**EE 59869 - Senior Design II (Spring 2024)**

**Final Report**

**Blind Sight -** LIDAR Motorcycle Collision Blind Spot System

**Team Members -** David Wang, Iftiak Ahmed, Kevin Campoverde, Li Ting Melisa Chen Chen.

[**Project Description 1**](#_te2oi8f4nswo)

[**Project Approach 2**](#_v0qfor9t4830)

[**Arduino Code Implementation 3**](#_lqxo52kgilep)

[Header Files & Variables 3](#_1zaoa7z6071f)

[Program Body 5](#_dnkra9iv3uk2)

[Functions 5](#_tvkhu5btwhci)

[**3D Design (Motivation, steps, and printing process) 10**](#_b2jvwdlqte0)

[**Testing Plan 13**](#_ft0abylww0yw)

[**Future Work 13**](#_ea15aldtxgrn)

[**References 15**](#_a0czg2qfhrtz)

[**APPENDIX A: CODE 16**](#_e2bemm50c58)

# 

# **Project Description**

In an era of rapid urbanization, shifting consumer preferences, and the relentless pursuit of convenience, delivery apps have emerged as a transformative force, reshaping how goods and services reach our doorsteps. Digital platforms such as Doordash, Grubhub, and Uber Eats powered by cutting-edge technology and an intricate network of couriers, have revolutionized the last-mile delivery industry and catalyzed a significant shift towards sustainable and efficient means of transportation [2].

One remarkable consequence of this technological evolution is the substantial increase in the use of bikes, motorcycles, and other one-rider vehicles as preferred delivery modes. The traditional internal combustion engine-based delivery fleets are slowly giving way to more agile, eco-friendly, and cost-effective alternatives[3].

The inclusion of bikes, motorcycles, and other single-rider vehicles introduces an elevated risk to their operators in the form of potential collisions with various obstacles, including other vehicles [1]. The implementation of a straightforward sensor designed to identify and lower the risk of collisions with other vehicles and other obstacles on the road., leveraging the advancements in LIDAR technology, holds the potential to avert such incidents. LIDAR's noteworthy utility lies in its ability to furnish precise data, which can be effectively processed and applied for enhanced safety measures. Expanding further on the concept of LIDAR, our focus is particularly on pulsed lasers. These pulsed lasers emit rapid bursts of light to swiftly assess the objects in the path ahead. Our product will be designed to be practical and accurate for the riders.

Our central challenge is to enhance the safety of bike, motorcycle, and single-rider vehicle operators by providing an affordable, easy-to-use, and accurate device for the user. The primary goal was to design a reliable low-cost device that wouldn't divert drivers' attention from potential road hazards by giving the minimum information needed of condition in the biker’s blind spot. Our device must work seamlessly, even in challenging situations, alerting the driver when necessary to avoid distractions that could lead to accidents. To address these concerns, we are employing LIDAR technology to ensure the utmost safety through highly accurate information provided by our sensors. A secondary challenge was ensuring that our device was easy to use, inconspicuous, preventing overreliance on the technology, and activating effectively only when needed by the rider.

# **Project Approach**

Various things need to be considered to build a project of this type. To do so, we consider the following elements for the project:

* Need to adequately alert the rider of danger without interfering with motor controls.
* Placement of the device module and flash warning LEDs needed to be placed in ideal locations.
* If the module were placed on the rear pillion seat of the motorcycle, the detection angle for the LIDAR sensors would be too high and not be able to detect smaller-sized vehicles.
  + We decided to place the module as low as possible on the vehicle.
* Communication with experienced riders needed to take place to determine optimal performance for the device.

# **Arduino Code Implementation**

We'll dive into the inner workings of the code that powers our Arduino-based blind spot detection system, exploring how it acquires data from LIDAR sensors, calculates alert and warning zones, and controls LEDs to provide visual feedback based on detected distances. The primary goal of this program is to create an intelligent system that can detect objects within predefined ranges and notify users through LED indicators. By understanding how each function contributes to the overall functionality, we can gain insight into the seamless coordination of hardware components and software logic in achieving our system's objectives.

