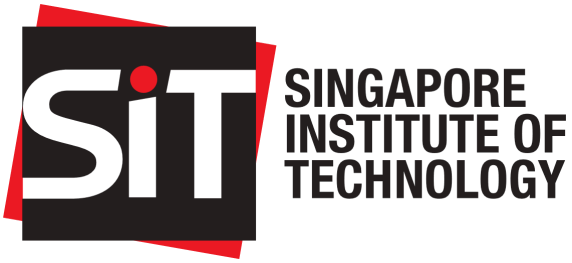

Design Project – Final Report

PathPilot: Autonomous Course Follower

Joseph Selva Raj

2102883

Academic Supervisor:
Dr Xin Lou

Submitted as part of the requirements for TLM3001 Design Project

SINGAPORE INSTITUTE OF TECHNOLOGY

Design Project Report Submission Form

Declaration of Authorship

Name of candidate: **Joseph Selva Raj**	Student ID Number: **2102883**
Degree: **BEng (Hons) Computer Engineering**	
Design Project Title: **PathPilot: Autonomous Course Follower**	
Academic Supervisor(s): **Dr Xin Lou**	Cluster: **Infocomm Technology**
Industry Supervisor: (*if applicable*)	Organisation: (*if applicable*)

I hereby confirm that:

1. this work was done wholly or mainly while in candidature for a degree programme at SIT;
2. where any part of this design project has been previously submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. I have acknowledged the use of all resources in the preparation of this report;
4. the **report ~~contains~~** / **does not contain** patentable or confidential information *(strike through whichever does not apply)*;
5. the work was conducted in accordance with the research integrity policy and ethics standards of SIT and that the research data are presented honestly and without prejudice. The SIT Institutional Review Board (IRB) approval number is _____ *(where applicable)*;
6. I have read and understood the University's definition of plagiarism as stated in the SIT Academic Policies Scheme 14: Academic Integrity.

   Plagiarism is the copying, using or passing off of another's work as one's own work without giving credit to the author or originator, and also includes self-plagiarism. For example, reusing, wholly or partially, one's previous work in another context without referencing its previous use.

Joseph

_____ 25 July 2024

Signature of Student **Date**

Abstract

The PathPilot project presents an innovative approach to autonomous navigation by addressing the limitations of traditional systems like SLAM (Simultaneous Localization and Mapping) and line-following robots, which often involve high costs, complex setups, and environmental dependencies. It integrates advanced machine learning techniques with cost-effective, resource-constrained embedded systems to offer a scalable solution for diverse applications. Structured into two phases, the project first gathers training data through user-controlled robot navigation using an Android application and a Light Detection and Ranging (LiDAR) sensor. This data is then used to train a machine learning model with Python libraries, including Scikit-Learn and XGBoost, resulting in a robust classifier for real-time decision-making. Key software components include the Android application for user interaction, Arduino software for data collection, and LidarPlayback for data visualization. In the Testing Phase, the trained model is deployed on an Arduino microcontroller, guiding the robot based on real-time sensor inputs and utilizing obstacle detection through both LiDAR and an ultrasonic sensor. The integration of sophisticated software and machine learning algorithms within an economical hardware setup positions PathPilot as an efficient, cost-effective approach in the field of autonomous systems.

Table of contents

1. Introduction

This project endeavours to develop an autonomous robot navigation system designed to navigate predefined courses using resource-constrained embedded systems. By harnessing machine learning algorithms on an efficiency-optimized hardware platform, the robot aims to achieve navigation with efficiency and reliability. Central to this endeavour is the cultivation of robust perception capabilities, enabling the system to adeptly address obstacles and adapt to dynamic environmental changes, thereby ensuring a resilient and perceptive navigation experience.

The motivation for this project arises from the limitations present in contemporary autonomous robot navigation systems, such as line followers and SLAM. Line followers, though relatively accurate, face challenges such as complex PID (Proportional-Integral-Derivative) tuning and environmental sensitivity, limiting their broader applicability. Similarly, SLAM, while effective, incurs high financial and computational expenses due to the integration of multiple sensors, hindering its adoption in cost-effective applications.

The existing limitations in contemporary autonomous robot navigation systems act as significant barriers, constraining their capabilities and hindering widespread adoption across various sectors. There is a clear need for a robot that not only is financially accessible, thus expanding its availability to a broader spectrum of users and applications, but also exhibits essential qualities of adaptability and robustness. This adaptability is crucial to ensure the system can effectively navigate through changing environmental conditions and detect obstacles. Furthermore, there is a growing demand for an autonomous navigation system that operates independently and flexibly, without relying on external environmental cues. Such a system would enable robots to seamlessly function in diverse environments without requiring alterations to the surroundings, thereby greatly improving their overall usability and adaptability across a wide range of scenarios.

By addressing these challenges head-on, this project aims to redefine the landscape of autonomous robot navigation, offering a cost-effective solution without the need for high computational resources. Through a synergy of pioneering algorithms and hardware refinements, the envisioned navigation system aspires to transform the field, unlocking new possibilities for application in diverse industries and domains.

2. Literature Research

In the contemporary era, machines have assumed responsibility for physical tasks previously carried out by humans, with the primary objective of mitigating the labour burden. However, there is an escalating demand for machines to not only execute physical tasks but also possess cognitive abilities, enabling them to think and make decisions akin to human beings.

Within the realm of robotics, autonomous navigation systems play a pivotal role in enabling a robot to traverse a course autonomously, without requiring human input or oversight. The efficacy of navigation systems is inherently linked to the sensors integrated into the robot and the structural characteristics of the environment (McKerrow, 1991).

2.1 Robot Perception Sensors

Robotic perception entails the utilization of sensors and other technological tools to gather and analyse data concerning the robot's surroundings, encompassing people, objects and other robots. These sensory inputs originate from a diverse array of sources, including cameras and laser sensors. Machine learning algorithms are then employed to process this data, empowering the robot to make informed decisions based on the collected data. Through perceiving and comprehending their environment, robots achieve autonomy and engage in meaningful interactions within their surroundings (Robot Perception and Sensing, 2023).

Ultrasonic sensors

Ultrasonic sensors play a pivotal role in the landscape of robotic perception, providing a dependable method for detecting obstacles and navigating environments with precision. These sensors operate by emitting high frequency sound waves and measuring the time it takes for the waves to reflect back after encountering an object, as shown in Figure 1. This mechanism grants ultrasonic sensors the ability to deliver accurate distance measurements in real-time, making them indispensable tools for robot perception tasks (Robocraze, 2022).

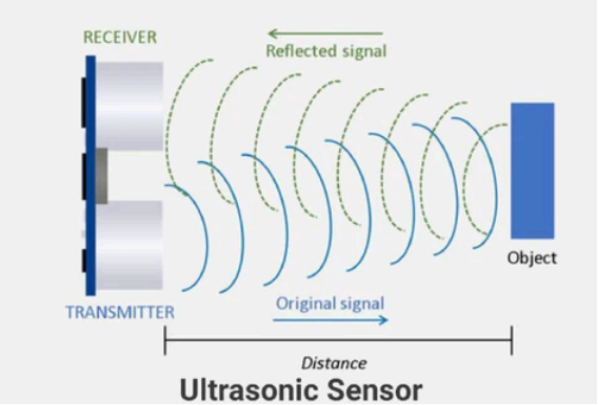

Figure 1: Working principle of ultrasonic sensors (Robocraze, 2022).

These sensors are independent of environmental conditions, unlike optical image sensors such as a camera. They perform reliably in diverse settings, including low-light environments. Their ability to detect obstacles ensures efficient navigation and collision avoidance, enhancing safety and stability in robotic applications.

However, ultrasonic sensors have limitations, such as a narrow field of view, leading to blind spots, and potential inaccuracies from wave reflection, absorption, or diffusion by materials like fabric. These challenges necessitate careful consideration of the operating environment to ensure optimal performance.

Cameras

In vision-guided robotics, cameras are crucial for enhancing perception, enabling robots to understand their surroundings in detail. Cameras play a central role in both Computer Vision and Robot Vision. While Computer Vision has broad applications across various domains, Robot Vision is specifically tailored to empower robots in perceiving and interacting with their environment.

Robot Vision integrates seamlessly with robotic hardware and control systems, requiring swift perception and decision-making in real-time scenarios. In contrast, Computer Vision often operates independently of physical systems, allowing for tasks to be performed offline or with relaxed time constraints. Using a diverse array of sensors, including cameras, depth sensors, and LiDAR, Robot Vision captures a comprehensive understanding of the environment, essential for navigating dynamic landscapes with precision and efficiency. Its closed-loop control system provides real-time feedback for precise robot control, a capability lacking in Computer Vision's predominantly indirect feedback loop (Pm, 2023).

Cameras, ranging from RGB to depth sensors, capture visual data critical for object identification, movement tracking, and environmental analysis. Their versatility in capturing both colour and depth information equips robots to make informed decisions in navigation and object interaction (Yang, 2024).

However, cameras also present challenges. Processing vast amounts of visual data demands significant computational resources, potentially straining the robot's onboard processing capabilities. Variations in lighting conditions can affect image quality, impacting the accuracy of perception algorithms and hindering navigation performance (AZoRobotics, 2023). In low-light conditions, robots relying on cameras for perception use additional light sources that increase power consumption, as illustrated in Figure 2.

Figure 2: Utilization of an onboard light source in a camera-based perception system (ECOVACS US, 2024).

Light Detection and Ranging (LiDAR) sensors

LiDAR sensors represent a cutting-edge solution in the realm of robotic perception, offering unique advantages over traditional sensors like ultrasonic sensors and cameras. LiDAR sensors operate by emitting laser pulses and measuring the time it takes for the pulses to reflect back from objects in the environment, enabling precise distance measurements and 2D mapping of surroundings (Velodyne LiDAR , 2022).

Unlike ultrasonic sensors, which have limited field of view and can only measure distances in specific directions, LiDAR sensors offer 360-degree coverage, allowing robots to perceive objects from all directions without blind spots. This omnidirectional perception capability is invaluable for tasks such as mapping, navigation, and obstacle avoidance, enabling robots to

navigate complex environments with unparalleled precision and safety (Marie, 2024). Figure 3 shows an omnidirectional LiDAR sensor similar to the one used in the PathPilot robot.

Figure 3: An omnidirectional LiDAR sensor (RPLiDAR , 2024).

Moreover, LiDAR sensors are also robust against environmental conditions that affect other sensors. Unlike cameras, which struggle in low-light conditions, LiDAR sensors operate independently of ambient light, ensuring consistent and reliable performance in diverse environments. This makes them particularly well-suited for applications where lighting conditions vary (Why LiDAR Is Ideal, 2021).

Despite their advantages, LiDAR sensors have some limitations. They are relatively high cost compared to other sensors, which may limit their widespread adoption. Additionally, LiDAR sensors may struggle to detect transparent objects or highly reflective surfaces, potentially leading to inaccuracies in perception (Issuu, 2023).

The unique capabilities and advantages of LiDAR sensors make them an increasingly popular choice for robotic perception systems, particularly in applications where accuracy, reliability, and omnidirectional perception are paramount. With ongoing advancements in LiDAR technology and continued research into overcoming its limitations, LiDAR sensors are poised to play a central role in the future of robotic navigation and perception (LiDAR Mag, 2019).

2.2 Existing Autonomous Robot Navigation Systems

Popular existing autonomous robot navigation systems include line following and SLAM.

Line followers

Line followers, a prevalent autonomous robot navigation method, undergo four key stages as shown in Figure 4: sensor detection (using infrared sensors or photodiodes), signal processing by a microcontroller, motor control adjusting speed and direction, and PID feedback for precise movement and accurate line following.

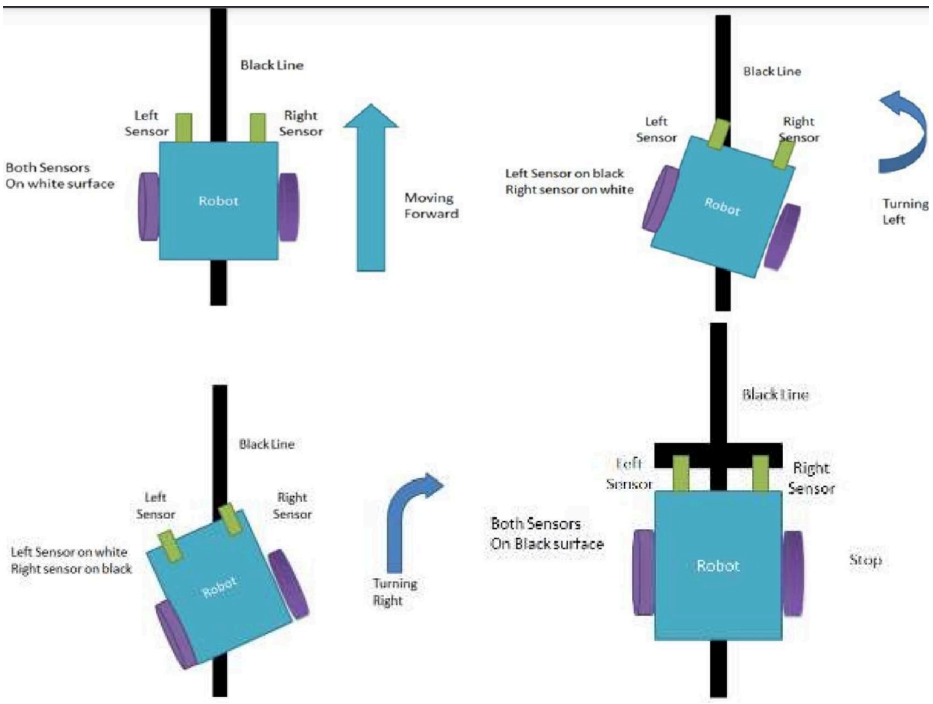

Figure 4: Working principle of a line follower robot (Line Follower Robot, 2021).

