Intro. to Real Time Embedded Systems

Final Project Report

Motorcycle Adaptive Safety System

Group Number : #15

Student Name	Student Number
Yulia Isaeva	100743213
David Adegoke	100778847
Nicholas Maliavine	100754668
Rudra Vala	100763773

Table of Contents

Introduction

More than half of all motorcycle-related accidents are caused by other vehicles on the road. This project aims to design a safety system for motorcycles which would be able to detect if the motorcycle is dangerously close to a vehicle around it by warning the motorcycle rider. Apart from that, if there is a vehicle in front of the motorcycle, if the vehicle suddenly brakes, the motorcycle safety system is expected to react appropriately by slowing down, and even stop the motorcycle to avoid collision. Another goal of this project would be to scale this system to work with scaled to mopeds and E-bikes as well. This safety system should be similar to blind spot detection, adaptive cruise control, and auto-braking, which are becoming common in modern vehicles.

This project aims to develop a monitoring system for motorcycles that incorporate vehicle proximity danger zone identification and warnings, along with adaptive speed control for the motorcycle. The adaptive system being proposed entails the use of a real-time system that uses motor speed control or braking in order to mitigate collisions, in case a vehicle suddenly merges into the motorcycles lane, or brakes suddenly while traveling in front of the motorcycle. Alert mechanisms for car drivers are also considered, which helps to foster a more secure and safe road for both vehicles and motorcycle drivers. In order for the system to function properly, a multicore processor with fast response times is implemented in the system. One of the major requirements for the system to function properly is to have the system perform its tasks in real time and within hard real times (between 50 and 300 ms). This is because the adaptive system needs to react in milliseconds in order to simulate certain scenarios that befit the use of the adaptive system in order to improve safety and efficiency.

Problem Statement

Motorcycle safety stands as one of the primary concerns in daily transportation due to the vulnerabilities associated with operating a motorcycle. Unlike cars, motorcycles lack the safety offered from a car chassis and other enhanced safety systems , making riders more exposed to potential hazards. Statistics show that motorcycle operators have a 30 times higher mortality rate that increases steadily each year, emphasizing the urgent need for an innovative safety solution that can be implemented into the motorcycle industry.

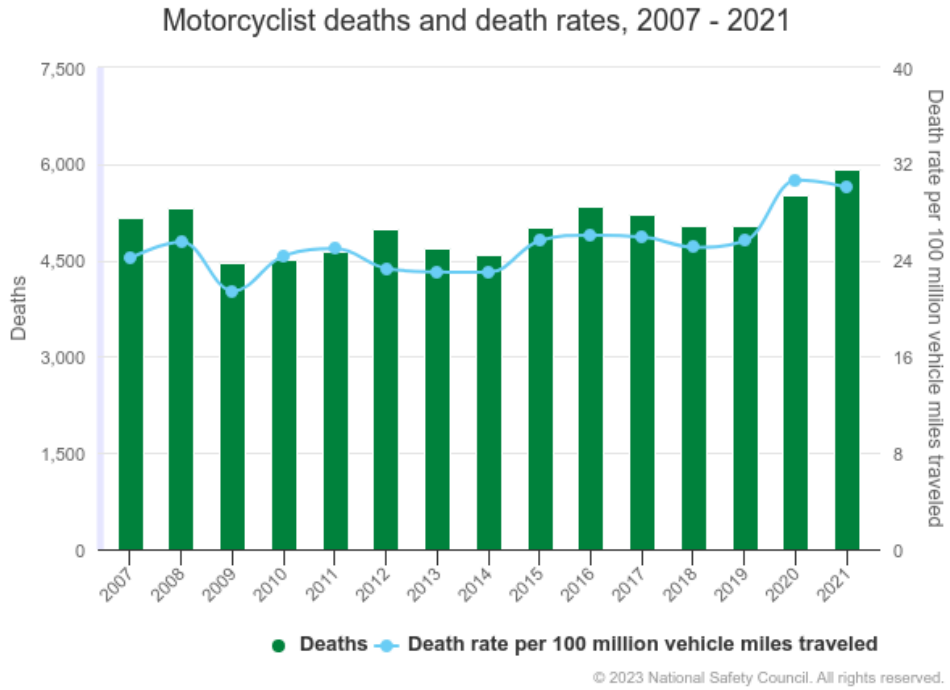


Figure 1: Statistics for motorcycle accidents

While existing safety systems have significantly evolved in the automotive industry, motorcycles have not witnessed the same advancement. Common safety mechanisms such as airbags and stability control are primarily designed for cars and do not address the specific dynamics of motorcycles. The absence of features like blind spot detection and adaptive cruise control leaves motorcycle riders at a higher risk of being involved in an accident.

Some safety features that are already integrated in motorcycles include, Anti-Lock Braking System (ABS), Traction control, Rear Lift-off Protection , Blind Spot Detection and Lane Assist, and Automatic Emergency Call System. The blind detection and lane assist feature is still new and underdeveloped. BMW introduced a Side View Assist (SVA) to alert bikers when there are other motorists surrounding them within a 5 meter or less radius from the rider. However, this system only indicates other close vehicles by flashing a warning light at the mirror foot and does not provide speed response features. As shown, most of these systems do not prevent lane merging accidents , thus , developing an adaptive safety system would be greatly beneficial.