## Header Files & Variables

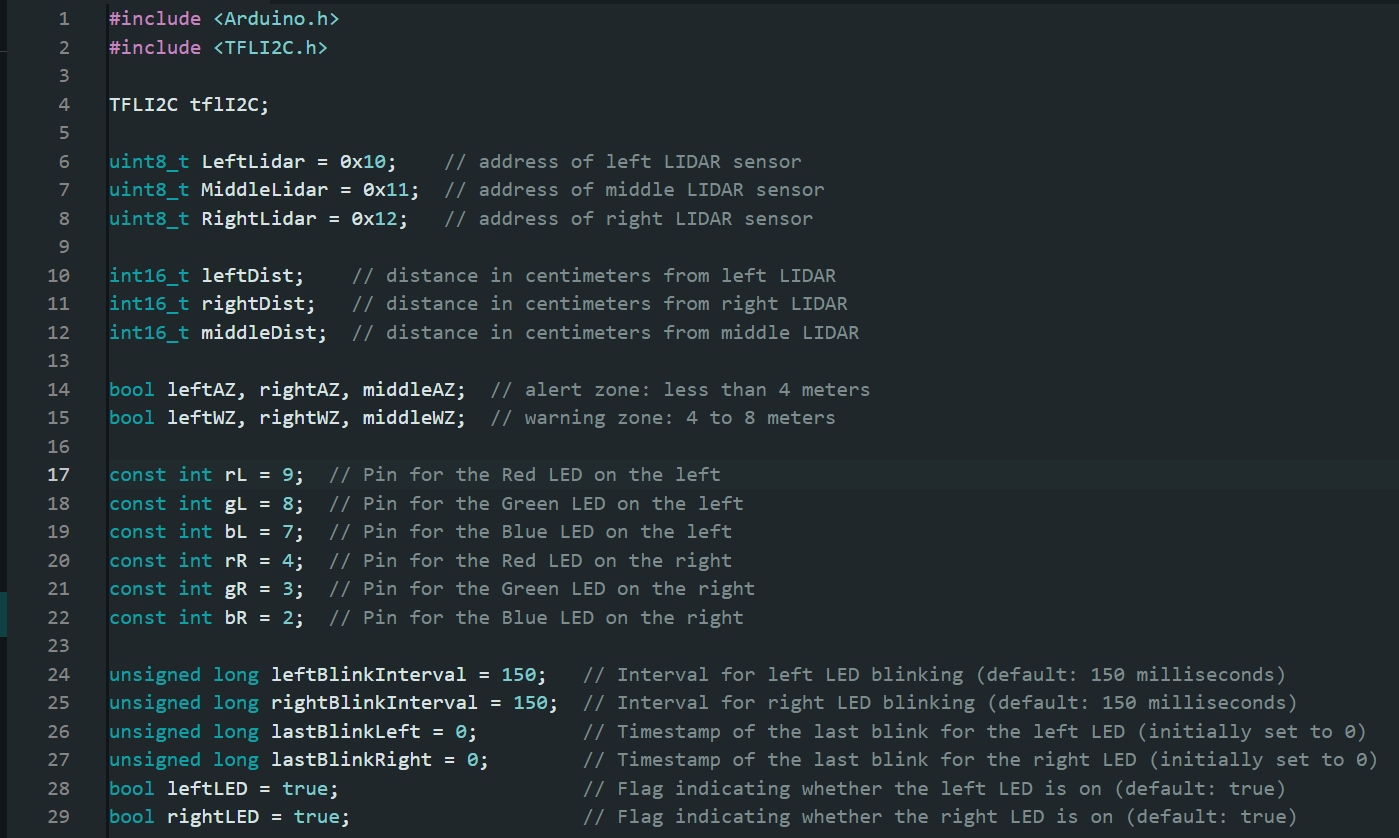
First, we will discuss the library and variables that are used in the program and their functionalities.

Header file:

* #include <Arduino.h>: the Arduino library for the sketch
* #include <TFLI2C.h>: A custom library create by Bud Ryerson [4] for working with LIDAR sensors via I2C. We will discuss more about the functionality

Global Variables:

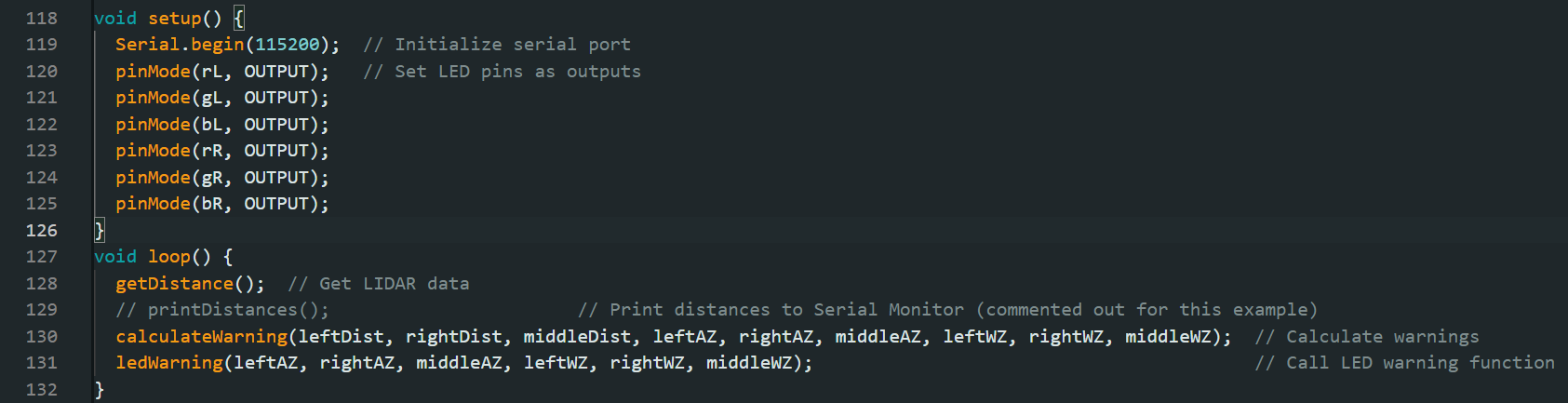
* TFLI2C tflI2C: We then declare an object to interface with the LIDAR sensors.
* uint8\_t LeftLidar, MiddleLidar, RightLidar: Addresses of the left, middle, and right LIDAR sensors on the I2C bus.
* int16\_t leftDist, rightDist, middleDist: Variables to store distances from left, right, and middle LIDAR sensors in centimeters.
* bool leftAZ, rightAZ, middleAZ, leftWZ, rightWZ, middleWZ: Flags indicating alert and warning zones for the LIDAR data.
* Constants for LED pins (rL, gL, bL, rR, gR, bR) to make it addressable RGB LED and default blink intervals (leftBlinkInterval,rightBlinkInterval).
* unsigned long lastBlinkLeft, lastBlinkRight: Timestamps for the last blink of the left and right LEDs.
* bool leftLED, rightLED: Flags indicating whether the left and right LEDs are currently on.