Despite its relative accuracy, challenges like complex PID tuning and environmental sensitivity limit line followers' broader applicability . The tuning process requires control theory knowledge, and environmental factors necessitate frequent adjustments, posing instability risks (Line Follower Robot, 2021). Moreover, line followers depend on pre-installed lines, and their visibility is sensitive to environmental factors such as dirt and shadows. Additionally, they exhibit limited adaptability to obstacles. These drawbacks highlight the necessity for more versatile and robust robotic navigation systems.

Simultaneous Localization and Mapping (SLAM)

SLAM is another autonomous robot navigation system that is gaining popularity. SLAM is the process of creating a map of the environment while simultaneously determining the robot's position and orientation within that map, allowing it to navigate around its environment or course. Existing autonomous robots that utilize SLAM, such as autonomous robotic vacuum cleaners, are typically implemented with a LiDAR sensor and a camera vision system.

These robots use LiDAR sensors to capture 3D point clouds of the environment and estimate the robot's position and orientation. The point cloud is then processed using a SLAM algorithm, which estimates the robot's position and orientation based on the distance measurements and other sensor data, such as the robot's velocity or acceleration.

The algorithm then uses this estimate to update the map of the environment and track the robot's movements in real-time. SLAM algorithm is conclusively based on the environment sensor readings. When no objects are in the correct position for the sensors to read them, the robot utilises its computer vision system to capture images and extract features from them to estimate the robot's position and orientation relative to its surroundings (Szczepaniak, 2022).

Despite its effectiveness, SLAM comes with certain drawbacks, notably in terms of cost, computational resources and susceptibility to environmental lightning conditions. The heightened financial and computational costs associated with the SLAM algorithm, integration of multiple sensors and systems in SLAM pose significant obstacles for its adoption as an autonomous navigation system in applications that prioritise cost-effectiveness and have computational resource constraints (Hookii, 2023).

Additionally, the reliance on environmental lighting for the computer vision system renders SLAM impractical in light-deprived settings. These factors collectively limit the feasibility of implementing SLAM in such contexts.

2.3 Supervised vs. Unsupervised Learning

Machine learning offers a powerful toolkit for enabling robots to perform tasks autonomously. There are two primary approaches to machine learning: supervised learning and unsupervised learning.

Supervised Learning

In supervised learning, the algorithm is trained on a labelled dataset. Each data point consists of an input and a corresponding output value, or label. This labelled data acts as a guide for the algorithm, allowing it to learn the relationship between the input and output. The model can then use this learned relationship to make predictions for new, unseen data. Common supervised learning tasks include classification, where the model predicts the category an input belongs to, for example in spam detection, and regression, where the model predicts a continuous value for application such as weather forecasting (What Is Supervised Learning? | IBM, 2023). The process for supervised learning is shown in Figure 5.

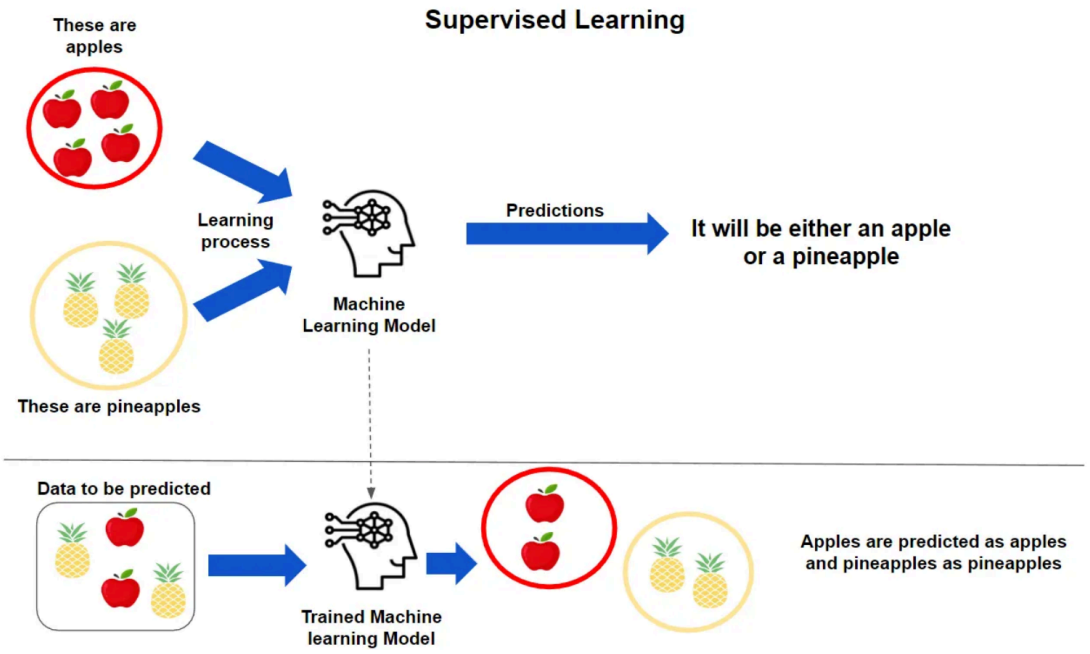

Figure 5: Supervised learning of a machine learning model (What Is Supervised Learning? | IBM, 2023).

Unsupervised Learning

In contrast to supervised learning, unsupervised learning involves training a model on data without any labelled responses. The goal of unsupervised learning is to uncover hidden patterns or intrinsic structures within the input data. Since there are no predefined labels to guide the learning process, the algorithm must independently discover the underlying structure of the data. The process for unsupervised learning is illustrated in Figure 6.

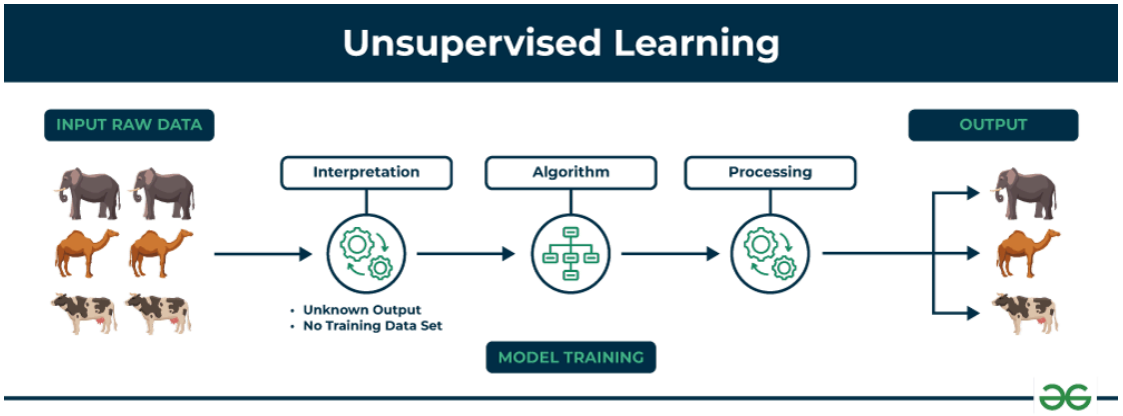

Figure 6: Unsupervised learning of a machine learning model (GeeksforGeeks, 2023).

Common tasks in unsupervised learning include clustering and dimensionality reduction. Clustering algorithms, such as k-means or hierarchical clustering, group similar data points together, which is useful in applications like customer segmentation or image compression. Dimensionality reduction techniques, such as Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbour Embedding (t-SNE), are employed to reduce the number of features in a dataset while preserving as much information as possible, aiding in data visualization and noise reduction (What Is Unsupervised Learning? | Google Cloud, 2024).

Unsupervised learning is particularly valuable when dealing with large datasets where manual labelling is impractical or impossible. By identifying patterns and relationships within the data, unsupervised learning can provide insights that might not be immediately apparent through traditional analysis methods.

2.4 Machine Learning for Resource-Constrained Embedded Systems

A specific focus was made on the relationship between machine learning and resource-constrained embedded systems when reviewing research on autonomous robot navigation for the PathPilot project. This concentrated investigation resulted in a more profound comprehension of the difficulties associated with implementing complex algorithms on constrained hardware platforms, setting the foundation for creative solutions during the PathPilot project's development stage.

Tiny Machine Learning (TinyML) is a field of study concerned with running machine learning models on resource-constrained devices like microcontrollers. This research area is particularly relevant for embedded applications where sensor data processing needs to occur

locally on the device due to resource constraints or real-time requirements. Unlike cloud-based approaches, TinyML allows devices to make smart decisions on their own, despite limited processing and memory (Braun, 2022). Figure 7 depicts the cons in the typical financial cost and resource requirement of cloud-based machine learning systems, which are much higher and costly to deploy as compared to TinyML systems designed for microcontrollers.

Table 1: Cloud & Mobile ML systems versus TinyML systems.

Platform	Architecture	Memory	Storage	Power	Price
Cloud E.g., Nvidia V100	GPU Nvidia Volta	HBM 16GB	SSD/disk TB–PB	250W	~$9,000
Mobile E.g., cellphone	CPU Arm Cortex-A78	DRAM 4GB	Flash 64GB	~8W	~$750
Tiny E.g., Arduino Nano 33 BLE Sense	MCU Arm Cortex-M4	SRAM 256KB	eFlash 1MB	0.05W	$3

Figure 7: Comparison of price and resource constraints between Cloud-Based ML Systems and TinyML Systems (Braun, 2022).

In their research paper, Immonen et al. (Immonen & Hämäläinen, 2022) investigates the relationship between TinyML and resource-constrained embedded systems, offering insights relevant to this project. They emphasize the challenge of adapting traditional machine learning models to microcontrollers, which have limited computational power and memory compared to typical processors. Microcontrollers usually provide only tens of kilobytes to a few megabytes of memory and operate at megahertz speeds, while traditional models require much more memory and processing power.

This resource gap leads to issues like slow performance, increased power consumption, and limited functionality when deploying models on microcontrollers, especially in critical tasks like robot navigation. To address this challenge, TinyML focuses on techniques to reduce model size and complexity, such as quantization, pruning, and clustering.

The paper highlights the significance of frameworks and libraries designed for TinyML, bridging the gap between Python-based model training and C code deployment on microcontrollers, streamlining development processes. Additionally, the paper presents benchmarking results using TinyML libraries such as emlearn, sklearn-porter, and

MicroMLgen on an Arduino Uno microcontroller. Decision Tree (DT) and Random Forest (RF) algorithms showed the best accuracy, lowest memory footprint, and fastest classification speed, while Multilayer Perceptron (MLP) demonstrated promising accuracy of 0.97 but exceeded the Arduino Uno's SRAM capacity for weights and biases with increasing network complexity.

There are several machine learning algorithms that are supported by the TinyML framework, which are as follows:

Decision Tree (DT)

DT is a fundamental machine learning algorithm that operates by recursively partitioning the input space into regions, assigning a simple model, typically a constant, to each region. These partitions create a tree-like structure where each internal node represents a decision based on one of the input features, and each leaf node corresponds to a class label or a numerical value (Dhasenfratz, 2017). Figure 8 provides a depiction of a DT functionality.

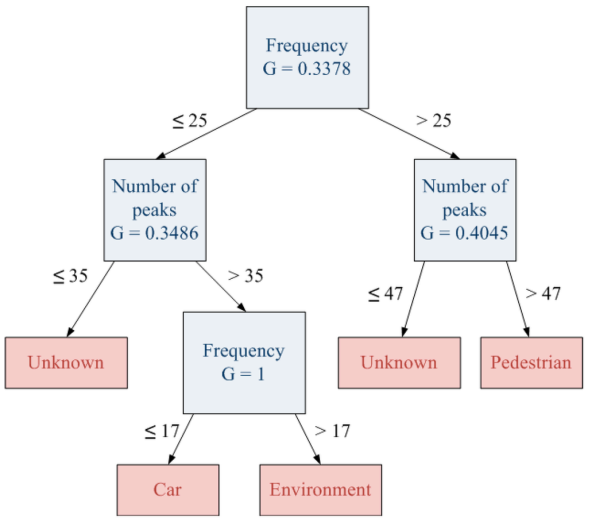

Figure 8: Functionality of a DT algorithm (Dhasenfratz, 2017).

DTs offer advantages in embedded systems. They are interpretable, facilitating debugging and validation. They have low computational cost during training and inference, suitable for systems with limited processing power. They are memory-efficient, crucial in resource-constrained environments. They excel in fast decision-making due to simple comparisons. Additionally, they handle noise and irrelevant features well, automatically selecting informative features. However, they can overfit, tailoring excessively to training

data, impacting performance on unseen instances (Machine Learning for Embedded Systems, 2018).

Random Forest (RF)

RF is a powerful tree learning technique that builds multiple decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Each tree in the Random Forest is trained on a random subset of the training data and a random subset of the features, which helps to decorrelate the trees and reduce overfitting. Figure 9 illustrates how the Random Forest Algorithm works.

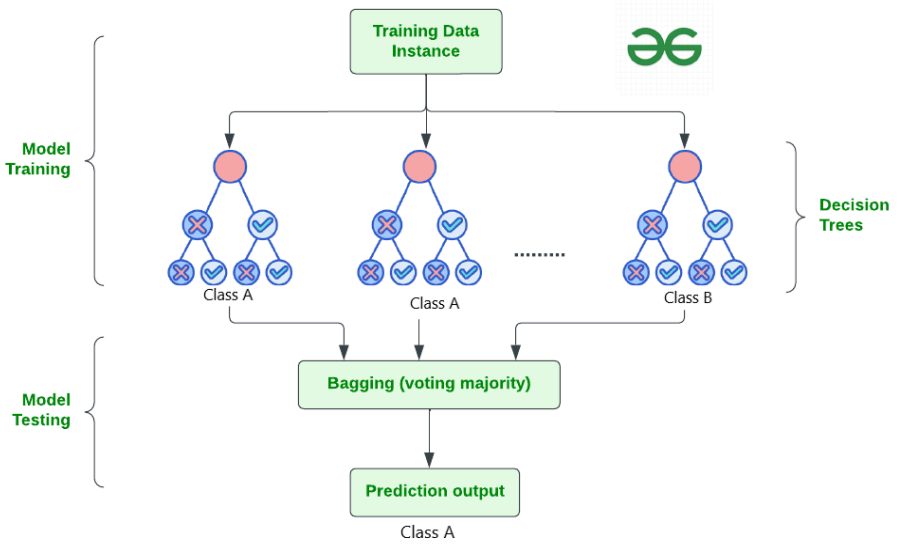

Figure 9: How a Random Forest Algorithm works (GeeksforGeeks, 2024).