Figure 2: BMW SVA System Range

1.0 - Project Architecture

Electronic Component Identification & Layout

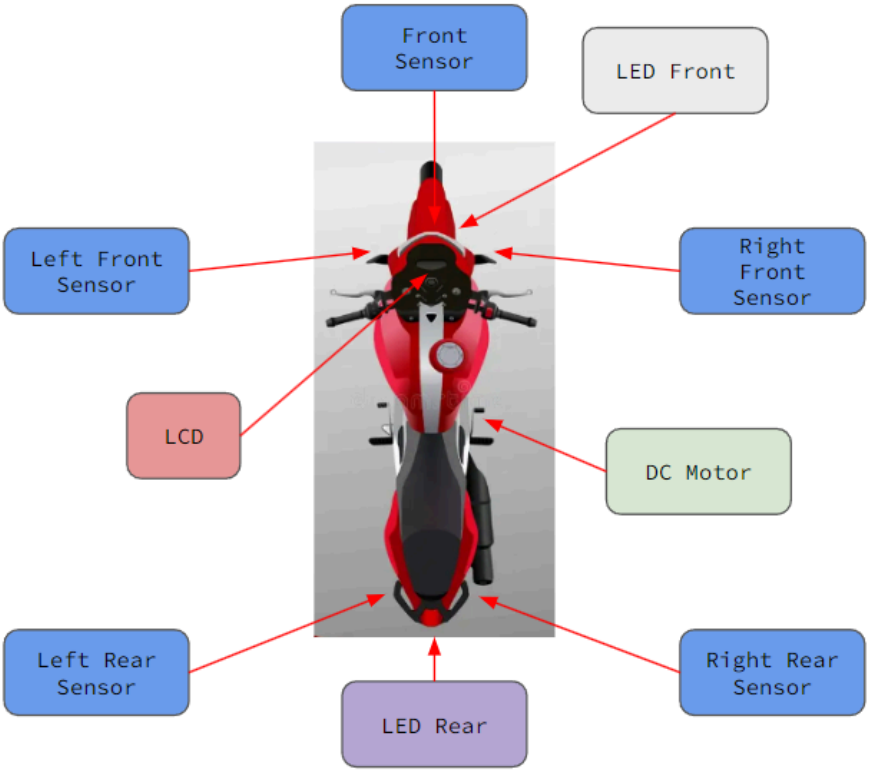


Fig 1.0: Layout of components such as sensors, LED's, LCD and DC motor.

To achieve the set tasks of a motorcycle adaptive safety system, it was necessary to identify some of the available electronic components. These components primarily consisted of sensors to detect vehicles surrounding the motorcycle, and means of visual communication for the motorcycle driver to warn them of any potential dangers. Additionally, for demo purposes, motors would be required to showcase the adaptive speed control, and emergency stopping system of the motorcycle.

Once these components were identified, a general design layout of the identified components and their positions on a motorcycle body was created. This layout can be seen in Fig 1.0.

Motorcycle Small Scale Physical Prototype

Once the motorcycle electronics component layout was finalized, a motorcycle structure was required to mount these components onto. At first a real motorcycle was considered for the project, however due to harsh weather conditions in December, this idea was not considered further. Instead, a small scale prototype was developed. This prototype is developed using CAD software, and its fabrication was done using additive manufacturing technology. Fig 1.1 shows the developed CAD model.

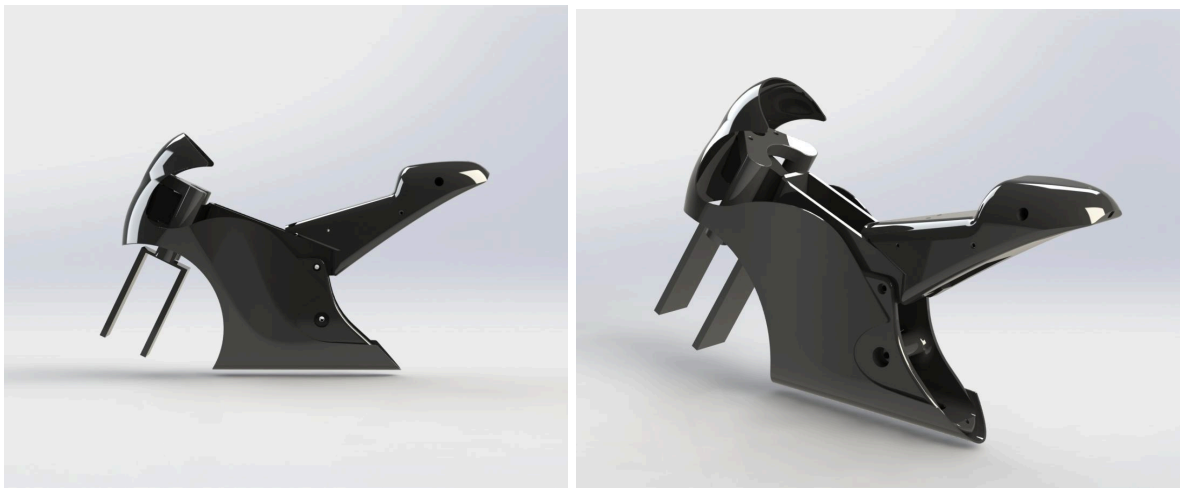


Figure 1.1: CAD model used to develop the prototype using 3D printing technology.

The motorcycle prototype was modeled using SolidWorks. The 3D model was designed to incorporate all the electrical components easily within the body of the motorcycle. Additive manufacturing was used to build a physical model. PLA filament was used due to its strength and durability.