## **Program Body**

Arduino program uses the setup() function to initialize serial communication, Wire library for I2C and sets LED pins as outputs. Then the loop() function continuously performs the main operations:

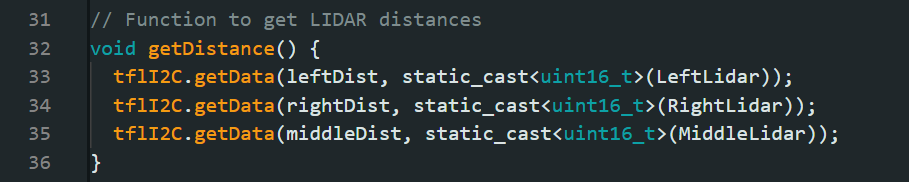
* Calls getDistance() to fetch LIDAR data.
* Optionally prints distances using printDistances() for testing. (commented out).
* Calls calculateWarning() to determine alert and warning zones and update LED blink intervals.
* Calls ledWarning() to trigger LED warnings based on zones and blink intervals.



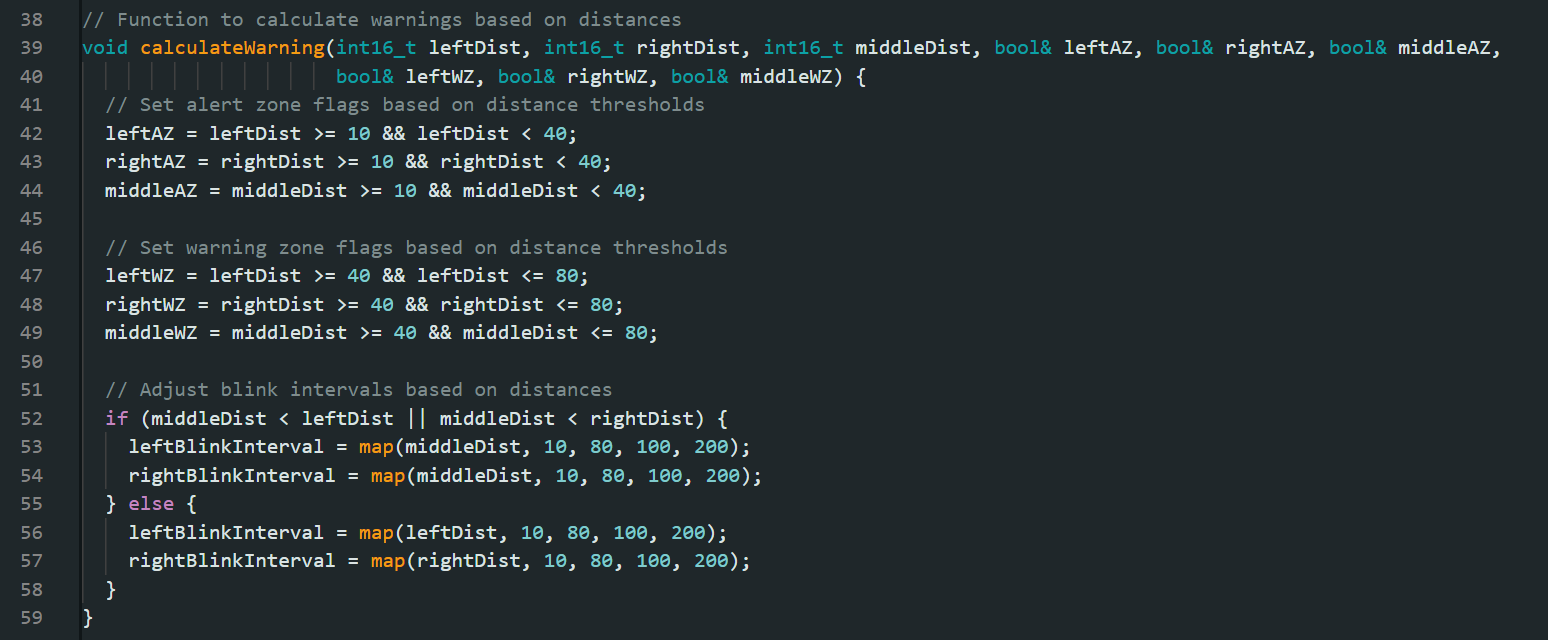
## **Functions**

Let's delve into the workings of each function in the code:

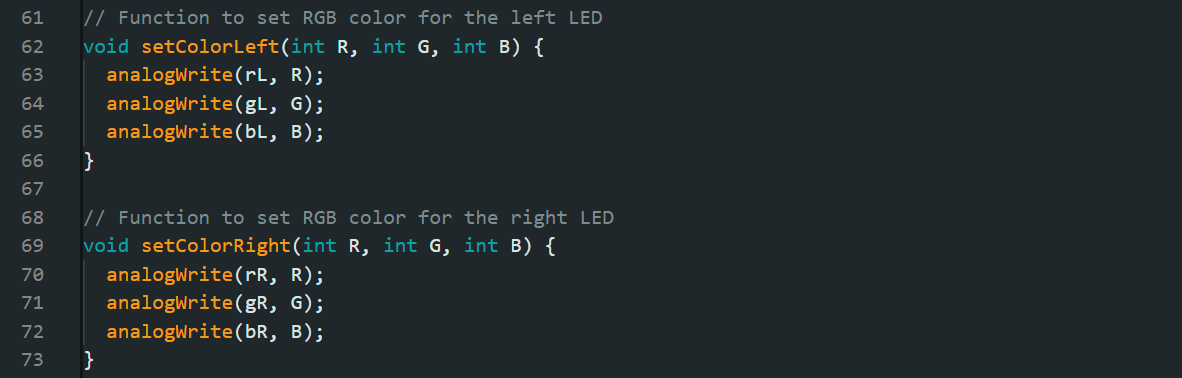
1. `void getDistance():
   1. This function retrieves distance data from the LIDAR sensors using the TFLI2C library.
   2. It uses the `tflI2C.getData()` method to fetch distance values from the left, right, and middle LIDAR sensors and stores them in the respective variables (`leftDist`, `rightDist`, `middleDist`).
   3. The tflI2C.getData()function in the program utilizes the TFLI2C library to retrieve data from LIDAR sensors connected via the I2C communication protocol. This function is designed to fetch three main measurement values distance, flux (indicate the strength or quality of the signal), and temperature of the chip, and the function also requires the correct I2C address (address) in the range of 0x08 to 0x77 (8 to 119) to communicate with the specific LIDAR sensor. We separately set the address of each LIDAR using the same library.
   4. Seen our program only uses distance measurement we fetched only distance using the function getData(dist, addr)