Random Forests offer numerous advantages in embedded systems. Their ensemble nature often leads to enhanced accuracy and generalization, making them effective with complex or noisy datasets. They resist overfitting, valuable when training data is limited or noisy. They provide insights into feature importance, aiding in understanding data patterns. They handle missing data and outliers well, simplifying preprocessing and deployment in real-world applications. Moreover, they are easy to use and require minimal hyperparameter tuning compared to complex models like deep neural networks, enabling rapid prototyping and deployment in resource-constrained systems (GeeksforGeeks, 2024).

XGBoost (eXtreme Gradient Boosting)

XGBoost (eXtreme Gradient Boosting) is an advanced ensemble learning technique that builds multiple decision trees in a sequential manner, where each tree attempts to correct the errors made by the previous ones. Figure 10 illustrates the distinction between a single machine learning model and the Ensemble Learning technique, which is analogous to the mechanism of XGBoost.

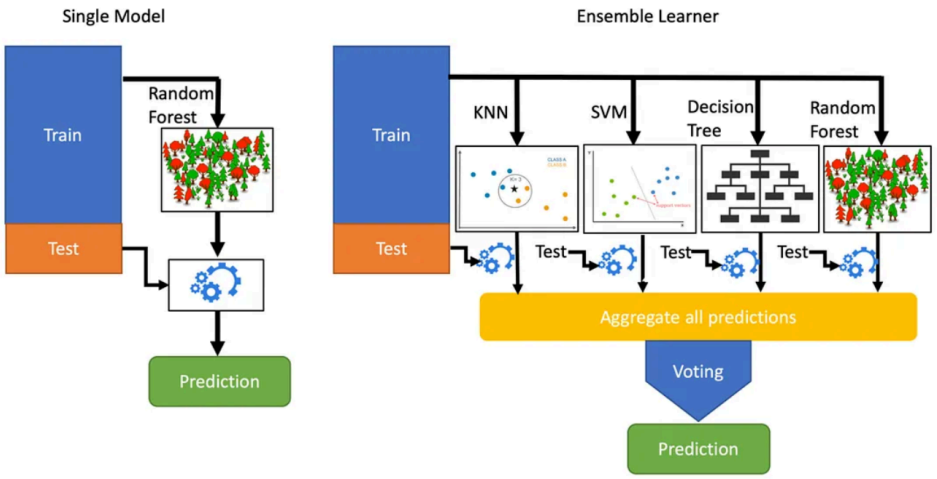

Figure 10: Single model vs Ensemble Learner technique (Malik, 2020).

XGBoost is built upon the gradient boosting framework, which combines the predictions of several base learners to enhance overall performance. The core idea is to sequentially add new models to the ensemble, each one correcting the errors made by the combined ensemble of all previous models. In XGBoost, the base learners are decision trees. Unlike traditional decision trees grown independently, the trees in XGBoost are built sequentially, with each tree learning from the mistakes of the previous ones. This approach captures complex patterns and interactions in the data more effectively (Malik, 2020).

The objective function in XGBoost consists of two components: a loss function that measures how well the model fits the training data, and a regularization term that penalizes model complexity. The loss function typically used is mean squared error (MSE) for regression tasks and logistic loss for classification tasks. The regularization term includes L1 (Lasso) and L2 (Ridge) penalties, which help prevent overfitting by constraining the model's complexity.

XGBoost employs gradient descent to minimize the objective function. In each iteration, the algorithm computes the gradients of the loss function with respect to the predictions of the

current ensemble. These gradients guide the adjustments needed to reduce errors. A new tree is then trained to predict these gradients, with its weights adjusted to minimize the loss function, and it is added to the ensemble (Datamapu, 2023).

To prevent overfitting, XGBoost incorporates a technique called shrinkage (or learning rate), where the contribution of each new tree is scaled down by a factor. This allows for smaller adjustments at each iteration, slowing down the learning process and ensuring more gradual convergence to the optimal solution.

To further enhance generalization and reduce overfitting, XGBoost supports feature subsampling (or column sampling), where a random subset of features is used to build each tree. This technique, inspired by random forests, ensures the model does not rely too heavily on any single feature, thus improving robustness (Ellis, 2022).

XGBoost has a built-in mechanism for handling missing values. During the training process, the algorithm learns the best direction to take when it encounters a missing value in a feature, allowing it to manage datasets with incomplete information effectively without extensive preprocessing (Malik, 2020).

Implementing XGBoost in embedded systems involves several considerations to ensure optimal performance and resource utilization. Techniques such as model compression—including pruning, quantization, and distillation—are used to reduce the model size, making it suitable for environments with limited storage and memory. Leveraging specialized hardware like GPUs or TPUs can further enhance XGBoost's performance by efficiently handling its parallel nature. XGBoost models can be integrated into edge devices using frameworks like TensorFlow Lite or ONNX Runtime, enabling machine learning capabilities directly on the device without constant cloud communication. This integration is crucial for real-time applications requiring quick model updates and fast inference times (What Is XGBoost?, 2024).

In conclusion, XGBoost is a powerful machine learning algorithm with advanced features that make it highly effective for embedded systems. Its ability to handle large datasets, manage missing values, and prevent overfitting through regularization techniques ensures high accuracy and efficiency. The support for parallel processing and real-time model updates makes XGBoost a suitable choice for performance-critical applications in modern embedded

systems. By leveraging these strengths, developers can build more intelligent, responsive, and efficient embedded systems capable of handling complex tasks and providing valuable insights from data.

Multilayer Perceptron (MLP)

A Multilayer Perceptron (MLP) is a class of feedforward artificial neural network (ANN) that consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each node, or neuron, in one layer connects to every node in the following layer, making MLP a fully connected network. The learning process in MLP is supervised and involves adjusting the weights of the connections to minimize the error in predictions through a process called backpropagation (Kostadinov, 2021). Figure 11 depicts the structure of an MLP and the flow of information through the network.

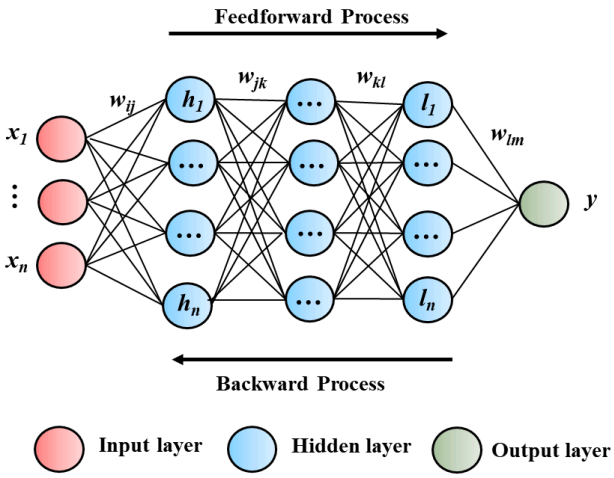

Figure 11: Structure of a Multilayer Perceptron (Kostadinov, 2021).

MLPs are capable of capturing complex patterns and relationships within data due to their multiple layers and non-linear activation functions. This makes them suitable for a variety of tasks, including classification, regression, and even more complex tasks like image recognition and natural language processing.

In the context of embedded systems, MLPs offer several benefits. Their ability to learn non-linear relationships makes them powerful tools for tasks that require more than simple linear separation, such as sensor data analysis or control systems. Moreover, with advancements in hardware accelerators and efficient training algorithms, deploying MLPs in real-time applications has become more practical (Ajani et al., 2021).

However, MLPs also present challenges. They require substantial computational resources for training due to the backpropagation process and the need to adjust many parameters. This can be mitigated by using pre-trained models or leveraging transfer learning. MLPs also tend to be less interpretable than simpler models like decision trees, as understanding the contributions of individual neurons can be complex. Despite these challenges, the flexibility and power of MLPs make them a valuable tool in the machine learning toolkit for embedded systems and beyond (Ajani et al., 2021).

3. Problem Statement and Solution

3.1 Problem Statement

Modern autonomous robot navigation systems encounter several critical challenges, including high costs, substantial computational requirements, and sensitivity to environmental conditions. Specifically, systems employing SLAM incur high financial costs and demand significant computational power. Additionally, SLAM systems that use cameras are prone to failure in low-light conditions, limiting their effectiveness in various environments. Traditional line-following robots also present issues, such as complex PID tuning. Moreover, these robots rely on pre-installed lines, making them vulnerable to environmental changes, which compromises their reliability.

These combined challenges impede the broad adoption of autonomous robot navigation systems across diverse applications. There is, therefore, a pressing need for an innovative solution that is cost-effective, resource-efficient, and resilient to environmental conditions.

3.2 Proposed Solution

The proposed solutions aim to address the outlined challenges by developing a robust, cost-effective navigation system that operates independently of environmental conditions and does not rely on pre-installed lines.

This solution will utilize an Arduino microcontroller to ensure affordability and accessibility, facilitating widespread adoption. By integrating a TinyML framework-supported machine learning algorithm, the system will enable efficient processing of sensor data and intelligent decision-making in real-time. Incorporating LiDAR technology provides PathPilot with a

robust perception system that delivers accurate and reliable data, unaffected by varying environmental conditions.

Additionally, a user-friendly Android app interface will be developed to simplify the training process of the autonomous navigation system. By leveraging advancements in hardware and machine learning, these solutions aim to overcome the limitations of current navigation systems, offering a cost-effective, high-performance alternative that is resilient to environmental changes. This innovative approach is poised to revolutionize autonomous robot navigation across various applications.

4. Conceptual Design and Implementation Details

4.1 Project Phases

The project is structured into two integral phases to ensure a systematic approach to developing and validating the robot's capabilities: the Training Phase and the Testing Phase.

During the Training Phase, a user assumes control of the robot through Bluetooth and an Android application, guiding it through a designated course to collect crucial training data. This data, comprising measurement points from the robot's LiDAR sensor, is labelled with corresponding control command inputs of the robot’s movement from the user. The collected dataset is then stored on an onboard SD card.

In the Testing Phase, the dataset is exported to a computer where a classification machine learning model, implemented using Python libraries such as Scikit-Learn, is trained. The resulting trained model is then exported back to the Arduino for seamless execution. The supervised learning process and the expected output of the trained model for the PathPilot robot are illustrated in Figure 12.

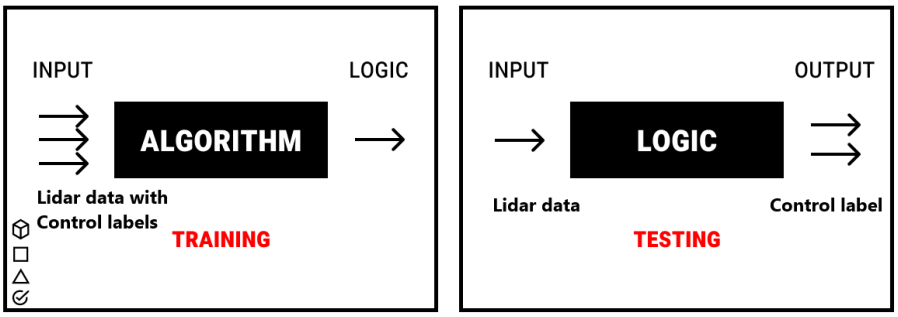

Figure 12: Supervised learning of the machine learning model for PathPilot.

The trained machine learning model undergoes evaluation by directing the robot based on predicted control command labels. Rigorous testing follows, incorporating various obstacles and changing environmental conditions, such as different lighting scenarios. This phase serves as a comprehensive assessment of the system's reliability under diverse circumstances, providing insights into its adaptability and performance in real-world scenarios.

4.2 Robot Hardware Development

The development of the robot hardware involved designing and assembling the robotic platform to complement the software functionalities established during the Training Phase. This encompassed electrical design and the integration of various hardware components to create a durable and agile robotic system capable of autonomous traversal. Appendix Figure 13 shows the assembled robot, while Appendix Figure 14 depicts the hardware schematic of the robot.

Safety Switch

The PathPilot robot is equipped with a strategically positioned safety switch on its top deck. This switch is directly connected between the Lithium Polymer (LiPo) battery and the circuitry of the robot. In the event of a dangerous malfunction, the safety switch allows the user to power off the robot by breaking the connection between the battery and circuitry directly. This ensures that no components of the circuit receive power once the switch is activated. Appendix Figure 15 illustrates the location of the safety switch on the PathPilot robot.

Power source

During the early stages of the PathPilot project, a 12.6V 3-cell LiPo battery was chosen as the

primary power source for the robot. However, subsequent testing revealed critical issues: the LiDAR sensor failed to receive power from the Arduino Mega microcontroller, and the microcontroller itself overheated. Investigation revealed that, according to its datasheet, the maximum allowable power supply voltage for the microcontroller is 12V. Despite the initial power supply voltage being only 5% higher than this limit, it caused the onboard voltage regulator to overheat and malfunction, preventing the LiDAR sensor from functioning.

To resolve this, the power supply was changed to a 7.4V 2-cell LiPo battery, ensuring it remained within the microcontroller's allowable voltage limit without affecting the robot's performance. This adjustment successfully prevented overheating of the microcontroller and enabled the proper functioning of the LiDAR sensor.