System General Architecture

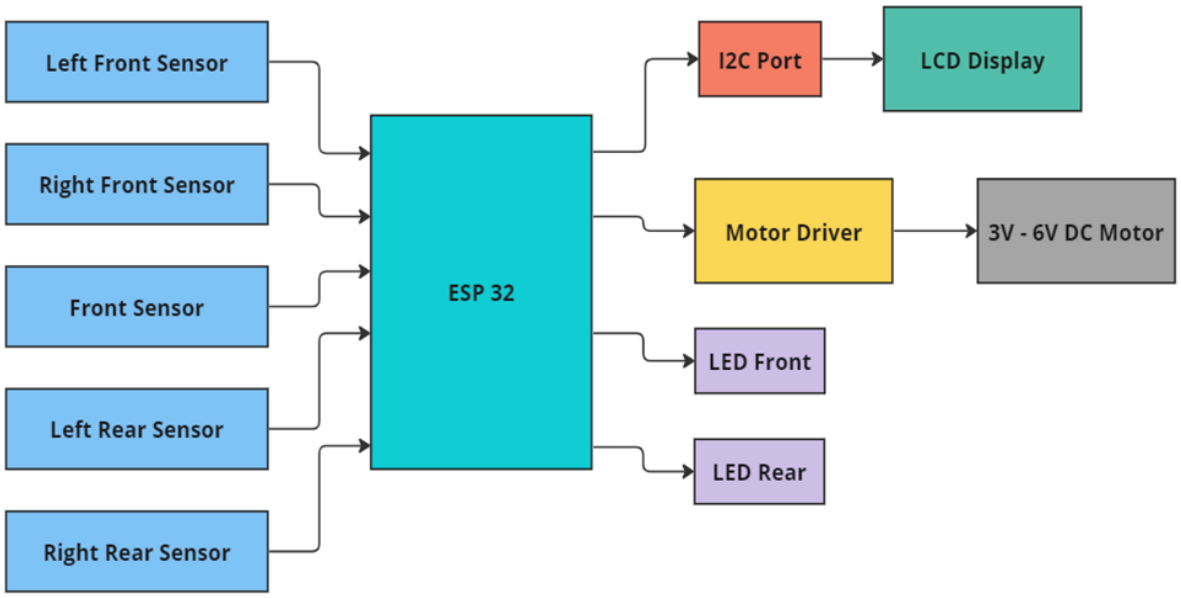


Figure 1.2: Electrical / Electronics Architecture

The high level system architecture for the motorcycle safety system is shown in Figure 3. An ESP-32 is used as a dual core processing microcontroller to handle two tasks simultaneously. Task 1 is to control the motorcycle speed when a vehicle closes into the motorcycle from the front, activating the front sensor. Task 2 is to alert the motorcycle driver via the LCD, and surrounding vehicle via pulsing LED, when one of the Left Front, Right Front, Left Rear, and Right Rear sensors detect a vehicle in the danger spot of the motorcycle. Another reason to use an ESP-32 was to take the opportunity to learn and use a different microcontroller. The components to be used for the general architecture used in Fig 1.2 is further explored under the Hardware & Software Design section.

2.0 - Hardware & Software Design

Hardware Design

As per the general architecture of the project shown in Fig 1.2, identifying some of the hardware components was necessary to perform the tasks required. Various components were researched and evaluated to identify correct components. These components are described and distinguished in Table 2.0. Some components were established from the beginning of the project such as the LED's, LCD, and ESP-32.

Task	Identified Hardware	Discussion
Detects vehicles in the front of the motorcycle for motorcycle speed control.	Ultrasonic Sensor HC-SR04.	Ultrasonic sensors were readily available in the market. These sensors are capable of providing numeric distance values for an obstacle which can be used in the software for programming.
Detects Vehicles in the Left Front, Right Front, Rear Left and Rear Right corners of the motorcycle.	IR Sensor.	IR sensors are readily available in the market. These sensors were considered over ultrasonic sensors for the danger zone monitoring on the corners of the motorcycle as additional ultrasonic sensors can add weight, bulk and wiring on the prototype. Additionally ultrasonic sensors used for danger zone monitoring can cause interference with the front ultrasonic sensor, which can cause issues with the critical function of speed control.
System Control. Task management.	ESP-32	ESP-32 was considered over an arduino due to the presence of two cores, preemptive priority based parallel tasking. This also helps the system to keep its response time under the hard real time systems deadline of 300 ms for automotive systems. Furthermore, ESP-32 provides PWM, I2C, Serial

		communication support.
Display Warning	I2C based LCD	An LCD seemed to be a suitable hardware to be used as the motorcycle dashboard to warn the motorcycle rider about any vehicles in its danger zones.
Motor Speed Control	L298N Motor Driver	L298N motor driver is easily available in the market. Its small size makes it ideal for small scale prototypes. PWM based motor speed control and motor direction control is also supported by this motor driver. Additionally, it consists of a +5v supply port, which can be used to power the 5v hardware such as the IR sensors, ultrasonic sensors, and the ESP-32.
Motors to drive the prototype.	3v to 6v geared DC Motor with wheel.	Due to the prototype being lightweight, the required motor did not need high torque figures. A max 6v motor was deemed suitable. A motor which comes with a pre-existing gearbox, shaft and a wheel was considered to keep the design simple and eliminate any additional requirement of mechanical design and fabrications such as motor shaft and wheel coupling.
Alerting the Surrounding vehicle of the motorcycles presence.	LED's	A technique of pulsing LEDs was considered everytime a vehicle was found in the motorcycles danger zones. Ideally a system like V2V (Vehicle to Vehicle) should be considered to alert the vehicle driver of the motorcycle. However, to keep the project complexity low, LED's in position of the headlight and taillight were considered to demonstrate the feature.
Power Supply	Two 3.7 volt batteries @3000mAh	Two 3.7v 18650 batteries were deemed suitable with a total voltage

		of 7.4 volts. This supply is suitable for the L298N motor driver, which acts like a power unit for the entire system using its 5 volt power terminal. The batteries are rechargeable as multiple test runs can deplete the power of the batteries. Further battery requirement calculations are shown in Table 2.1 and below.