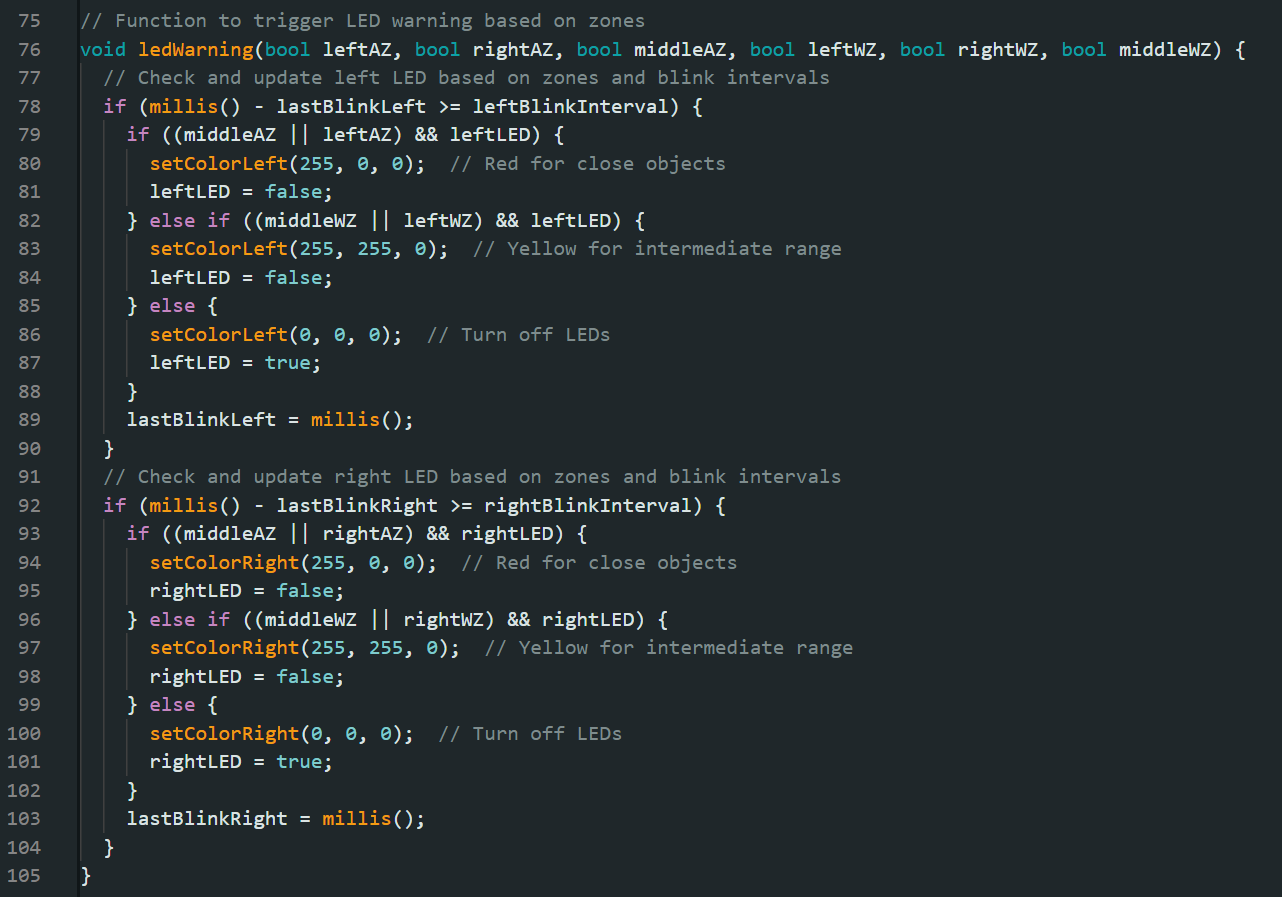
1. void calculateWarning(...)
   1. This function calculates alert and warning zones based on LIDAR distances and updates blink intervals for the LEDs accordingly to represent the closing of approaching vehicles.
   2. It takes in parameters representing the distances from the left, right, and middle LIDAR sensors (`leftDist`, `rightDist`, `middleDist`) as well as several boolean flags for alert and warning zones (`leftAZ`, `rightAZ`, `middleAZ`, `leftWZ`, `rightWZ`, `middleWZ`).
   3. The function sets the alert zone flags (`leftAZ`, `rightAZ`, `middleAZ`) to true if the corresponding distances fall within the alert zone (e.g., less than 400 centimeters).
   4. Similarly, it sets the warning zone flags (`leftWZ`, `rightWZ`, `middleWZ`) to true if the distances are within the warning zone (e.g., between 400 and 800 centimeters).
   5. The function then adjusts the blink intervals for the left and right LEDs based on the middle distance compared to the left and right distances. If the middle distance is smaller than the left or right distance, it updates both blink intervals to a mapped value based on the middle distance; otherwise, it updates each blink interval based on the corresponding left or right distance.



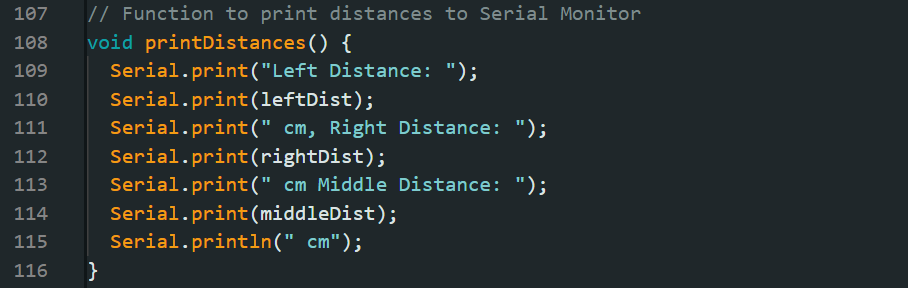
1. void setColorLeft(...)`, `void setColorRight(...)`:
   1. These functions set the RGB colors for the left and right LEDs, respectively.
   2. They take in parameters representing the intensity of the red (`R`), green (`G`), and blue (`B`) colors and use `analogWrite()` to set the corresponding PWM (Pulse Width Modulation) values to control the LED colors.



1. void ledWarning(...)`
   1. This function triggers LED warnings based on the alert and warning zones calculated earlier.
   2. It takes in parameters representing the alert and warning zone flags for the left, right, and middle zones (`leftAZ`, `rightAZ`, `middleAZ`, `leftWZ`, `rightWZ`, `middleWZ`).
   3. The function uses `millis()` to check the time elapsed since the last blink of each LED and triggers a new blink based on the configured blink intervals (`leftBlinkInterval`, `rightBlinkInterval`).
   4. If an alert zone or warning zone condition is met and the LED is currently on (`leftLED` or `rightLED` is true), it sets the LED color accordingly (e.g., red for alert, yellow for warning). If the condition is not met, it turns off the LED.



