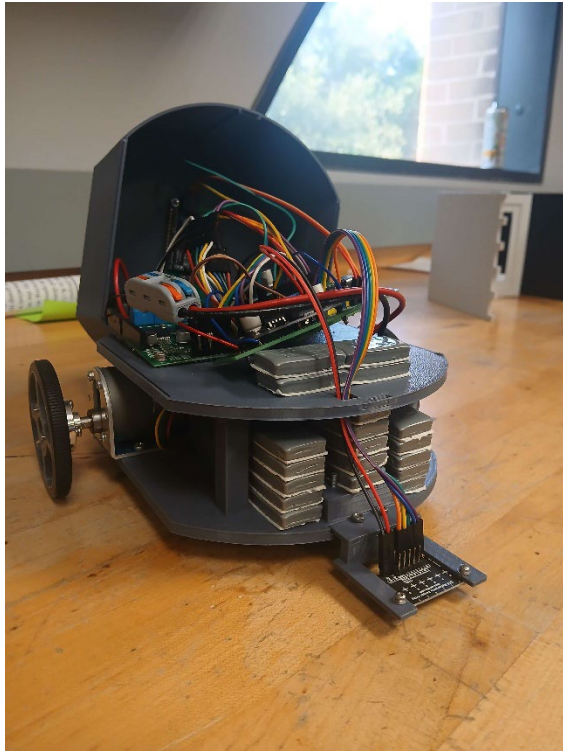
// Disconnect from the device
Future<void> disconnectFromDevice() async {
    if (targetDevice != null && targetCharacteristic != null) {
        await targetCharacteristic!.setNotifyValue(false);
        await targetDevice!.disconnect();
        targetDevice = null;
        targetCharacteristic = null;
        isConnected.value = false;
        logger.w("Disconnected from device");
    } else {
        logger.w("No device connected.");
    }
}

// Add this to your BluetoothService class to handle reconnection
Future<void> attemptReconnect() async {
    if (targetDevice != null && !isConnected.value) {
        logger.w("Attempting to reconnect...");
        try {
            await connectToDevice(targetDevice!);
        } catch (e) {
            logger.e("Reconnection failed: $e");
            // Try again after a delay
            Future.delayed(Duration(seconds: 3), () => attemptReconnect());
        }
    }
}

```

Figure 40 Bluetooth Communication code





*Figure 418 Interior of SpeediBot*

The Figure to the left shows the internal components of the robot, including the main control system and wiring layout. An ESP32 microcontroller is mounted on the top platform, connected to various sensors and motor drivers through color-coded jumper wires. The middle section houses the stacked DC-DC converters that regulate power distribution to different components. At the front of the robot, the QTR reflectance sensor is mounted and wired for surface detection during navigation. The organized internal layout ensures stable mechanical structure,

efficient power management, and reliable sensor and motor operation.

The figure to the right shows the fully assembled robot with its protective outer cover installed. The cover shields the internal components, such as the microcontroller, wiring, and power electronics, from external damage and environmental interference. The compact and enclosed design improves durability while maintaining easy access to external features like the



*Figure 427 Exterior of SpeediBot*

wheels, the front-mounted QTR sensor array, and the LCD display on top for system feedback.