Table 2.0: Project hardware evaluation

A general battery requirement calculation was performed to identify the correct batteries for the above architecture in Fig 1.2. Max current consumption values were derived for each component from the components datasheet. These values can be seen in Table 1.0.

#	Part Name	Max Rated Current Consumption	Qty
1	ESP-32	250 mA (with all GPIO used)	1
2	L298N Motor Driver	36 mA (max operating current draw)	1
3	6v DC Motor	1000 mA (Stall Current)	1
4	IR Sensor	Approx 1 mA (max)	4
5	Ultrasonic Sensor	100 mA (max)	1
6	LED	20 mA (max)	3

Table 2.1: Max current draw for each component

Total Max Current Draw (mA) = 250 + 36 + 1000 + 1(4) + 100 + 20(3) = 1450 mA

Batteries considered: 3000 mAh
Time the battery would last at max current draw: 3000 / 1450 = 2.06 = 2 hours approx.

The above calculation shows that the battery considered for the prototype is adequate.

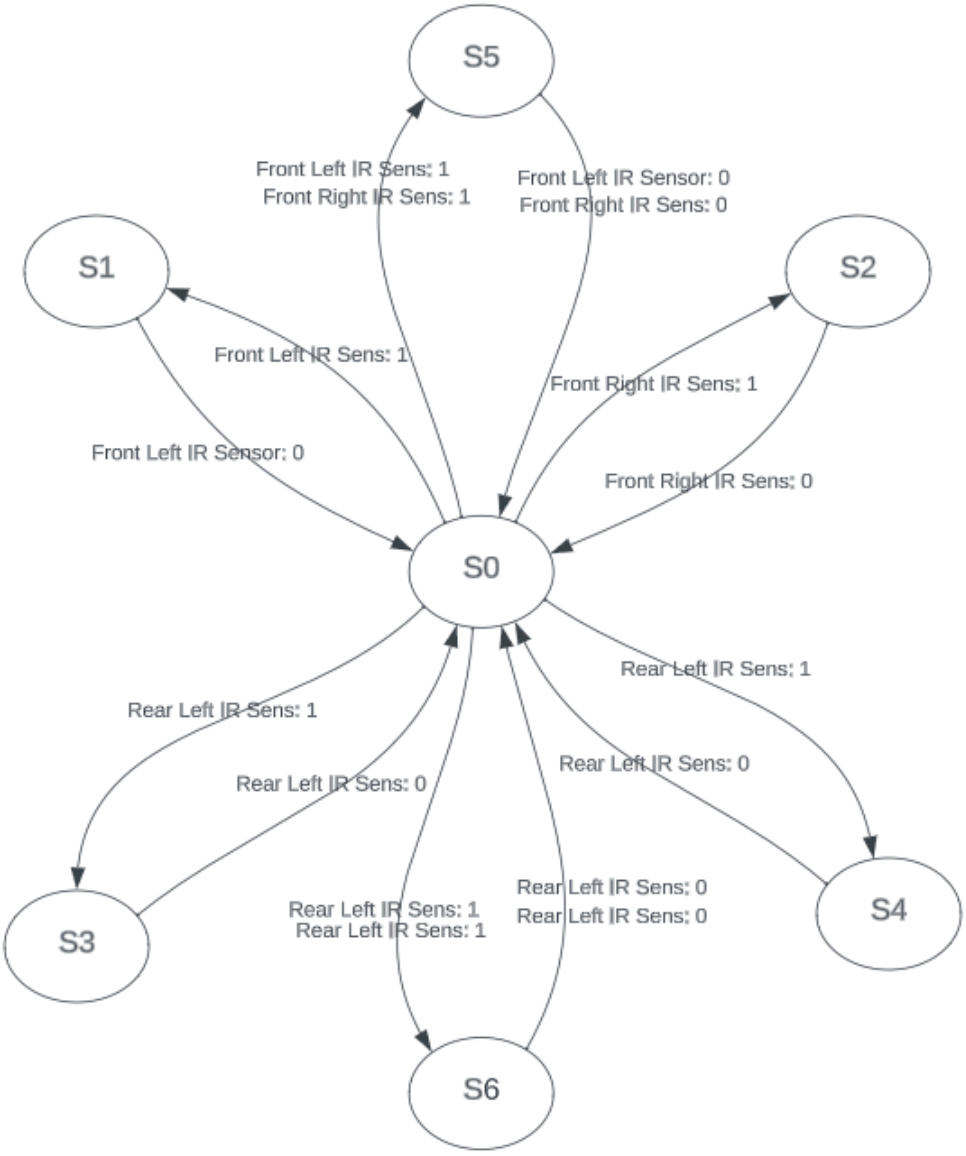
Software Design

For the software design, state machine diagrams were developed to thoroughly understand how the tasks would be performed. Since ESP-32 enabled the use of two processors, the tasks were divided for each processor. Table 2.2 shows the task distribution for each processor. Task 2 was considered as a high priority task compared to task 1 since it is more critical. However, the two processors perform each task in parallel in their respective dedicated processors, a priority declaration is not required since they are single tasks in each cores to be performed. Although, to demonstrate the knowledge of task priority in multi-task scheduling systems, the task priorities were considered.