SD Card, Bluetooth module and Ultrasonic sensor connections

The SD card module, ultrasonic sensor, and Bluetooth module are all powered by the Arduino's 5V output pin. The SD card module uses the SPI pins for data transfer, the ultrasonic sensor uses digital IO pins for its Trigger and Echo functions, and the Bluetooth module interfaces via the UART (TX1 & RX1) pins.

A voltage divider circuit is used to adapt the 5V digital signal from the Arduino's UART TX1 pin to the Bluetooth module's 3.3V RX pin. This voltage divider, designed using 1kΩ and 2kΩ resistors, ensures the signal is correctly adjusted from 5V to 3.3V, as shown in the following calculation:

$$Output\ of\ voltage\ divider\ circuit\ = \frac{2k\Omega}{2k\Omega+1k\Omega} \times 5V\ = 3.33V$$

L298N motor driver

The L298N motor driver is powered directly by the 7.4V LiPo battery. Two of its pulse width modulation (PWM) pins, ENA and ENB, connect to the Arduino, allowing it to regulate the speed of the left and right motors respectively. To control the direction of motor spin, four digital output signal pins from the Arduino interface with the IN1 through IN4 pins of the motor driver. The truth table for these digital signals, which regulate the motors' spin direction, is displayed in Appendix Figure 16.

LiDAR sensor connections

The LiDAR sensor is powered by the Arduino's 5V output pin, while a 5V voltage regulator powers the sensor's motor through its VMOTO pin. The motor of the sensor is powered separately from the main power input of the sensor to prevent motor signal noise from interfering with the sensor's operation.

The LiDAR sensor interfaces via its TX and RX pins to the Arduino's UART (TX2 & RX2) pins for data transfer. Additionally, the sensor's PWM pin interfaces with the Arduino digital IO pin to control the sensor's motor speed.

The hardware development of the robot involved precise assembly of physical wire connections, meticulously soldered to ensure flawless operation and prevent any issues stemming from faulty connections. The soldering and craftsmanship of the wires are depicted in Figure 17.

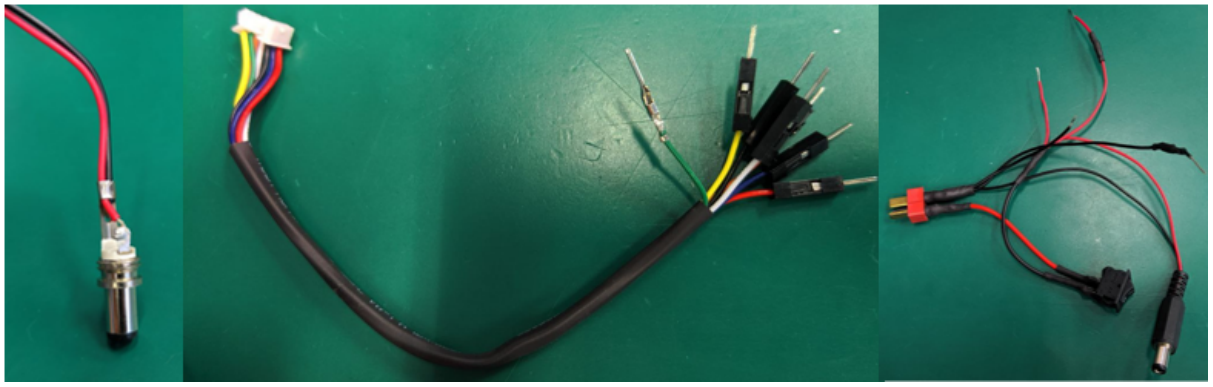

Figure 17: Assembled wires with excellent soldering.

4.3 Android Application Development

The Android application developed for the PathPilot project facilitates communication with the robot via Bluetooth for precise control during data collection on the course circuit. The application features intuitive buttons with haptic vibration feedback for the user to click to control the movement of the robot. The application transmits control commands to the robot in the form of single characters whenever the user clicks the respective button in the application.

Initially designed for sending only motor movement commands during the early stages of the project, enhancements were made to incorporate a feature enabling users to start and stop

data recording of the LiDAR sensor via an intuitive switch in the application. This addition grants flexibility, allowing users to activate recording exclusively during the robot's traversal of the course track, preventing unnecessary data accumulation during miscellaneous testing or outside the designated circuit, thus safeguarding data integrity and optimizing memory usage. This feature additionally helps reduce the robot's power consumption when data recording ceases, thereby conserving the battery life of the robot.

The development of the Android application was facilitated using the MIT App Inventor platform. Appendix Figure 18 illustrates the programming of the application, while Figure 19 showcases its user-friendly interface design.

Figure 19: User interface of the Android application.

4.4 Arduino Software Development (Training Phase)

During the Training Phase of the project, the Arduino software plays a pivotal role in orchestrating various tasks, including communication with the Android application via Bluetooth, motor control for the robot, capturing measurement data from the LiDAR sensor and managing the storage of this acquired data on an SD card. The software seamlessly interfaces with essential components such as the LiDAR sensor, SD card module, motor driver, and Bluetooth module, ensuring smooth integration and cohesive functionality. The design flowchart of the software and its Interrupt Service Routine (ISR) is shown in Appendix Figure 20.

The software interfaces with the LiDAR sensor using the "RPLiDAR " library to acquire distance measurement data. During each program iteration, if the user has enabled the data recording feature through the Android application, the software reads the distance measurement, angle value, and quality value of the measurement from the LiDAR sensor. If the quality value of the measurement is acceptable (greater than zero), the distance measurement is stored in a buffer at an index corresponding to its angle value.

After reading and processing measurement data from the LiDAR sensor into the buffer, the software checks for control commands received via the Bluetooth module from the Android application. If a control command is received, the software executes the relevant action, such as motor control or enabling/disabling the data recording function.

A 182ms timer interrupt is used to periodically construct data strings by annotating the 360 data measurements in the buffer with the corresponding control label received from the user, as shown in Figure 21, and storing it to the SD card. These data strings will be used as the dataset to train the machine learning model later in the project. Initially, the software design aimed to write distance measurement data to the SD card immediately upon measurement. However, this approach caused system 'hang' issues due to continuous SD card writing. To address this, the periodic storage mechanism was introduced, ensuring robust system operation by saving measurement data every 182 milliseconds.

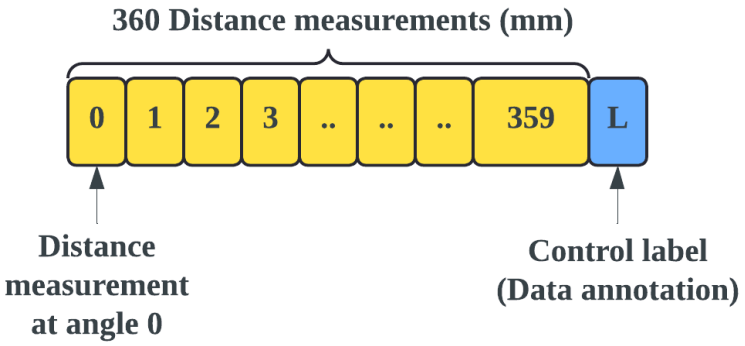

Figure 21: Format of a string of training data.

The 182-millisecond interval was selected because the LiDAR sensor has a scanning rate of 5.5 Hz, which corresponds to a 182-millisecond period. By choosing a 182-millisecond interval, each angle's distance measurement is updated at least once before being written to the SD card. This approach ensures the PathPilot's rate of the data capture is synchronized with the scanning rate of the LiDAR.

To implement this enhancement, a timer interrupt was employed using the 'Timer1' registers of the Arduino microcontroller. The configuration involved precise calculations to establish the required interval and appropriate timer register values. By dividing the Arduino clock frequency and setting the Timer1 counter value, accurate timing control was achieved. The calculations are as follows:

$$Required\ interval\text{: } 5.5Hz$$

$$Prescale\ Arduino\ clock\ frequency\text{: } 16MHz \div 64 = 250000Hz$$

$$Timer1\ register\ counter\ value\text{: } \frac{250000Hz}{5.5Hz} - 1 = 45453$$

Figure 22 illustrates the detailed configuration of registers for the Timer1 interrupt, encapsulated within a dedicated function to promote code modularity and readability.

```
void setupTimer1(void)
{
  // Configure Timer1 for 182ms
  cli(); // Disable all interrupts for register configuration
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0;
  // 5.5 Hz (16000000/((45453+1)*64))
  OCR1A = 45453;
  // CTC
  TCCR1B |= (1 << WGM12);
  // Prescaler 64
  TCCR1B |= (1 << CS11) | (1 << CS10);
  // Output Compare Match A Interrupt Enable
  TIMSK1 |= (1 << OCIE1A);
  sei(); // Enable global interrupts
}
```

Figure 22: Configuration of registers for Timer1 interrupt in the Arduino microcontroller.

The software also incorporates an error-handling mechanism in the event of LiDAR sensor failure, where it initiates corrective actions to recalibrate the sensor and resume scanning operations seamlessly.

Throughout the software, emphasis is placed on encapsulating functionalities into cohesive functions, promoting code modularity, and enhancing readability. This modular approach facilitates easier debugging of the software.

4.5 LidarPlayback Program

The LidarPlayback Program is an essential diagnostic tool developed with Python during the PathPilot robot's development. Its primary function is to display the dataset collected by the robot and highlight its selected features, translating distance measurements and their corresponding angle indices into an intuitive visual representation of the LIDAR sensor data.

This program significantly benefits the robot's development by providing visualization of the data collected by the LIDAR sensor, making it easy to interpret and analyse the robot's perception of its environment and the quality of the dataset. This visualization helps identify discrepancies or errors in the sensor data, aiding in diagnosing issues related to the LIDAR sensor and the data collection process. By displaying real-time data, it enables problems with the sensor and data to be pinpointed and addressed more efficiently. Furthermore, the program provides a platform to test and validate the sensor data, ensuring that the robot's perception system is accurate and reliable, which is crucial for the successful navigation and operation of the robot. Appendix Figure 23 shows a screenshot of the display in the program.

The LidarPlayback Program operates by reading the data from a text file stored on the SD card, where the LiDAR sensor measurements are recorded. The program uses the Pygame library to create a visual representation of the data on a graphical interface. It starts by initializing Pygame and setting up constants such as the LIDAR resolution, maximum distance, screen size, and colours used for the display. The program then reads the LIDAR sensor data from the specified file path, which consists of distance measurements recorded at different angle indices.

To calculate the positions of points based on the LIDAR resolution, the program determines the angle for each point based on its index in the data string and uses trigonometric functions to calculate the x and y coordinates. The distances are scaled to fit the screen size, and for each distance measurement, the program calculates the position on the screen and draws a circle dot at that position. The distance measurements that are in the selected features of the dataset are highlighted as red dots in the display.

Additionally, a circle representing the robot is drawn at the centre of the screen, helping to understand the robot's perspective and the relative positions of the captured data

measurements around it. The program continuously updates the display with new data, providing a real-time view of the LIDAR sensor data from the dataset.

4.6 Arduino Software Development (Testing Phase)

During the Testing Phase of the project, the Arduino software plays a crucial role in enabling the robot to navigate autonomously. The software is responsible for interacting with various components such as the LiDAR sensor, ultrasonic sensor, motor driver, buzzer, and the machine learning classifier. The key tasks performed by the software in this phase include obstacle detection, motor control, handling LiDAR data and getting the control label prediction from the machine learning classifier. The design flowchart of the software and its ISR is shown in Appendix Figure 24.

Similar to the Arduino software for the Training Phase of the project, the Testing Phase software interfaces with the LiDAR sensor using the "RPLiDAR " library to acquire distance measurements. During each program iteration, the software reads the distance measurement, angle value, and quality value from the LiDAR sensor, and the distance measurement reading from the ultrasonic sensor. The software then checks if there is an obstacle based on the distance measurements from the LiDAR and ultrasonic sensor, and the angle value from the LiDAR sensor.

An obstacle is identified when the measurements from either the LiDAR or ultrasonic sensor indicate that an object is within the robot’s obstacle detection zone, which is illustrated in Figure 25. The inclusion of an ultrasonic sensor, in addition to the LiDAR sensor, addresses the challenge of detecting low-height obstacles that are physically below the LiDAR sensor’s line of sight. This additional sensor also provides a layer of redundancy, enhancing overall safety.

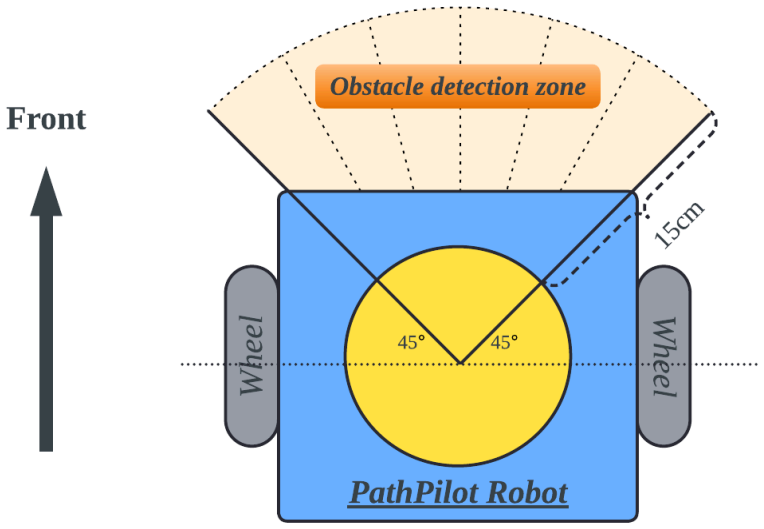

Figure 25: Obstacle zone of the PathPilot robot.

When an obstacle is detected, the software stops the robot's motors and activates the buzzer to provide auditory feedback to indicate the presence of the obstacle. The buzzer remains active until the obstacle is cleared, ensuring the user is continuously informed of potential hazards.

If there are no obstacles detected, the software ensures that the buzzer is turned off, and the distance measurement from the LiDAR sensor is stored into a data buffer at the index that corresponds to the angle value of the distance measurement. This data buffer stores up to 360 distance measurements.

