Safety Car

Jorlly Chang and Pranav Peddinti

Table of Contents

1.0 Introduction.....3
2.0 Specifications.....3
Table 1: Measurements..... 4
Table 2: Tasks.....4
3.0 Design Development.....4
3.1 Hardware Design..... 5
Figure 1: Circuit Schematic..... 5
Figure 2: Printed Circuit Board.....6
Figure 3: Complete hardware setup of the Safety Car including the ToF breakout board attached to the top panel of the car..... 7
Figure 4: Main PCB in detail.....8
3.2 Software Design.....9
Figure 5: Screenshot of the Wi-Fi based control on a smartphone web browser.....9
Figure 6: Task Diagram..... 10
Figure 7: Distance measurement code..... 11
Figure 8: Braking code.....12
4.0 Results.....12
5.0 Appendices.....13
5.1 Board Design Sources.....14
5.2 Code Sources..... 14
5.3 Safety Car GitHub Repository..... 14

1.0 Introduction

Project Safety Car began as the combination of ideas relating to electronic scooters and their inability to brake quickly and safely due to the poor grip of the tires and lack of an antilock braking system (ABS) to compensate. We also wanted to explore the possibility of replicating the automatic emergency braking system (AEB) of existing cars into a smaller scale, like a remote controlled car.

The original scope of the ABS project was geared towards potential customers with existing single track vehicles like electric scooters and bicycles to have an add-on accessory package. This could also be amplified to manufacturers to incorporate the technology into future products of similar nature.

The AEB project was intended to just replicate modern automotive active safety features into a simple existing remote controlled car. As both of these technologies are essential safety features in today's cars, we wanted to combine them into a single proof of concept.

2.0 Specifications

Table 1 and Table 2, shown below, are the original measurement and task guidelines we set for the project, respectively. As mentioned later in this report, we originally wanted to include an accelerometer reading to modulate the pulsing brakes of an ABS system, but the project scope was reduced to primarily the AEB. As a result, the distance measurement, and the entire program as a whole was finalized to 20 Hz as we worked around the motor driver operating frequency.

Measurement (units)	Range	Accuracy	Frequency (Hz)
Motor (rpm)	0-700 rpm	100 rpm	20
Ground speed (m/s)	0-3 m/s	0.01 m/s	20
Distance (m)	0.01m - 2m	0.01 m	50

Table 1: Measurements

Name	Function	Frequency (Hz)
Distance	Distance Detection	40k
Controller	Bluetooth Inputs	1000
Motor	Motor Speed Variation	5000
Safety Overview	Manages Safety for System	1000
Sensor Check	Updates sensor faster than task	40k

Table 2: Tasks

The task functions had a similar narrative with respect to the original specifications. The safety overview was removed entirely and the distance detection, controller, motor actuation, and sensor frequencies were toned down to better align with the operating frequency of the components.

3.0 Design Development

We began with the basis of our project as the standard remote controlled car. This was partially due to the procurement, cost, as well as relative comparability to the original single track idea. As a result, the components of our project should be able to fit entirely within the car itself. This includes the main stock board, and ESP32, components, as well as all the associated wiring. Therefore, we chose an R/C car that was large, yet affordable enough to leave room to purchase all the other components.

The overall idea and design was to have the ESP32, motor driver, and accelerometer on the main PCB, while the time of flight (ToF) sensor would be separated and placed somewhere on the front of the vehicle to ensure its functionality without interfering with the stock appearance. This left us a choice of either at the top of the windshield like camera systems in real cars, or in the grille, like radar systems.