Task	**ESP-32 Processor**	**Priority**
Task 1: Detecting vehicles in the danger zone / blind spot of the motorcycle	Processor: 0	Low
Task 2: Detecting vehicle in the front of the motorcycle to control motorcycle speed / emergency stopping.	Processor: 1	High

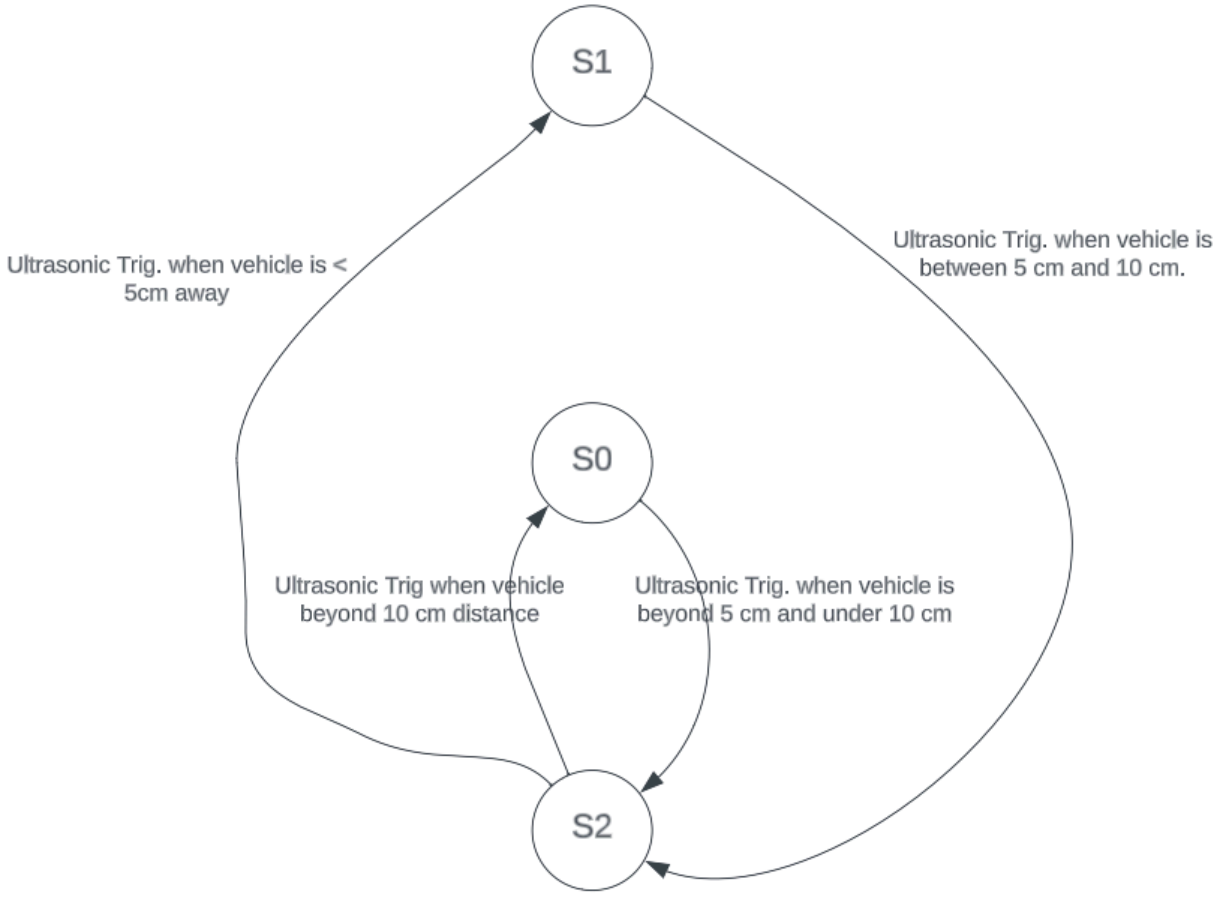
Table 2.2: Task scheduling and allocation for both processors of ESP-32

Task 1: State Diagram



S0: State of the LCD displaying no warnings. No LED modulation.
S1: State of the LCD displaying Left Front Danger warning.
S2: State of the LCD displaying Right Front Danger warning.
S3: State of the LCD displaying Rear Left Danger warning.
S4: State of the LCD displaying Rear Right Danger warning.
S5: Front LED Modulation
S5: Rear LED Modulation

Task 2: State Diagram



S0: State of the motor speed running at the motorcycles users defined speed.
S1: State of the motor speed gradually stopping for emergency stop.
S2: State of the motor speed running at a slower speed.

The state diagrams helped visualize the general logic of the program that would be implemented in the ESP-32 microcontroller. The ESP-32 was programmed according to the state machine diagram, and the pseudo code can be seen below:

```
// Include necessary libraries
   (FreeRTOS.h, FreeRTOS_Task, Wire.h, LiquidCrystal_I2C )
// Define constants for pins and sensor thresholds

Setup:

  Initialize LCD, LEDs, motor pins, and sensor pins
  Create tasks for IR sensor handling and motor control
  Set sensor pins as inputs

  // Create a task to handle LCD printing in processor 0
     xTaskCreatePinnedToCore(irSensTask, "IR Sensor Task", 4096, NULL, 1, &irTask, 0);

 // Create a task to control the motor speed in processor 1
    xTaskCreatePinnedToCore(motorControlTask, "Motor Control Task", 4096, NULL, 5,
    &motorTask, 1);


Loop:
  // Tasks handle specific functionalities, so the loop remains empty

IR Sensor Task:
  Loop:
    Read IR sensor inputs for front and rear sensors
    Display sensor triggered messages on the LCD
    Flicker headlight and taillight LEDs based on sensor triggers
    Display time taken for sensor processing on the LCD
    Delay for LCD responsiveness

Motor Control Task:
  Loop:
    Read forward ultrasonic sensor distance
    Adjust motor speed based on the distance
    Set motor speed using PWM
    Toggle the hard reaction timer LED based on motor processing time
    Delay for motor control

Front Ultrasonic Sensor Function:
  Trigger ultrasonic sensor and calculate distance
  Return the calculated distance
```

In the pseudocode shown above, the line - xTaskCreatePinnedToCore(irSensTask, "IR Sensor Task", 4096, NULL, 1, &irTask, 0); specifically creates a task using the FreeRTOS libraries for using the multitasking and task scheduling features of a real time operating system.