Similar to the Arduino software for the training phase, a timer interrupt is employed to periodically process data from the LiDAR sensor. This interrupt triggers every 182 milliseconds. If no obstacles are detected, the distance measurements at the selected feature indexes of the data buffer are extracted and compiled into a data string containing 210 distance measurements, which is the number of selected features. This data string is then fed into a pre-trained classifier model to predict the control label based on the sensor input. This approach ensures that only the most informative distance measurements, determined by their angles (selected features), are considered for predicting the control label.

Finally, based on the classifier's prediction, the software sends the corresponding signals to the motor driver to control the robot's movement.

4.7 Software for Training of Machine Learning Model

The software development for training the machine learning model entails creating a Python program specifically designed for this purpose. After evaluating four different models, the one with the highest accuracy—XGBoost—was chosen for deployment in PathPilot. The process of testing and selecting the best model will be detailed in the "Testing, Evaluation, and Outlook" section of the report. The Python program follows a structured process to handle the dataset and train the model.

The training process begins with data cleaning, where the dataset, collected during the Training Phase of the PathPilot project, is prepared. This involves removing data strings annotated with unwanted control labels, such as those for backward movement or enabling data recording, leaving only data strings annotated with control labels for forward, forward-left, and forward-right movements of the robot.

After cleaning the data, the distance measurements and their corresponding control labels are set as input and output respectively. The control labels are then encoded from characters to integers to be compatible with the training model. The dataset is split into training and test sets with an 80/20 ratio, as studies suggest this ratio yields optimal training results (Gholamy et al., 2018).

Next, feature selection is performed. Using the SelectKBest method with the f_classif score function from the sklearn library, which is the ANOVA F-value between the label and the feature, to select the most relevant features for classification and the program then displays the list of selected feature indexes to be used in the Arduino code deployment. The training dataset is then transformed to retain only these selected features.

For the PathPilot project, the features of the dataset are the indexes of each data string, which are the angles of the distance measurements. Selecting the optimal number of features (K) is critical for model performance, ensuring that only the most informative LIDAR measurement angles are considered for the prediction of the control label. An optimal K value of 210 was implemented in the training of the model, where this value was determined through testing based on the accuracy performance across different K values. This testing for the optimal K value is also covered in the "Testing, Evaluation, and Outlook" section of the report.

With the optimal features selected, the model training process begins. Using the Fit function of the sklearn library, the model is trained on the selected features from the training dataset and its performance is evaluated on the test data. The XGBoost model is configured with a maximum depth of 3 for each tree, which results in the maximum allowable size for deployment on the Arduino microcontroller. Reducing the depth of the decision trees in the XGBoost model had little effect on the accuracy of the model (less than 1% drop in accuracy).

After training, the model is exported for deployment to the Arduino microcontroller. The micromlgen library converts the trained model to a C code header file, which is then integrated into Arduino software for real-time classification. The resulting header file, XGBoost.h, contains the necessary code for executing the classification on the Arduino platform.

The flowchart for this Python program is illustrated in Appendix Figure 26.

5. Testing, Evaluation, and Outlook

5.1 Testing and Evaluation Process of Machine Learning Models

Four machine learning models were tested and evaluated for implementation in the PathPilot robot: DecisionTree, RandomForest, MLP, and XGBoost. These models, all supported by the sklearn software library and suitable for supervised learning, were trained on the same dataset collected from the project's Training Phase. The evaluation process, depicted in Appendix Figure 27, followed a consistent structure for the testing of all models.

The process is similar to the Python program used to train the machine learning model for the PathPilot, with the exception of the feature selection stage. The comparison of the accuracy results for the four models is illustrated in Figure 28. Based on these results, XGBoost, which achieved the highest accuracy score of 81.1%, was selected as the classification model for the PathPilot robot.

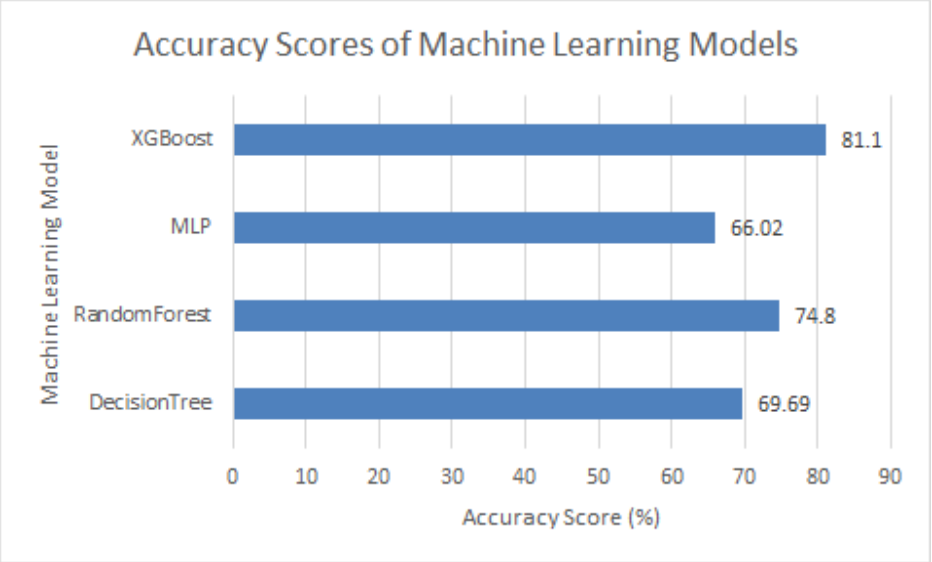

Figure 28: Result of accuracy scores of the tested machine learning models.

After selecting the XGBoost model, efforts were made to further enhance its performance by determining the optimal K value for feature selection. The program developed for this purpose is outlined in Appendix Figure 29.

The process, similar to the Python program used to train the machine learning model, begins with data cleaning and splitting the data into input/output, and the training/test sets with an 80/20 ratio.

The program iteratively evaluates K values, incrementing in steps of 10 from 10 to 360, where 360 is the dimension of the data string in the dataset. For each K value, the following steps are performed:

Feature Selection: Using the SelectKBest method with the f_classif score function from the sklearn library, the program determines the top k highest scoring features. The training set is then transformed to retain only these selected features.

Model Training: The XGBoost model is trained using the selected features from the training set, with performance evaluated on the test data.

Model Accuracy Testing: The model's accuracy score is obtained using the test set. The accuracy score and the corresponding k value are stored for further analysis.

After evaluating all K values up to 360, the program exits the loop and plots a graph of K values against their corresponding accuracy scores, shown in Figure 30. This graph helps

visualize the performance of different K values, facilitating the selection of the optimal K value.

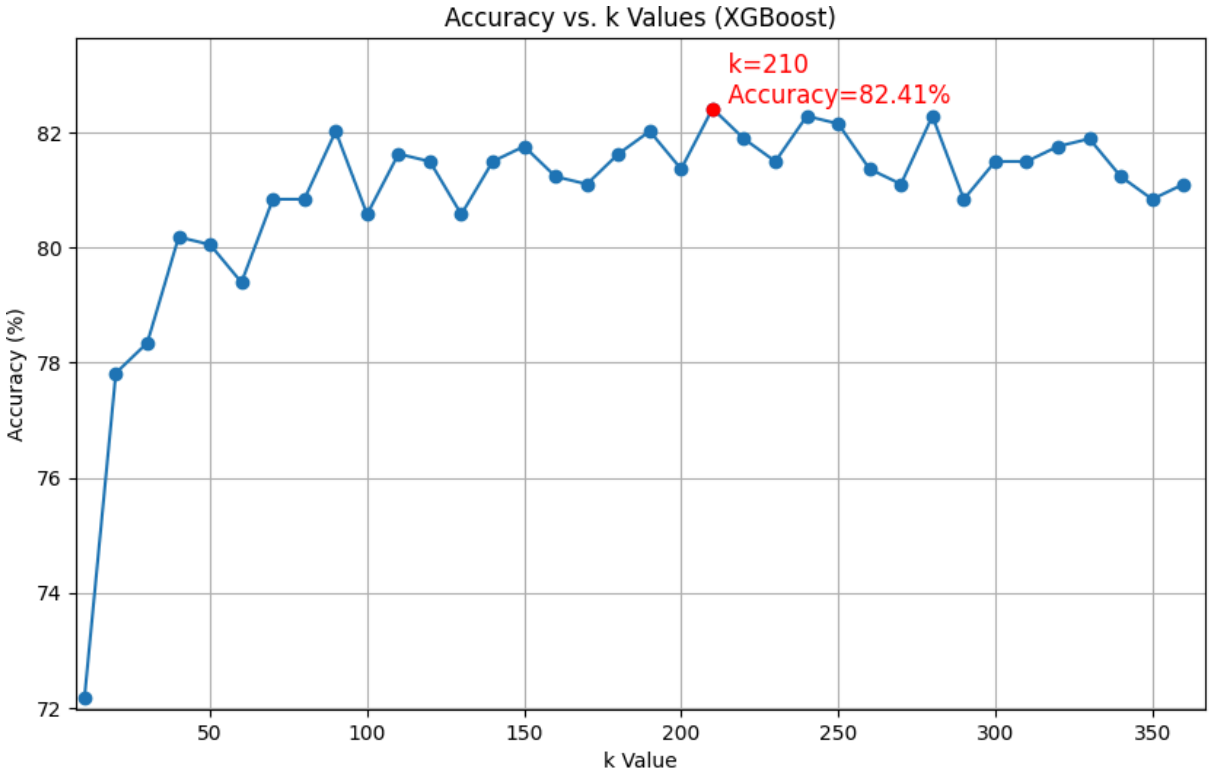

Figure 30: Result of K values and their corresponding accuracy of the XGBoost model.

The results in Figure 30 indicate that the optimal K value is 210, which improved the model's accuracy from 81.1% (shown in Figure 28) to 82.41%, representing a 1.31% improvement. This optimal K value of 210 was subsequently used in the model training for the PathPilot robot. This systematic approach ensures the selection of the best possible number of features, maximizing the model's accuracy and efficiency for deployment on the Arduino microcontroller.

5.2 Prototype Testing and Functional Demonstration

Overview of PathPilot Functionality Testing

The functionality of the PathPilot was tested using three distinct course layouts: a circle, a square, and a B-shaped course track. The colour white was selected for the circuit course walls, grounded in the scientific understanding that white reflects all wavelengths of the light spectrum most effectively (Vanik, 2023). This choice optimizes the LiDAR sensor performance and ensures accurate data acquisition during the tests.

Appendix Figure 31 illustrates the B-shaped course track layout used in testing PathPilot.

Data Collection and Model Training

For each course, PathPilot collected 15 MB of training data, representing 10 minutes of driving the robot. The datasets from all three course layouts were combined into a single dataset, which was then used to train the XGBoost model. Feature selection techniques were implemented, resulting in an accuracy of 82.41%. The trained model was subsequently deployed into PathPilot for testing on each course layout.

Testing and Results

Testing revealed that PathPilot successfully traversed all course layouts, demonstrating its versatility and ability to adapt using knowledge from the combined dataset. The results of these tests are displayed in Figure 32.

Course layout	Able to transverse for 30mins without crashing?
Circle	Yes
Square	Yes
"B" shaped	Yes

Figure 32: PathPilot functionality and versatility testing across different course layouts

Robustness Testing at Different Motor Speeds

To evaluate the robustness of PathPilot's navigation capabilities, the system was tested at varying motor speeds. The PWM value for motor speed was incrementally increased in steps of 50, ranging from 50 to 255. Figure 33 illustrates the results of this testing.

Motor speed (PWM)	Able to transverse for 30mins without crashing?
50	Yes
100	Yes
150	Yes
200	No
255	No

Figure 33: Robustness of PathPilot at different motor speeds

The results in Figure 32 indicate that PathPilot demonstrates a certain level of robustness across various motor speeds. However, it was observed that when the PWM value reaches 200 or higher, PathPilot exhibits a delayed reaction to turns in the course layout. This delay was likely due to the limitations of the LiDAR scanner's scanning rate. At higher motor speeds, the scan rate of 5.5 Hz was insufficient for PathPilot to update its understanding of environmental changes quickly enough to anticipate upcoming turns.

Reliability Testing in Varying Environmental Conditions

The reliability of PathPilot was also assessed in different environmental conditions. Tests were conducted in a controlled room with lighting conditions set at 16 LUX to simulate a dark environment and 150 LUX for a bright environment. The testing procedures mirrored those shown in Figures 32 and 33, and the results remained consistent across lighting conditions, confirming that PathPilot is reliable under different environmental settings.

Obstacle Detection Testing

The final test evaluated PathPilot's obstacle detection capabilities under the same lighting conditions of 16 LUX and 150 LUX. In both scenarios, PathPilot effectively detected obstacles swiftly and safely, demonstrating its reliable performance in diverse lighting environments.

5.3 Project Evaluation and Outlook

Evaluation Overview

The PathPilot project was evaluated based on its success in meeting the defined objectives and addressing the key problems identified in the introduction. The evaluation criteria were centred on affordability, adaptability, robustness, and reliability, ensuring that the solution meets the project’s goals of providing a cost-effective, efficient, and reliable navigation system.

Affordability

One of the primary objectives of PathPilot was to offer a more affordable alternative to existing autonomous navigation systems. A cost comparison was conducted, as shown in

Figure 34, between PathPilot and a benchmark autonomous robot, the TurtleBot, which is equipped with a LiDAR sensor and camera system for SLAM-based navigation.