3.1 Hardware Design

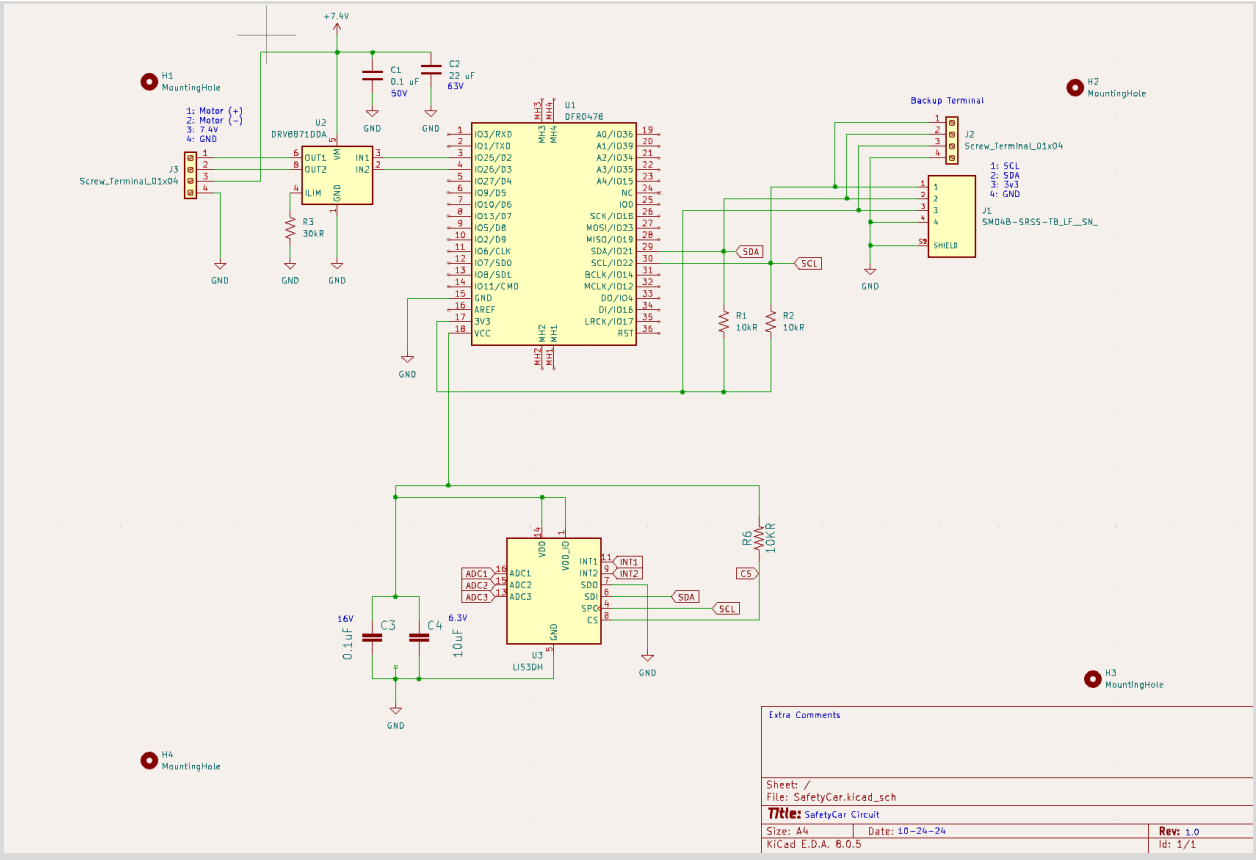

Figure 1: Circuit Schematic

Figure 1 represents the original circuit schematic of our safety car project. When originally deciding the components of our system, we followed the datasheets recommended values for resistors, capacitors, and I^2C. The circuit schematic is a faithful representation of the recommended values for each component, however due to the differing application each

component was designed for, we had to alter certain aspects. We left the time of flight sensor as an individual board, due to its nature as an isolated component in the system and its need to be closer towards the front, rather than on the PCB.