For the mentioned line of the program, a function pointer “irSensTask” is declared which is responsible for performing the task. Followed by that is the task name for human understanding which is a string “IR Sensor Task”. The 4096 is the 4kB word length for ESP-32 which is utilized at its max. NULL is the pointer to the data that could be passed to the function, however, no data was passed in this case. The number 1 is the priority rating where 0 is the lowest priority and 24 is the highest priority for that task. The term “&irTask” is the address of the variable where the task handle is stored. This handle allows the program to refer to or control the task after its creation. Lastly, the numeric value at the end represents the processor number. Since the IR sensor task is being handled by the first core of the ESP-32, the processor ID is 0. This part of the program was the main feature that helped implement features of the real time operating system. The remaining program utilizes if / else statements to use values from the IR sensor and the ultrasonic sensors to control the LCD message, LED modulation and Motor speed control.
Note: The full program is shown in the appendix section.

3.0 - Project Testing & Results

Several tests were performed during various phases of the programming and hardware troubleshooting. At first, the hardware was evaluated by connecting various components with the ESP-32 as per the electrical schematic shown in the appendix section. Simple programs were tried using a non real time operating system environment. Once the hardware was validated, the libraries to implement FreeRTOS were added to implement and use features of the real time operating system. The main testing required for the software portion was to run both task 1, and task 2 in parallel on both processors of EPS-32. These tasks were programmed for a preemptive approach of task handling; both tasks had a dedicated processor for them, and no interrupt routines were used. Such an approach was identified after trial and error of a couple of task scheduling configurations. Table 3.0 shows some of the tests performed to achieve the result of both tasks under the boundaries of 20 ms to 300 ms, which was established at the beginning of the project as a hard deadline for an automotive system. To achieve timings of the processes, the FreeRTOS function “vTaskDelay(pdMS_TO_TICKS());” was used along with millis();.

Task Scheduling Configuration	Result
Task 1: To read the IR sensors and display warning on the LCD (on Processor 0, low priority task). Task 2: To read the ultrasonic sensors and send values to Task 3 for motor speed control (on Processor 0, high priority task). Task 3: Motor Speed controlling based on values obtained from Task 2 (on Processor 1).	ESP-32 programmed with the given task scheduling configuration resulted in system response beyond the boundary of 300ms for Task 1. The system with this configuration responded in approximately 400 ms for Task 1. This might have happened due to a preemptive task management approach, where Task 2 occupies the memory of processor 0, adding a delay for task 1. The preemptive methodology can cause the high priority task to occupy the processor causing a delay for the low priority task adding to the time taken for the response.
Task 1: To read IR sensors and display warning on the LCD (on Processor 0) Task 2: To read the ultrasonic sensors values and control the motor speed (on Processor 1)	With this configuration, both task 1 and task 2 were achieved under 300 ms. The task 1 processing time was displayed on the LCD. Similarly, task 2 processing times were attempted to be displayed on the LCD, however due to LCD clearing issues in a short amount of time, achieving this was difficult. Instead, another LED was used which stays on as long as task 2 is achieved within 100 ms.

Table 3.0: Task scheduling configuration tests and results.

Conclusions

This project addresses a very critical issue in motorcycle safety, with the goal of reducing accidents caused by proximity to other vehicles. The proposed adaptive safety system integrates a RTOS (real-time operating system), an ESP-32 microcontroller, four IR sensors and one ultrasonic sensor used for danger zone detection and adaptive speed control. The small scale prototype was modeled using SolidWorks and fabricated with the help of additive manufacturing, providing a well constructed body for the integration of electronic components. As written in the earlier parts of the report, the system's architecture shows that all tasks handled by the microprocessor emphasize the importance of real-time response between the specified hard deadlines (between 50 ms and 300 ms). With the integration of adaptive features like danger zone detection and adaptive speed control, the project addresses the existing gap in the current motorcycle safety systems. The adaptive system helps to lay a groundwork for further future testing, modification and refinement in larger-scale motorcycles and motor vehicles. In conclusion, this project contributes to the overall advancement of motorcycle safety systems.