Main System Component	TurtleBot 3	Cost (TurtleBot 3)	PathPilot	Cost (PathPilot)
Microcontroller/Processor	Raspberry Pi 4	$99.90	Arduino Mega	$23.40
Sensors	LiDAR	$143.40	LiDAR	$143.40
	Raspberry Pi camera	$54.60	NA	NA
Total Cost		**$297.90**		**$166.80**

Figure 34: Comparison in cost of implementation

The evaluation showed that PathPilot is at least 44% more affordable than the TurtleBot for autonomous navigation applications. This significant cost advantage stems from the efficient use of an Arduino microcontroller and the integration of TinyML frameworks, which reduce both hardware and computational costs. By achieving this affordability, PathPilot successfully fulfils its objective of being accessible to a broader spectrum of users and applications, making autonomous navigation more financially viable across various industries.

Adaptability

The PathPilot project aimed to address the limitations of existing systems by creating a navigation system capable of adapting to different course layouts without reliance on external environmental cues or pre-installed infrastructure. The successful traversal of various course layouts with PathPilot underscores its ability to function effectively across diverse scenarios. This adaptability is crucial for ensuring the system's versatility and operational independence, aligning with the project's objective of overcoming the constraints of traditional navigation methods.

Robustness

Robustness was a key objective, particularly regarding the system's performance under different motor speeds and environmental conditions. The evaluation confirmed that PathPilot maintains operational efficiency across a range of motor speeds and environmental settings. Despite minor delays at higher PWM values due to LiDAR scanning limitations, the system's overall performance remains consistent. This indicates that PathPilot is robust

enough to handle variations in speed and environmental conditions, fulfilling the project's goal of creating a resilient navigation system.

Reliability

The reliability of PathPilot was assessed through its performance in varying lighting conditions and its ability to detect obstacles. The consistent performance across different lighting environments, coupled with effective obstacle detection, demonstrates the system's reliability. PathPilot's ability to function accurately in both low and high light conditions without significant performance degradation meets the objective of ensuring dependable operation in diverse real-world scenarios.

Outlook and Future Applications

The successful evaluation of PathPilot not only highlights its effectiveness but also opens several avenues for future development and application. The system's affordability and robustness make it an attractive option for industries seeking cost-effective autonomous solutions. Potential applications include warehouse automation, agricultural robots, security patrol robots, cleaning robots, hospital delivery robots, and smart manufacturing systems, where efficient and adaptable navigation is essential.

Looking ahead, future iterations of PathPilot could benefit from advancements in sensor technology and machine learning algorithms. Enhancing LiDAR technology or integrating additional sensors could address current limitations and further improve the system's perception capabilities and navigation accuracy. These improvements would significantly boost the overall performance of PathPilot.

Additionally, PathPilot's design and cost-effectiveness position it well for scalability. The system could be adapted for various sizes and types of robots, potentially extending its utility to new domains such as personal robotics or autonomous vehicles. This scalability enhances the system's applicability across different sectors and contributes to its broader adoption.

6. Application of Classroom Knowledge and Acquisition of New Skills

6.1 Application of Classroom Knowledge

The PathPilot project allowed me to integrate and apply various concepts from my engineering coursework in a practical and interdisciplinary context. My academic background provided a solid foundation for the key components of the project, especially in microcontroller programming, embedded systems, and circuit design.

Microcontroller Programming and Embedded Systems: The course TLM2002-Embedded System Design was particularly relevant when developing the Arduino software for both the Training and Testing Phases of the project. This course covered microcontroller architecture, programming, and interfacing with peripheral devices, all of which were critical when integrating the LiDAR sensor, SD card, and Bluetooth module. The use of timers, interrupts, and UART communication in the Arduino software directly applied these theoretical principles. For instance, the Timer1 interrupt, which managed data recording intervals, addressed practical issues such as system hangs due to continuous SD card writing, demonstrating a seamless transition from classroom theory to real-world application.

Sensors and Control: My coursework in TLM2001-Sensors and Control provided essential knowledge for integrating and managing various sensors within the robot. The implementation of the LiDAR sensor and ultrasonic sensor for obstacle detection leveraged principles from this course. Understanding sensor data acquisition and processing was vital in ensuring the robot could navigate and respond to its environment accurately.

Electronics and Circuit Design: Courses TLM1007-Digital Systems and TLM3009-Automotive Electronics were critical when designing the robot's power system and addressing electrical challenges. This practical application of classroom knowledge in voltage regulation, power management, circuit design and protection were crucial for the project's success.

Systems and Software Engineering: The systematic approach to software development and project management, taught in TLM3004-Systems and Software Engineering, was integral to organizing and structuring the project phases. From planning the data collection process during the Training Phase to implementing and testing the machine learning model in the

Testing Phase, the principles of software engineering ensured a methodical and efficient workflow.

6.2 Acquisition of New Knowledge

While classroom knowledge provided a foundation, significant new learning was required to bring PathPilot to fruition. I self-taught several key areas not covered by my course, such as machine learning, data science, and Python programming.

Machine Learning and Data Science: The project demanded a robust control system, which I developed by independently studying machine learning and data science. I focused on supervised learning algorithms and their application in robotic control, which was crucial during the Testing Phase. Training a classification model using Python libraries like Scikit-Learn and XGBoost involved data cleaning, feature selection, and model evaluation—skills essential for the project but not part of my formal coursework.

Python Programming: Despite Python not being part of my curriculum, I learned the language to handle data processing, model training, and developing the LidarPlayback Program.

Android Application Development: To facilitate user interaction with the robot, I also learned to develop the Android application used in the Training Phase of the project by using the MIT App Inventor platform.

6.3 Logical Explanation and Rationale

The integration of classroom knowledge with newly acquired skills was driven by the logical need to address specific challenges encountered during the project. For instance, my decision to use a 182ms timer interrupt for data recording was based on the LiDAR sensor's scanning rate. Switching to a 7.4V battery was a calculated move to resolve power issues, informed by the microcontroller's voltage limits and the sensor's operational requirements. The choice of the XGBoost model for machine learning was justified by its accuracy and compatibility with the Arduino platform, ensuring efficient real-time classification.

7. Conclusion

The PathPilot project successfully developed a cost-effective, efficient, and adaptable autonomous navigation system using resource-constrained embedded systems and machine learning algorithms. The project achieved its primary objectives, demonstrating key advancements in affordability, adaptability, robustness, and reliability, which collectively redefine the landscape of autonomous robot navigation.

PathPilot proved to be a financially accessible solution, being at least 44% more affordable than existing alternatives such as the TurtleBot. This was achieved by leveraging Arduino microcontrollers and TinyML frameworks, which significantly reduced both hardware and computational costs while maintaining high performance. As a result, the system broadens the scope of autonomous navigation applications, making it accessible to a wider range of industries and users.

In terms of adaptability, PathPilot excelled by successfully navigating various course layouts without relying on external cues or pre-installed infrastructure. This demonstrated the system's ability to operate across diverse environments, meeting the project's objective of providing flexible and independent navigation solutions. The project's robustness was evident in its ability to maintain operational efficiency across different motor speeds and environmental conditions. Despite minor delays observed at higher speeds due to LiDAR scanning limitations, PathPilot consistently demonstrated resilience, effectively handling variations and ensuring dependable performance.

The project's reliability was further underscored by its successful navigation in varied lighting conditions and effective obstacle detection. PathPilot operated efficiently in both low and high-light environments, showcasing its capability to function accurately without significant performance degradation, even in challenging real-world scenarios.

Looking ahead, the PathPilot project paves the way for several exciting avenues of future exploration and development. Integrating advanced sensors or improved LiDAR technology could enhance the system's perception capabilities and address current limitations related to scan rates at higher speeds. Exploring more sophisticated machine learning algorithms could further optimize PathPilot's performance by improving decision-making and navigation accuracy. Additionally, a promising area of future work involves leveraging edge computing

technology to utilize edge servers for processing sensor data from multiple PathPilot robots. This approach would enable centralized processing of data, running classification model predictions, and sending the predicted control labels back to each robot, enhancing real-time decision-making and coordination across multiple autonomous systems.

PathPilot's scalable design positions it well for adaptation to various robot sizes and types, potentially expanding its applications to personal robotics and autonomous vehicles. This scalability enhances the system's applicability across different sectors, contributing to its broader adoption. Furthermore, the system's cost-effectiveness and versatility make it suitable for numerous industry-specific applications, such as warehouse automation, agricultural robots, security patrol robots, cleaning robots, hospital delivery robots, and smart manufacturing systems.

In summary, PathPilot's innovative approach addresses key challenges in autonomous robot navigation, offering a reliable and adaptable solution poised for significant impact across diverse industries. Continued development and refinement of the system will enhance its capabilities, ensuring PathPilot remains at the forefront of autonomous navigation technology.

Appendices

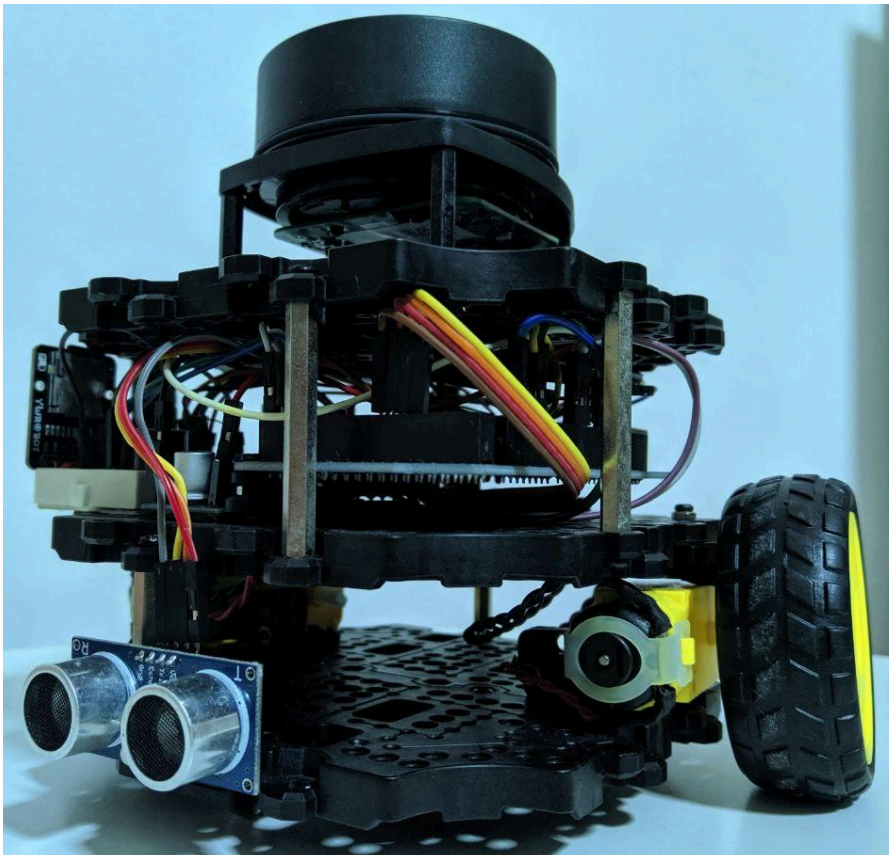

Figure 13: Assembled PathPilot robot.

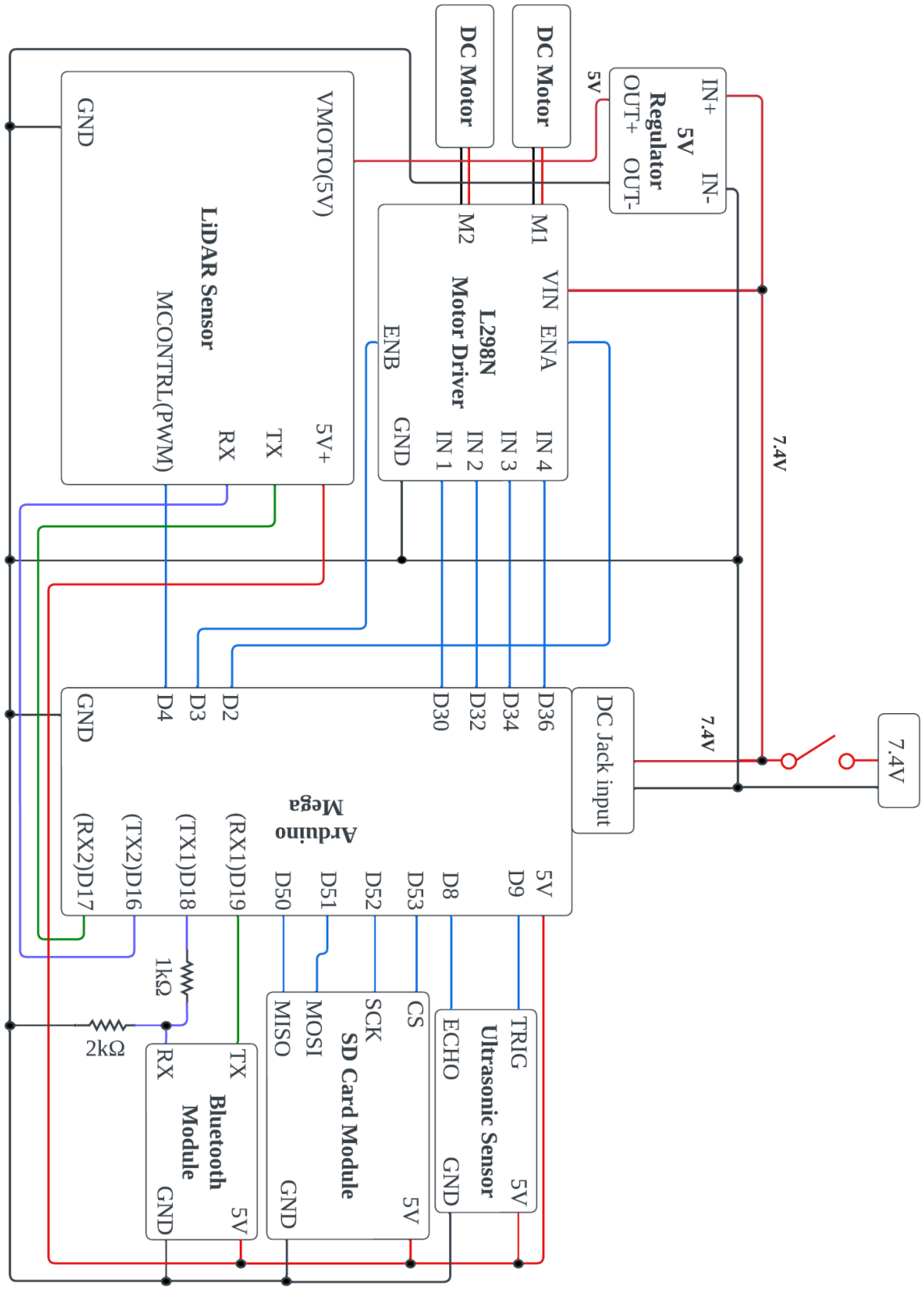

Figure 14: Hardware schematic of the robot.