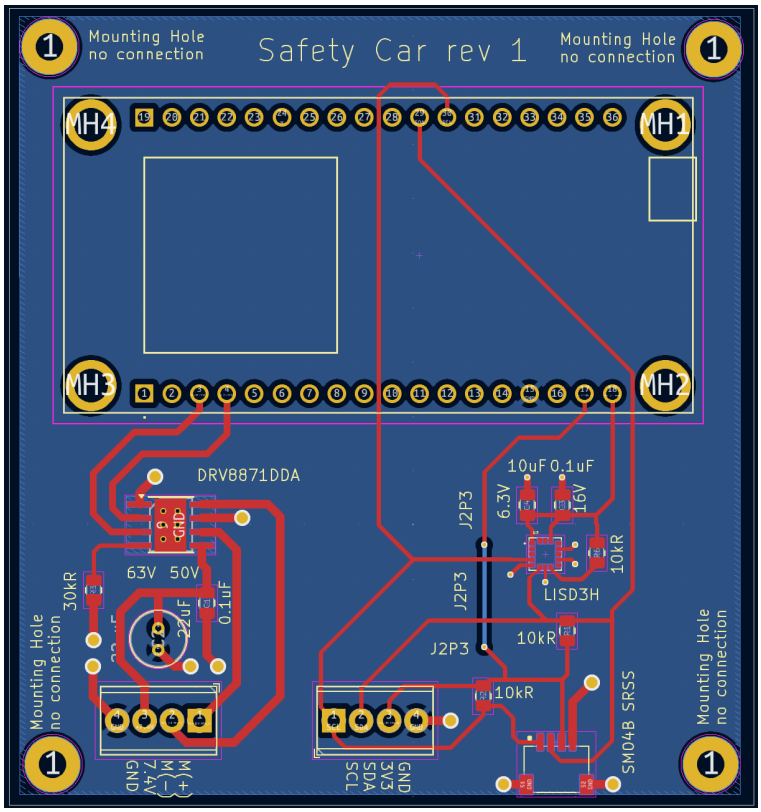

Figure 2: Printed Circuit Board

Shown in Figure 2 is the full assembly of the PCB components. With respect to Figure 1, we had to alter the values and footprints of certain components. Most noticeably, the 22µF capacitor had to be replaced with a through mounted can capacitor as there were no available surface mounts with the specified voltage ratings, referenced in the datasheet. A full surface mounted approach was initially pursued due to good grounding and cooling characteristics. Although we considered the fact that a 7.4V source was the maximum voltage each component would ever see in this board, and as a result a respective 16V rating, we decided to use the datasheets values. We did this in order to best comply with the tolerances intrinsic to the

components, as well as the fact that the original purpose of the project was to operate a much larger and powerful system. In addition, procurement of derivative components was not a hindering factor, so we thought it prudent to stick to the original values.

A particular issue we came across was the installation of an accelerometer, as due to its small size, we could not easily solder the pads of the chip. Although not a pitfall, we should have ordered an accelerometer with larger or exposed pins for ease of installation. In case the accelerometer on the main PCB was not properly soldered, we utilized our LIS3DH breakout board as a backup, which is the red board as seen in Figure 3 below.

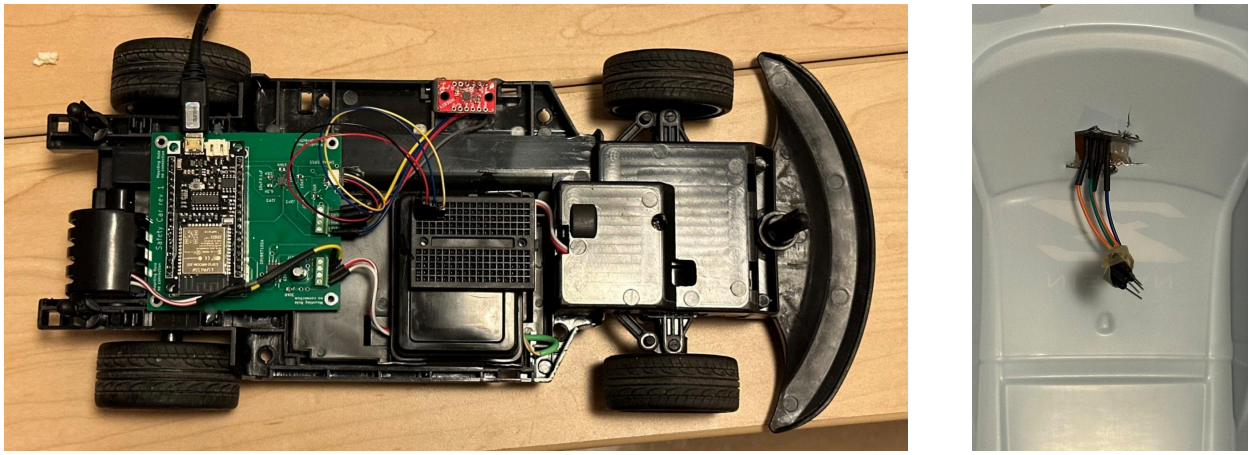

Figure 3: Complete hardware setup of the Safety Car including the ToF breakout board attached to the top panel of the car

The motor driver is connected between the primary battery of the R/C car and the drive motor, acting as a switch for the power going through them. This was done to prevent tampering with the battery wires, and keep the car easy to restore after the project completion. The downside of this, however, was that we have less control over the entire driving motion. We needed to keep the controller of the R/C car on the whole time we were using the car by taping the forward motion trigger in place to keep provide power to the motor and motor driver

constantly. With this, we could then use our own Wi-Fi controller from our computer or smartphone to control the car. We also needed to use the original controller to steer the car.