1. `void printDistances()`:
   1. This function prints the distances from the LIDAR sensors to the Serial Monitor.
   2. It uses `Serial.print()` to display the distances in centimeters for the left, right, and middle sensors.
   3. This function does not have any use in the actual device but its a significant part of developing the program.



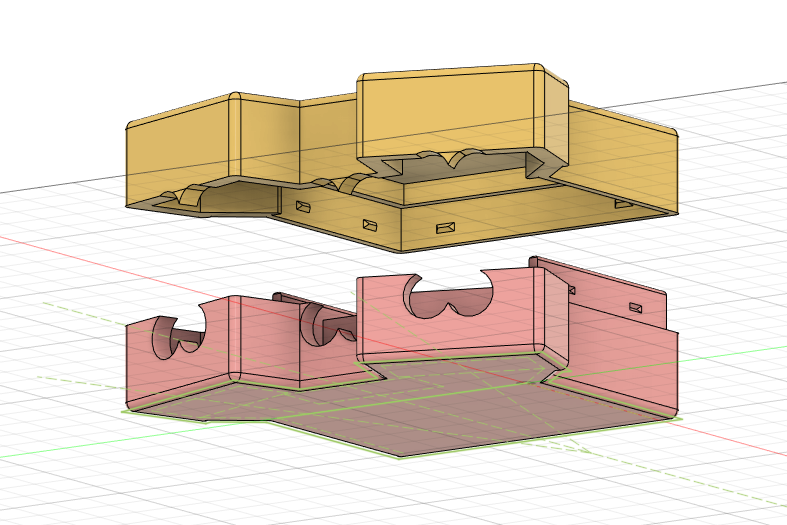
In summary, these functions work together to fetch LIDAR data, calculate alert and warning zones, update LED colors and blink intervals, and trigger LED warnings based on proximity to objects as measured by the LIDAR sensors.

# **3D Design (Motivation, steps, and printing process)**

In order for our work to be able to withstand some severe weather, such as rain or long-term exposure to the sun, we decided that we needed a case that could put our components inside to deal with the damage to these parts. So we will design the casing of our product based on the components and connections we use. Here we use Fusion 360 as our 3D design tool.

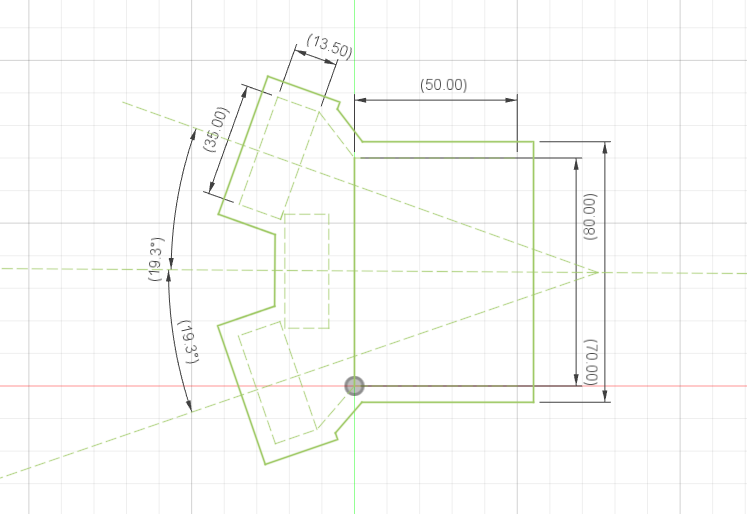
The components we will put in the design case are:

* PCB board
* Arduino Nano ESP32
* DC Power Jack
* TF-luna \*3



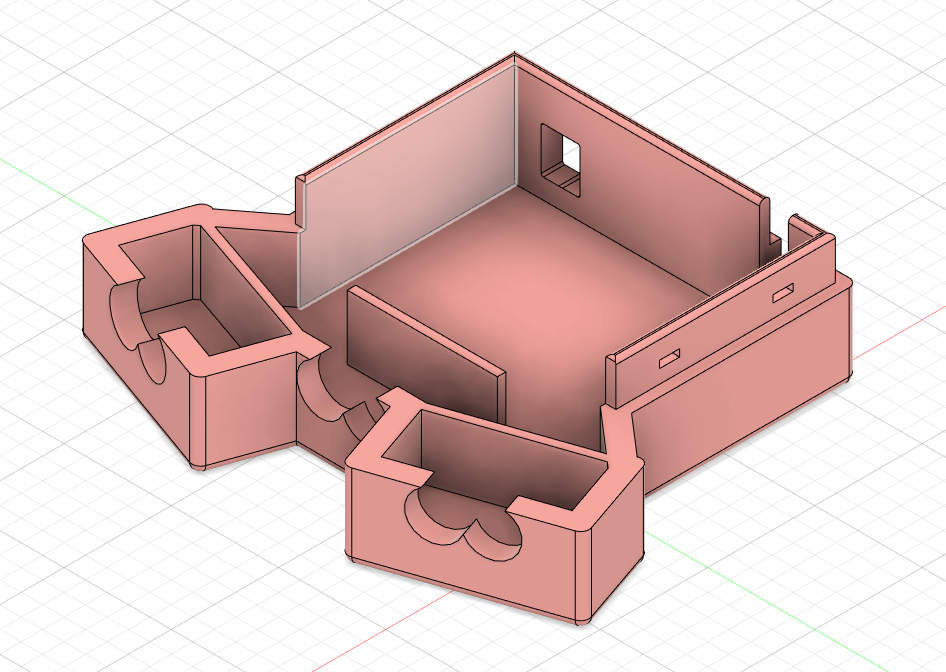
**Steps:**

1. Bottom draft
   1. Size: Among these components, we need to find the largest one, which is the PCB board, and then fix the internal area of the bottom to the size of the PCB board. Here we can add 1mm to prevent it from being too close and making it difficult to move the components.
   2. Angle: When designing the angle of the sensor, we considered the width of U.S. lanes and the average width of cars. After calculation, the angle between the sensors on both sides and the center line is designed to be 19.3 degrees.
   3. Position: In order to minimize the size of our project, we designed the box in the shape of a T-shirt, and placed the three sensors on the two sleeves and collar, close to the PCB board.



1. Hole design

Three holes are opened on the front of the box, corresponding to the lenses of the three sensors. The openings in this design completely fit the shape of the sensor lenses. Two holes were opened on the back of the box, one is to allow the LED to extend out of the box, and the other is to connect the power supply.



1. Buckle design

The buckles used in our design are embedded in the top and bottom of the box. The male buckle is at the top of the box and the female buckle is at the bottom of the box, so it does not affect the appearance of the lines and design. This design increases the firmness by increasing the friction between the two parts. It can be fixed well and is not very difficult to open.



**Printing**

Our design was completed using a 3D printer in the Market Space inside the grove school's basement. This took a total of around nine hours of printing time as we split it into two parts, five hours for the bottom and four hours for the top.

# **Testing Plan**

To test the device, several requirements and factors needed to be set and needed for.

* A motorcycle and another motor vehicle.
* An environment/location to be chosen in which the vehicles can operate safely.
* Testing the device
  + Testing if the left LED sensor alerts the rider when the device detects vehicles in the left blind spot.
  + Testing if the right LED sensor alerts the rider when the device detects vehicles in the right blind spot.
  + Testing if both LED sensors alert the rider when the device detects vehicles in directly behind the rider.

# **Future Work**

There are many areas to improve upon for this product.

* We would have liked to test our product under various weather conditions such as fog or rain but unfortunately riding in these conditions is not safe for testing purposes.
* Another upgrade we would have liked to improve upon would be to add LIDAR sensors throughout the vehicle to give a true 360° collision detection device but given our budget, this was not possible.
* We determined after some research that audio warning signals can be distracting to the driver which is why we chose to eliminate this portion of our project however, in the future, we would like to incorporate both audio and visual warnings safely and concisely.
* Modern cars have safety features such as auto braking. With a 360° FOV module, this could very well be possible on a motorcycle as well so this could also be a possible path to our project.

# 

# **References**

1. Cosimo Lucci, Trevor Allen, Niccolò Baldanzini & Giovanni Savino. (2022) Motorcycle curve assist A novel approach based on active speed control for crash injury reduction. Traffic Injury Prevention 23:sup1, pages S56-S61.
2. Ahuja, K., Chandra, V., Lord, V., & Peens, C. (2021, September 22). *Ordering in: The rapid evolution of food delivery*. McKinsey & Company.
3. *Motorcycles - Injury Facts*. (2023, April 25). Injury Facts.
4. B. (n.d.). GitHub - budryerson/TFLuna-I2C: Arduino library for the Benewake TFLuna LiDAR distance sensor in the I2C communication mode. GitHub. <https://github.com/budryerson/TFLuna-I2C>