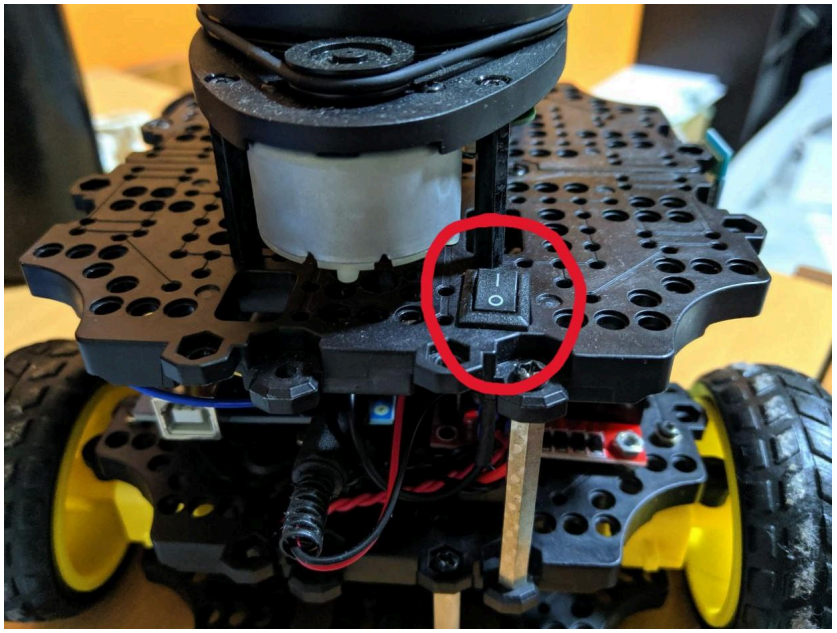

Figure 15: Safety switch on PathPilot robot.

L298N Motor Controller Truth Table			
ENA	**IN1**	**IN2**	**Description**
0	N/A	N/A	Motor A is off
1	0	0	Motor A is stopped (brakes)
1	0	1	Motor A is on and turning backwards
1	1	0	Motor A is on and turning forwards
1	1	1	Motor A is stopped (brakes)

Figure 16: L298N motor driver truth table (Miller, 2024).

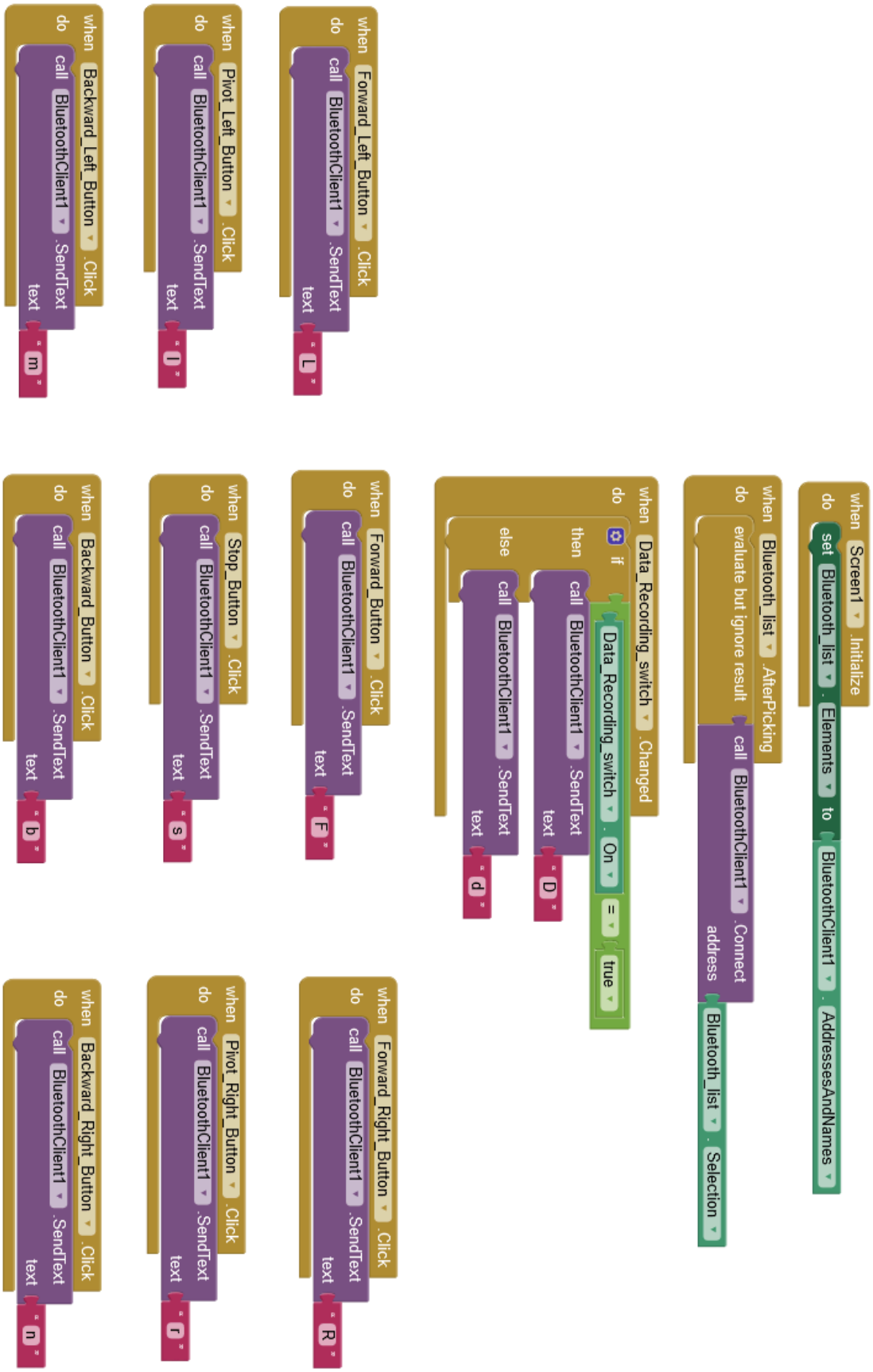

Figure 18: Programming of the Android application.

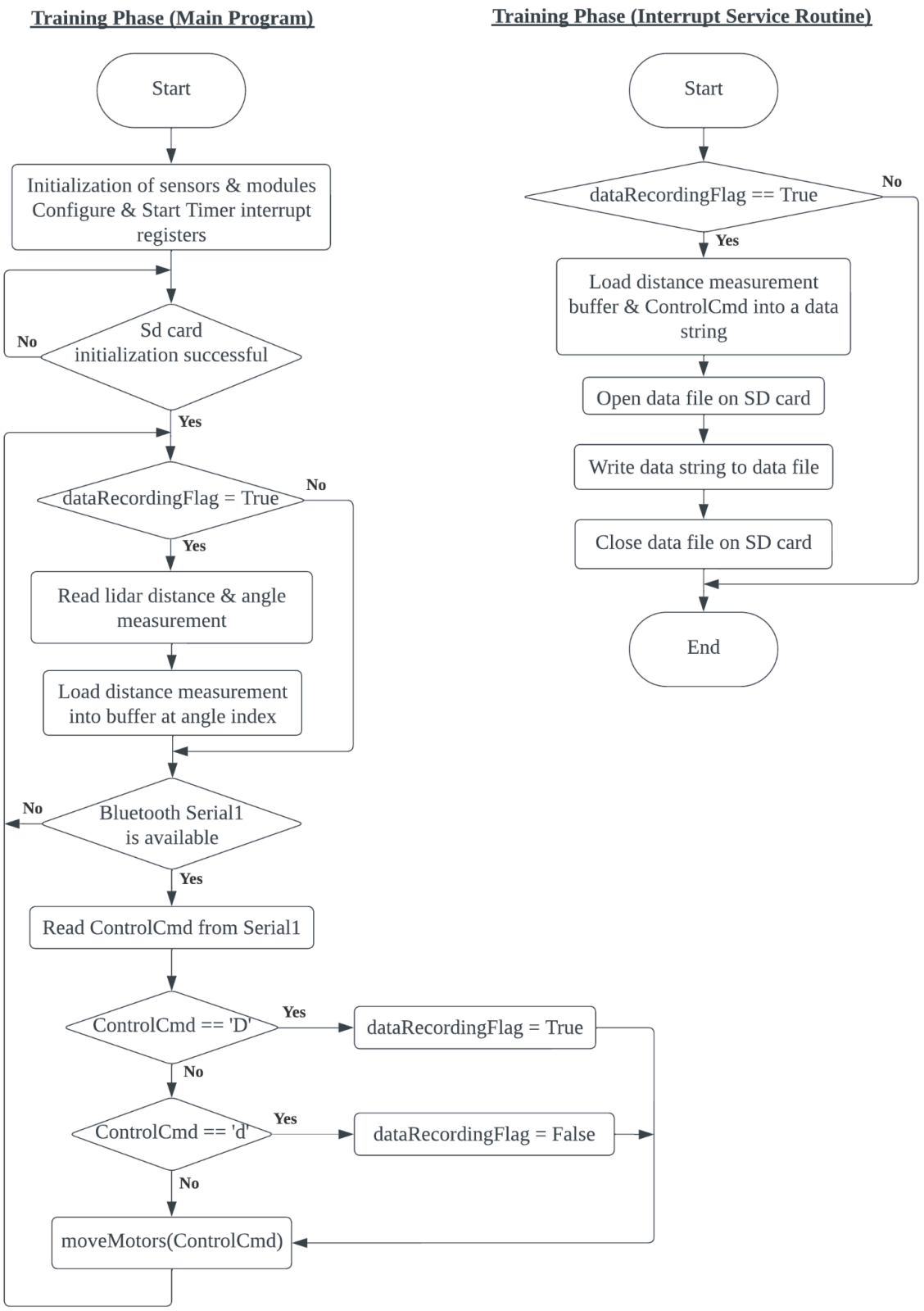

Figure 20: Design of the Arduino software and its ISR for the Training Phase of the project.

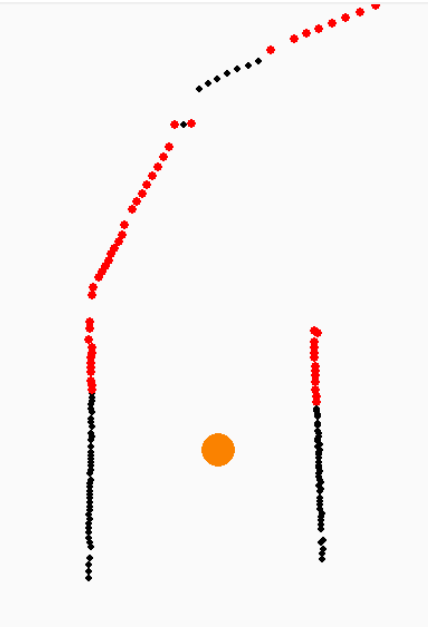

Figure 23: Screenshot of the LidarPlayback Program.

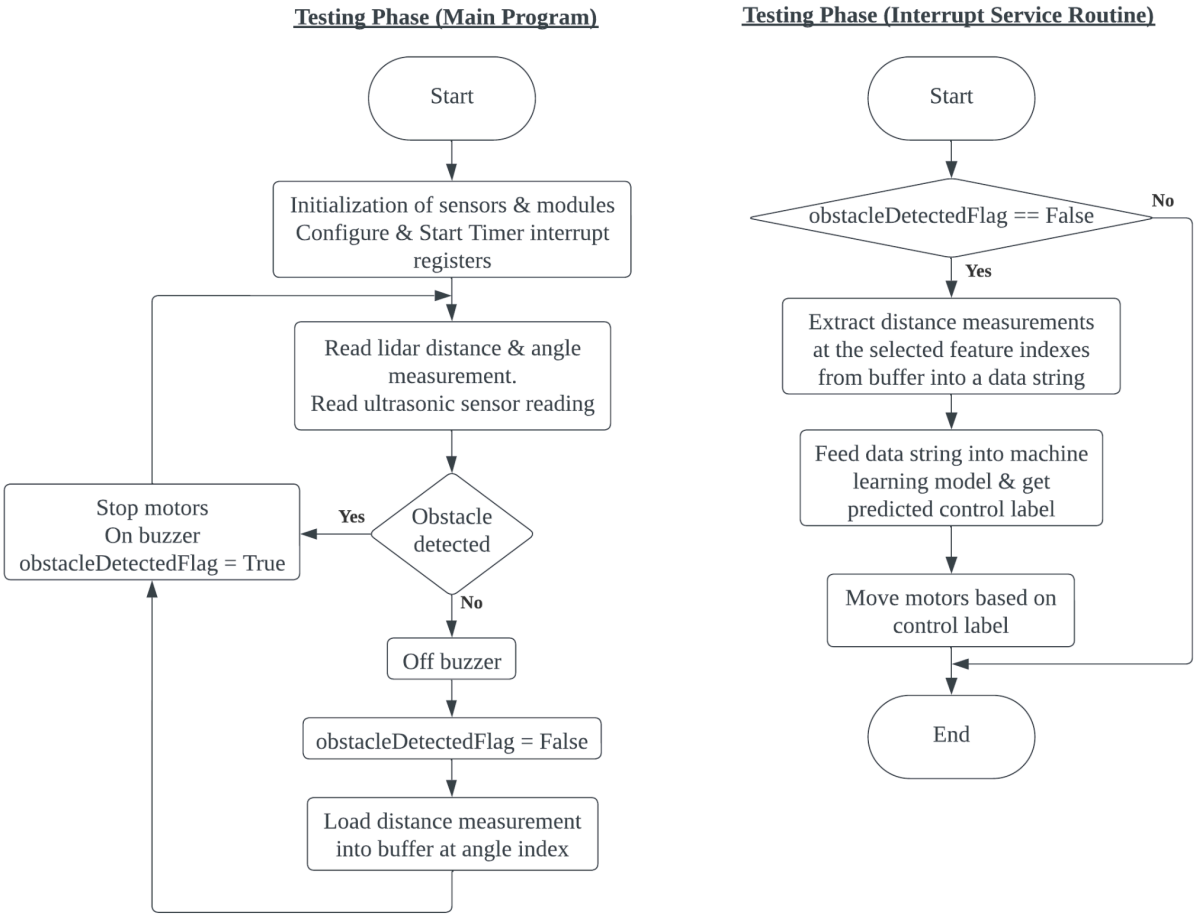

Figure 24: Design of the Arduino software and its ISR for the Testing Phase of the project.