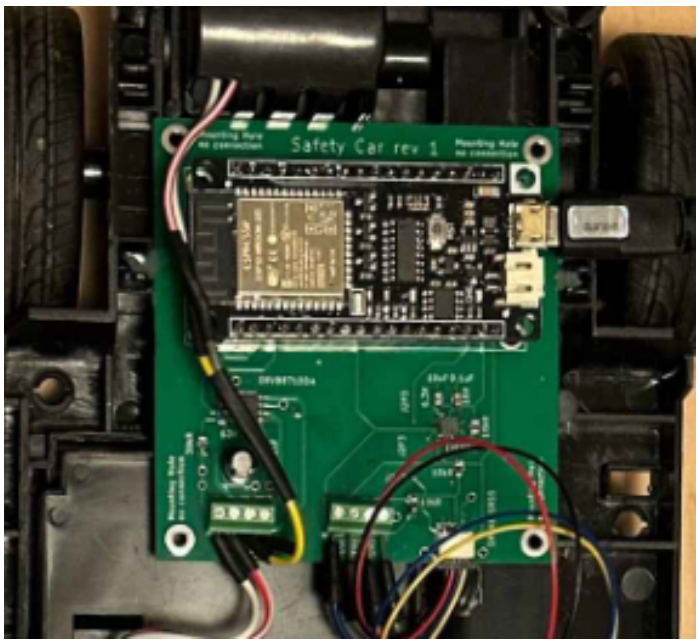

Figure 4: Main PCB in detail

Figure 4 represents the full assembly of the PCB, where all components were successfully mounted. The PCB includes the ESP32, accelerometer, motor driver, screw terminals for the motor and accelerometer, and a Stemma connector for the ToF sensor, along with a series of resistors and capacitors. One issue we had during procurement was the fact that we forgot to buy the cable for the Stemma connector, however another group was nice enough to lend us theirs. Overall, the PCB turned out great and fit perfectly into our designated area. We also utilized a breadboard to act as a mounting point for our PCB and ToF sensor, in order to preserve the modularity of the shell of the vehicle. This can be seen in the center of the car Figure 3.

3.2 Software Design

The software of the car is written in the C++ language, due to its ability to run very quickly. This is essential for safety related technology as the reaction time is fundamental to these systems. Each component, including the motor driver, accelerometer, and time of flight sensor had to be tied together in one program. As these components came on breakout boards from Adafruit and Sparkfun, we were able to utilize their example code for each component provided on their respective GitHub repositories. By selecting each code snippet for our individual functions, including cutting the power to the motor when the distance measured by the ToF sensor was one meter or less, we were able to compile a functioning program with the assistance of GitHub and Microsoft Copilot. With a little debugging for the motor control and specifying whether we wanted the car to base the distance measurement on the closest or furthest detected object, as well as integrating the Wi-Fi control, we were able to achieve acceptable braking performance.

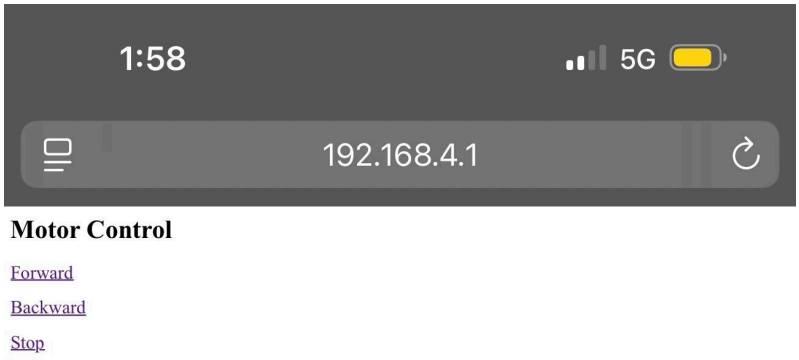

Figure 5: Screenshot of the Wi-Fi based control on a smartphone web browser.

The Wi-Fi control of the car, seen in Figure 5, works like a gear shifter in a car, with the ability to shift from park to drive or reverse utilizing a webpage that can be accessed on any web browser on a computer or smartphone. This is the original design, and once we understood how it functioned, we changed the names to Drive, Reverse, and Park, like a traditional gear shifter.

These enable the car to move forward and backwards at the touch of a button. When going forwards, the vehicle goes at a consistent speed until the ToF sensor detects an object and stops the motor. In reverse, the data obtained from the ToF sensor is irrelevant. While in park, the motor receives no power at all, and the car remains at a standstill. We can see how these work together in the task diagram, Figure 6, below.