# **APPENDIX A: CODE**

| #include <Arduino.h> #include <TFLI2C.h>  TFLI2C tflI2C;  int16\_t leftDist; // distance in centimeters from left LIDAR int16\_t rightDist; // distance in centimeters from right LIDAR int16\_t middleDist; // distance in centimeters from middle LIDAR  bool leftAZ, rightAZ, middleAZ; // alert zone: less than 4 meters bool leftWZ, rightWZ, middleWZ; // warning zone: 4 to 8 meters  uint8\_t LeftLidar = 0x10; // address of left LIDAR sensor uint8\_t MiddleLidar = 0x11; // address of middle LIDAR sensor uint8\_t RightLidar = 0x12; // address of right LIDAR sensor  const int rL = 9; // Pin for the Red LED on the left const int gL = 8; // Pin for the Green LED on the left const int bL = 7; // Pin for the Blue LED on the left const int rR = 4; // Pin for the Red LED on the right const int gR = 3; // Pin for the Green LED on the right const int bR = 2; // Pin for the Blue LED on the right  unsigned long leftBlinkInterval = 150; // Interval for left LED blinking (default: 150 milliseconds) unsigned long rightBlinkInterval = 150; // Interval for right LED blinking (default: 150 milliseconds) unsigned long lastBlinkLeft = 0; // Timestamp of the last blink for the left LED (initially set to 0) unsigned long lastBlinkRight = 0; // Timestamp of the last blink for the right LED (initially set to 0) bool leftLED = true; // Flag indicating whether the left LED is on (default: true) bool rightLED = true; // Flag indicating whether the right LED is on (default: true)  // Function to get LIDAR distances void getDistance() {  tflI2C.getData(leftDist, static\_cast<uint16\_t>(LeftLidar));  tflI2C.getData(rightDist, static\_cast<uint16\_t>(RightLidar));  tflI2C.getData(middleDist, static\_cast<uint16\_t>(MiddleLidar)); }  // Function to calculate warnings based on distances void calculateWarning(int16\_t leftDist, int16\_t rightDist, int16\_t middleDist, bool& leftAZ, bool& rightAZ, bool& middleAZ,  bool& leftWZ, bool& rightWZ, bool& middleWZ) {  // Set alert zone flags based on distance thresholds  leftAZ = leftDist >= 10 && leftDist < 40;  rightAZ = rightDist >= 10 && rightDist < 40;  middleAZ = middleDist >= 10 && middleDist < 40;   // Set warning zone flags based on distance thresholds  leftWZ = leftDist >= 40 && leftDist <= 80;  rightWZ = rightDist >= 40 && rightDist <= 80;  middleWZ = middleDist >= 40 && middleDist <= 80;   // Adjust blink intervals based on distances  if (middleDist < leftDist || middleDist < rightDist) {  leftBlinkInterval = map(middleDist, 10, 80, 100, 200);  rightBlinkInterval = map(middleDist, 10, 80, 100, 200);  } else {  leftBlinkInterval = map(leftDist, 10, 80, 100, 200);  rightBlinkInterval = map(rightDist, 10, 80, 100, 200);  } }  // Function to set RGB color for the left LED void setColorLeft(int R, int G, int B) {  analogWrite(rL, R);  analogWrite(gL, G);  analogWrite(bL, B); }  // Function to set RGB color for the right LED void setColorRight(int R, int G, int B) {  analogWrite(rR, R);  analogWrite(gR, G);  analogWrite(bR, B); }  // Function to trigger LED warning based on zones void ledWarning(bool leftAZ, bool rightAZ, bool middleAZ, bool leftWZ, bool rightWZ, bool middleWZ) {  // Check and update left LED based on zones and blink intervals  if (millis() - lastBlinkLeft >= leftBlinkInterval) {  if ((middleAZ || leftAZ) && leftLED) {  setColorLeft(255, 0, 0); // Red for close objects  leftLED = false;  } else if ((middleWZ || leftWZ) && leftLED) {  setColorLeft(255, 255, 0); // Yellow for intermediate range  leftLED = false;  } else {  setColorLeft(0, 0, 0); // Turn off LEDs  leftLED = true;  }  lastBlinkLeft = millis();  }  // Check and update right LED based on zones and blink intervals  if (millis() - lastBlinkRight >= rightBlinkInterval) {  if ((middleAZ || rightAZ) && rightLED) {  setColorRight(255, 0, 0); // Red for close objects  rightLED = false;  } else if ((middleWZ || rightWZ) && rightLED) {  setColorRight(255, 255, 0); // Yellow for intermediate range  rightLED = false;  } else {  setColorRight(0, 0, 0); // Turn off LEDs  rightLED = true;  }  lastBlinkRight = millis();  } }  // Function to print distances to Serial Monitor void printDistances() {  Serial.print("Left Distance: ");  Serial.print(leftDist);  Serial.print(" cm, Right Distance: ");  Serial.print(rightDist);  Serial.print(" cm Middle Distance: ");  Serial.print(middleDist);  Serial.println(" cm"); }  void setup() {  Serial.begin(115200); // Initialize serial port  pinMode(rL, OUTPUT); // Set LED pins as outputs  pinMode(gL, OUTPUT);  pinMode(bL, OUTPUT);  pinMode(rR, OUTPUT);  pinMode(gR, OUTPUT);  pinMode(bR, OUTPUT); }  void loop() {  getDistance(); // Get LIDAR data  // printDistances(); // Print distances to Serial Monitor (commented out for this example)  calculateWarning(leftDist, rightDist, middleDist, leftAZ, rightAZ, middleAZ, leftWZ, rightWZ, middleWZ); // Calculate warnings  ledWarning(leftAZ, rightAZ, middleAZ, leftWZ, rightWZ, middleWZ); // Call LED warning function } |
| --- |