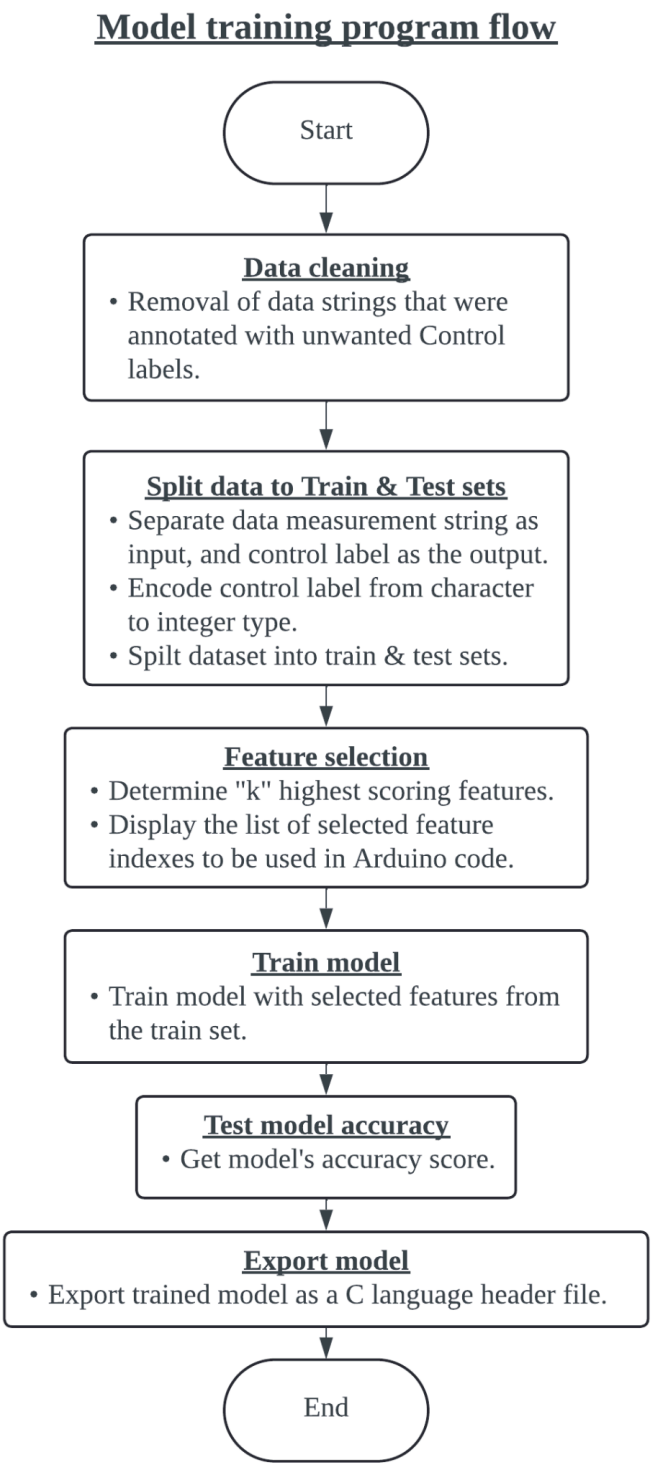

Figure 26: Program flow for training of the machine learning model.

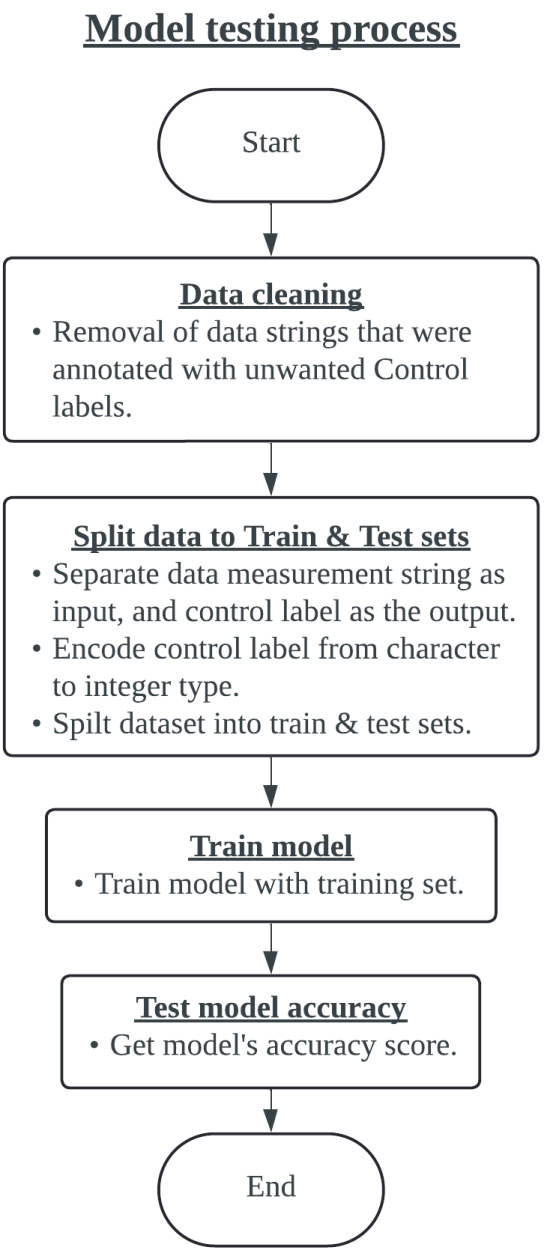

Figure 27: Process for testing and evaluating the accuracy of the machine learning models.

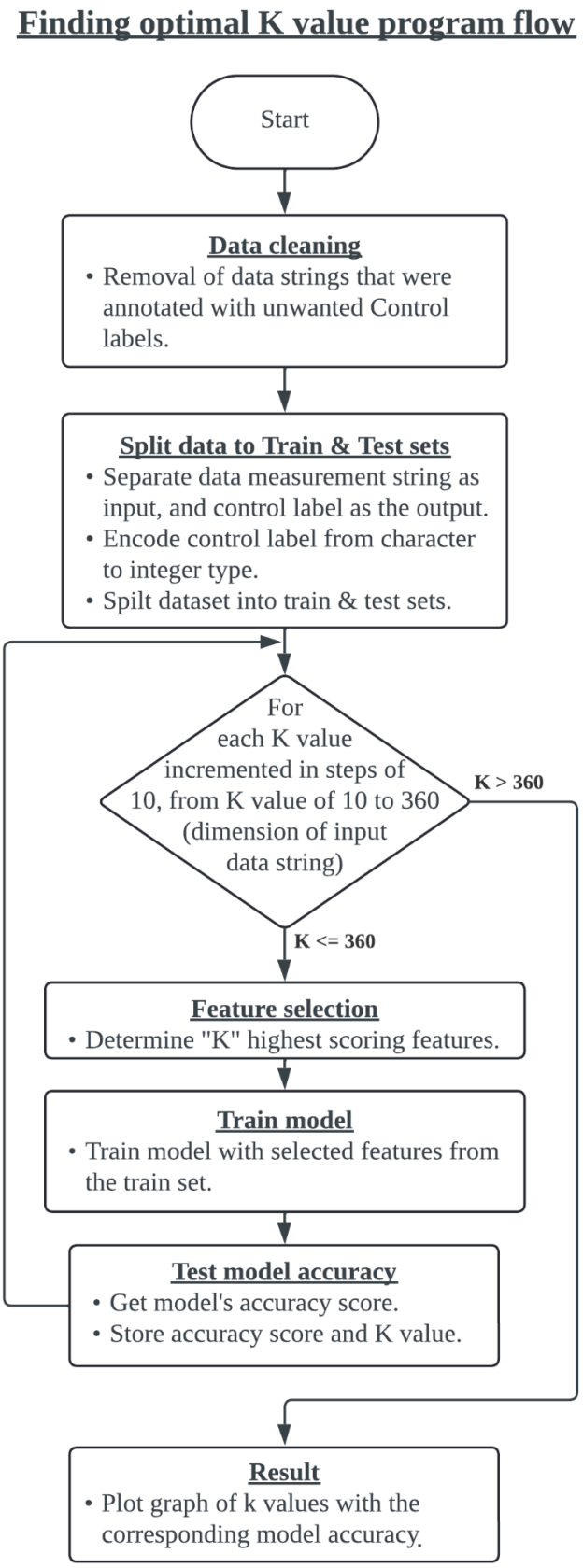

Figure 29: Program flow for the testing to determine the optimal value of K.

Figure 31: B shaped course track layout used in the testing of the PathPilot.

References

Ajani, T. S., Imoize, A. L., & Atayero, A. A. (2021). An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications. Sensors, 21(13), 4412. https://doi.org/10.3390/s21134412

AZoRobotics. (2023, May 8). Tackling the limits of robot perception and sensing. https://www.azorobotics.com/Article.aspx?ArticleID=600

Braun, A. (2022, March 30). TinyML is the most underrated field in Machine Learning and this is why | CodeX. Medium. https://medium.com/codex/tinyml-is-the-most-underrated-field-in-machine-learning-and-this-is-why-7076aed78b4f

Datamapu. (2023, November 26). Loss functions in XGBoost - DataMapu - Medium. Medium. https://medium.com/@pumaline/loss-functions-in-xgboost-c89885b57346

Dhasenfratz, A. (2017, January 6). Decision trees on embedded systems. David Hasenfratz. https://dhasenfratz.com/2016/12/23/decision-trees-on-embedded-systems/

ECOVACS US. (2024, January 5). Do robot vacuums work in dark spaces & floors? https://www.ecovacs.com/us/blog/robot-vacuum-dark-floors-environments

Ellis, C. (2022, September 24). XGBoost overfitting - Crunching the Data. Crunching the Data. https://crunchingthedata.com/xgboost-overfitting/

GeeksforGeeks. (2023, December 4). Unsupervised learning. GeeksforGeeks. https://www.geeksforgeeks.org/ml-types-learning-part-2/

GeeksforGeeks. (2024, February 22). Random Forest algorithm in machine learning. GeeksforGeeks. https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

Gholamy, A., Kreinovich, V., & Kosheleva, O. (2018, May 7). Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation. ScholarWorks@UTEP. https://scholarworks.utep.edu/cs_techrep/1209/

Immonen, R., & Hämäläinen, T. (2022). Tiny machine learning for Resource-Constrained microcontrollers. Journal of Sensors, 2022, 1–11. https://doi.org/10.1155/2022/7437023

Issuu. (2022, March 3). Limitations of LIDAR technology in robot vacuums. https://issuu.com/bikesunshinegirl/docs/what_is_good_LiDAR_robovac_and_how_it_works/s/23547462

Kostadinov, S. (2021, December 11). Understanding Backpropagation Algorithm - towards Data science. Medium. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

Machine Learning for Embedded Systems. (2018, October 15). Element14 Singapore. https://sg.element14.com/machine-learning-for-embedded-systems

Malik, S. (2020, March 23). XGBOOST: A Deep dive into boosting. dzone.com. https://dzone.com/articles/xgboost-a-deep-dive-into-boosting

Miller, L. (2024, April 25). How to Program the L298N with Arduino. Learn Robotics. https://www.learnrobotics.org/blog/how-to-program-the-l298n-with-arduino/

RPLIDAR. (2024, February 4). SLamTec RPLIDAR A1 Laser Ranging Sensor, 360° Omnidirectional LIDAR | RPLIDAR A1. https://www.waveshare.com/rplidar-a1.htm

Vanik. (2023, June 7). Colors of light - Absorption and reflection of light. US Learn. https://www.turito.com/learn/science/colors-of-light-absorption-and-reflection-of-light

Velodyne Lidar. (2022, June 3). What is lidar? Learn How Lidar Works | Velodyne Lidar. https://velodynelidar.com/what-is-lidar/#:~:text=A%20typical%20lidar%20sensor%20emits,calculate%20the%20distance%20it%20traveled.

What is supervised learning? | IBM. (2023, April 6). https://www.ibm.com/topics/supervised-learning

What is unsupervised learning? | Google Cloud. (2024, March 3). Google Cloud. https://cloud.google.com/discover/what-is-unsupervised-learning

What is XGBoost? (2024, February 3). NVIDIA Data Science Glossary. https://www.nvidia.com/en-sg/glossary/xgboost/

Why LiDAR is ideal. (2021, May 13). Quanergy Solutions, Inc. | LiDAR Sensors and Smart Perception Solutions. https://quanergy.com/blogpost/why-lidar-is-the-ideal-port-safety-solution/