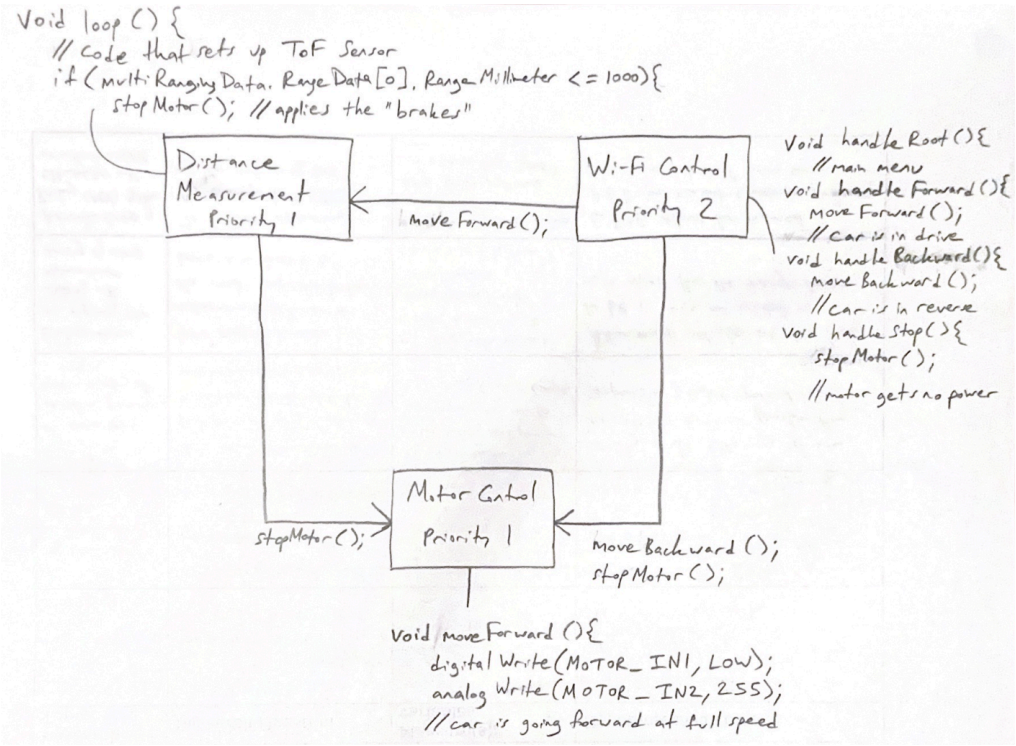

Figure 6: Task Diagram

The automatic braking function works by taking the data from the ToF sensor, checking whether the measured distance is less than one meter, and then telling the motor driver to cut power to the motor if this condition is met, as seen in Figure 7. The distance measurements are ignored when the car is in park or reverse, as the collision avoidance is forward facing.

```
if ((!status) && (NewDataReady != 0)) {
  status = sensor_vl53l4cx_sat.VL53L4CX_GetMultiRangingData(pMultiRangingData);
  no_of_object_found = pMultiRangingData->NumberOfObjectsFound;
  snprintf(report, sizeof(report), "VL53L4CX Satellite: Count=%d, #Objs=%1d ", pMultiRangingData->Strea
  Serial.print(report);
  for (j = 0; j < no_of_object_found; j++) {
    if (j != 0) {
      Serial.print("\r\n                                ");
    }
    Serial.print("status=");
    Serial.print(pMultiRangingData->RangeData[j].RangeStatus);
    Serial.print(", D=");
    Serial.print(pMultiRangingData->RangeData[j].RangeMilliMeter);
    Serial.print("mm");
  }
  Serial.println("");
  if (status == 0) {
    status = sensor_vl53l4cx_sat.VL53L4CX_ClearInterruptAndStartMeasurement();
  }

  // Check if the distance is less than or equal to 1 meter (1000 mm)
  if (MultiRangingData.RangeData[0].RangeMilliMeter <= 1000) {
    brakeMotor(); // Apply the "brakes" when the distance is less than or equal to 1 meter
  }
}
```