Group members' contribution

Group Member	Contribution
Yulia Isaeva	Prototype Design, CAD, 3D Printing, Report, Presentation
David Adegoke	Report, Presentation, Project Management
Nicholas Maliavine	Hardware Benchmarking, Coding, Report, Presentation
Rudra Vala	Coding, Report, Presentation, Circuit Design and Assembly of prototype

References

[1] R. Miller. "5 Motorcycle Safety Features You Need to Know." cycletrader. https://www.cycletrader.com/blog/2023/05/08/5-motorcycle-safety-features-you-need-to-know/ [accessed Dec 3, 2023]

[2] "SIDE VIEW ASSIST (SVA)." bmw-motorrad. https://www.bmw-motorrad.ca/en/Innovation/detail/comfort-ergonomics/sva.html [accessed Dec 3, 2023]

[3] "Motorcycles." injuryfacts. https://injuryfacts.nsc.org/motor-vehicle/road-users/motorcycles/ [accessed Dec 3, 2023]

Appendix

Main Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>

// Set the LCD address and dimensions according to your specific module
#define I2C_ADDR 0x27 // Change this to your LCD I2C address
#define SDA_PIN 21    // Define the SDA pin
#define SCL_PIN 22    // Define the SCL pin
#define LCD_WIDTH 16  // Change this according to your LCD columns
#define LCD_HEIGHT 2  // Change this according to your LCD rows

// Create an LCD object
LiquidCrystal_I2C lcd(I2C_ADDR, LCD_WIDTH, LCD_HEIGHT);

// IR sensor pins
const int irSensor_Front_Left = 5;   // Pin for front left IR sensor
const int irSensor_Front_Right = 18; // Pin for front right IR sensor
const int irSensor_Rear_Left = 19;   // Pin for rear left IR sensor
const int irSensor_Rear_Right = 23;  // Pin for rear right IR sensor

//Forward Ultrasonic Sensor
const int front_trig = 32;
const int front_echo = 33;


// Define distance threshold to trigger motor activation (in centimeters)
const int distanceThreshold = 10; // Modify this threshold as needed

// Headlight LED pin
const int headlight_led = 2;

// Taillight LED pin
const int taillight_led = 4;

// Motor Hard Reaction Time Blinker
const int hard_timer_led = 12;

// Motor control pins
int enA = 13; // Motor control pin 1
int in1 = 25;
```

```
int in2 = 26;

//Set Motor_Speed (255 max speed)
int motor_speed = 55;

// Task handles
TaskHandle_t irTask;
TaskHandle_t motorTask;

// Function prototypes for tasks
void irSensTask(void *pvParameters);
void motorControlTask(void *pvParameters);



void setup() {
  // Start the Wire library with defined SDA and SCL pins
  Wire.begin(SDA_PIN, SCL_PIN);

  // Initialize the LCD with the address and dimensions
  lcd.init();

  // Turn on the backlight
  lcd.backlight();

  // Set headlight LED pin as output
  pinMode(headlight_led, OUTPUT);

  // Set taillight LED pin as output
  pinMode(taillight_led, OUTPUT);

  // Motor Hard Reaction Timer LED pin as output
  pinMode(hard_timer_led, OUTPUT);

  // Set motor control pins as outputs
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);


  // Set ultrasonic sensor pins as inputs/outputs
  pinMode(front_trig, OUTPUT);
  pinMode(front_echo, INPUT);

  // Create a task to handle LCD printing in processor 0
  xTaskCreatePinnedToCore(irSensTask, "IR Sensor Task", 4096, NULL, 1, &irTask, 0);
```

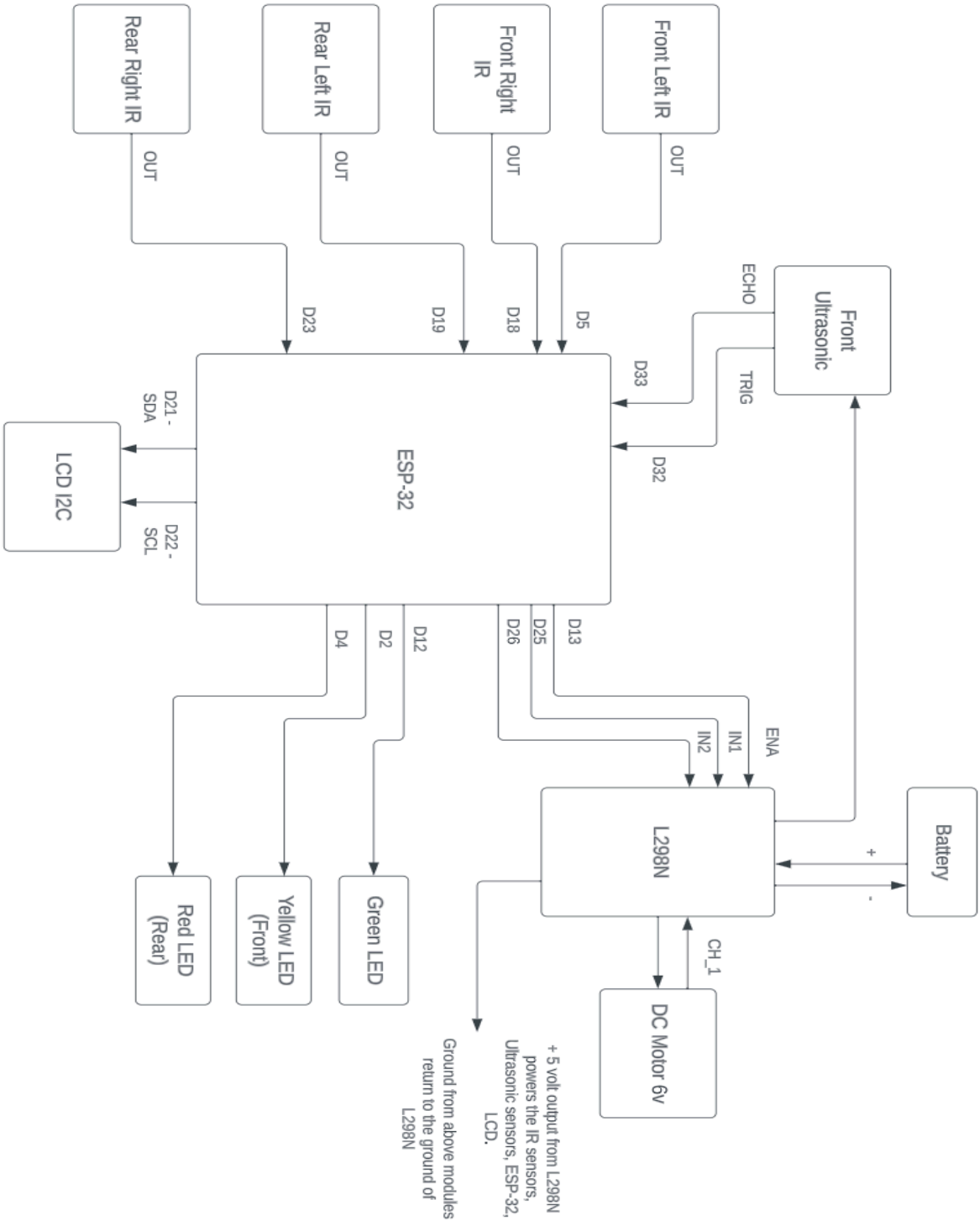
```
  // Create a task to control the motor speed in processor 1
  xTaskCreatePinnedToCore(motorControlTask, "Motor Control Task", 4096, NULL, 5,
&motorTask, 1);