Figure 7: Distance measurement code

```
/**
 * @brief Stops the motor.
 *
 * Sets both MOTOR_IN1 and MOTOR_IN2 to LOW to stop the motor.
 * Outputs a message to the Serial Monitor indicating the motor is stopped.
 *
 * @note This function was shown during the demonstration and cuts power to the motor,
 * resulting in the car coasting to a stop.
 */
void stopMotor() {
  analogWrite(MOTOR_IN1, 0);
  analogWrite(MOTOR_IN2, 0);
  Serial.println("Motor stopped");
}

/**
 * @brief Brakes the motor.
 *
 * Sets both MOTOR_IN1 and MOTOR_IN2 to HIGH to brake the motor.
 * Outputs a message to the Serial Monitor indicating the motor is braking.
 *
 * @note This function is a work in progress and was not shown during the demonstration.
 */
void brakeMotor() {
  digitalWrite(MOTOR_IN1, HIGH);
  digitalWrite(MOTOR_IN2, HIGH); // Both pins set to HIGH to brake the motor
  Serial.println("Motor braking...");
}
```

Figure 8: Braking code

We have been working on improving the software of the car to improve the braking performance and add more functionality to the Wi-Fi control, including keeping the motor

powered instead of letting it coast, as seen in Figure 8. This code snippet is the original braking method which resulted in the car to coast, and what we are working to replace it with to properly stop the car in a shorter distance. Another feature that could be added is the ability to steer the car with our Wi-Fi control utilizing the additional motor driver breakout board we have acquired, eliminating our reliance on the controller that came with the R/C car.

4.0 Results

The project was a success overall, as we completed our main objective of having a vehicle stop under its own initiative when the vehicle sensed an object it would collide with immediately, irrespective of the user's inputs. We tested the system using a full throttle drive, directed at a wall and the car cut power to its motors before a collision could occur.

While testing the braking performance of the R/C car, we found that if the car was going too fast, the stopping task would greatly reduce the velocity to minimize the impact of the collision, however the R/C car would still hit the wall. This issue is prevalent in real world application of AEB systems on most cars, where the braking performance is only optimal at speeds of around 30 miles per hour or less. This is essential for residential and high traffic areas as is the nature of their respective safe and low speed requirements. At higher speeds, the vehicle’s AEB can help reduce the magnitude of the impact, but is unable to completely negate it. In order to solve this issue relative to our project, we are working to include some code to keep the motor powered, but static.

One of the original reach goals we had was the ABS, which downscaled to the scale of the R/C car, would need to reverse the motors in quick succession in order to imitate the pulse braking of an actual ABS. By utilizing the motor driver and data obtained from the

accelerometer, the ABS would prevent the car from drifting when suddenly braking, which occurred when the R/C car was being used normally before modification.

Another issue that occurred sometimes was that the ToF sensor was prone to false positives, despite being in a controlled and clear environment. This could be due to ambient light sources or reflections. Since the car relies on the object detection of the ToF sensor, the extra objects being measured occasionally caused the car to brake unexpectedly. Overall, this issue was less severe than the alternative, of having too many true negatives which would lead to ur probably needing to buy another car. The most definite solution is to use a ToF sensor optimized for resolution. rather than range, as our six meter detection length may have been unnecessary. A higher resolution ToF sensor with a range of around three meters would suffice.

As mentioned earlier in section 3.2, adding another motor driver to control the front wheel steering would allow us to eliminate our need to use the original R/C car controller completely, and just use the Wi-Fi based webpage to control the car. While we were not able to incorporate these features during the course of this project, we were able to learn a lot about integrating software and hardware design to make something really cool and interesting, and can use these methods of programming and PCB design to create new products in the future.

5.0 Appendices

Below are the references we used to design our board and program the ESP32. For the board design, we observed the components on the breakout boards, such as the primary chip along with the resistors and capacitors, and incorporated these into our PCB design. The footprints for these components were also obtained from SnapMagic and the product listings for the breakout boards.

For the code, we looked at the examples provided by the manufacturers of the breakout boards, including Adafruit and SparkFun, and used these as the basis of our ESP32 software. These code snippets were then combined together with the help of Microsoft and GitHub Copilot.

5.1 Board Design Sources

https://www.adafruit.com/product/3190

https://www.adafruit.com/product/5425

https://www.snapeda.com/

https://www.sparkfun.com/products/13963

5.2 Code Sources

https://github.com/espressif/arduino-esp32.git

https://github.com/sparkfun/SparkFun_LIS3DH_Arduino_Library.git

https://github.com/stm32duino/VL53L4CX.git

https://learn.adafruit.com/adafruit-drv8871-brushed-dc-motor-driver-breakout/usage

5.3 Safety Car GitHub Repository

https://github.com/jchan293/SafetyCar.git