  // Set IR sensor pins as inputs
  pinMode(irSensor_Front_Left, INPUT);
  pinMode(irSensor_Front_Right, INPUT);
  pinMode(irSensor_Rear_Left, INPUT);
  pinMode(irSensor_Rear_Right, INPUT);
}



void loop() {
  // Nothing in the loop as the tasks handle their respective functionalities
}




// Task to handle LCD printing based on IR sensor activation in processor 0
void irSensTask(void *pvParameters) {
  unsigned long startTime = 0;
  unsigned long endTime = 0;
  unsigned long IR_time = 0;
  for (;;) {

    startTime = millis(); // Start time before processing

    bool frontLeftTriggered = digitalRead(irSensor_Front_Left) == LOW;
    bool frontRightTriggered = digitalRead(irSensor_Front_Right) == LOW;
    bool rearLeftTriggered = digitalRead(irSensor_Rear_Left) == LOW;
    bool rearRightTriggered = digitalRead(irSensor_Rear_Right) == LOW;

    if (frontLeftTriggered) {
      lcd.setCursor(0, 0);
      lcd.print("Front Left !!!");
    } else if (frontRightTriggered) {
      lcd.setCursor(0, 0);
      lcd.print("Front Right !!!");
    } else if (rearRightTriggered) {
      lcd.setCursor(0, 0);
      lcd.print("Rear Right !!!");
    } else if (rearLeftTriggered) {
      lcd.setCursor(0, 0);
```

```
    lcd.print("Rear Left !!!");
   } else {
    lcd.clear();
   }

   // Flicker headlight LED if both front sensors are triggered
   if (frontLeftTriggered || frontRightTriggered) {
    digitalWrite(headlight_led, HIGH);
    delay(random(20, 100)); // Flickering duration
    digitalWrite(headlight_led, LOW);
    delay(random(50, 200)); // Time between flickers
   } else {
    digitalWrite(headlight_led, LOW); // Turn off LED if sensors are not triggered
   }

   // Flicker headlight LED if both front sensors are triggered
   if (rearLeftTriggered || rearRightTriggered) {
    digitalWrite(taillight_led, HIGH);
    delay(random(20, 100)); // Flickering duration
    digitalWrite(taillight_led, LOW);
    delay(random(50, 200)); // Time between flickers
   } else {
    digitalWrite(taillight_led, LOW); // Turn off LED if sensors are not triggered
   }

   endTime = millis(); // End time after processing
   IR_time = endTime - startTime;
   lcd.setCursor(0, 1);
   lcd.print("Time in ms: ");
   lcd.print(IR_time); // Print time taken on LCD

   vTaskDelay(pdMS_TO_TICKS(200)); // Adjust the delay as needed for responsiveness
  }
}




// Task to control motor speed using PWM on processor 1
void motorControlTask(void *pvParameters) {

  unsigned long startTime_2 = 0;
  unsigned long endTime_2 = 0;
  unsigned long motor_time = 0;
```

```
  for (;;) {
    startTime_2 = millis(); // Start time before processing
    long front = forward_ultrasonic_sens();

    // Check if the distance is less than the threshold to activate the motor
    if (front < distanceThreshold) {
      motor_speed = 0; // Set motor speed here or modify as needed
    }

    else if (front > distanceThreshold && front <= 20) {
      motor_speed = 50;
    }

    else {
      motor_speed = 70; // Stop the motor if the distance is above the threshold
    }

    // Set motor speed using PWM
    analogWrite(enA, motor_speed);
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);

    endTime_2 = millis(); // End time after processing
    motor_time = endTime_2 - startTime_2;

    if (motor_time < 100) {
      digitalWrite(hard_timer_led, HIGH);
    }

    else{
      digitalWrite(hard_timer_led, LOW);
    }

//    lcd.clear();
//    lcd.setCursor(0, 16);
//    lcd.print(motor_time); // Print time taken on LCD

    vTaskDelay(pdMS_TO_TICKS(200)); // Adjust the delay as needed for motor control
  }
}



// Front Ultrasonic Sensor
```

```
long forward_ultrasonic_sens()
{
  long duration;
  digitalWrite(front_trig, LOW);
  delayMicroseconds(5);
  digitalWrite(front_trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(front_trig, LOW);

  //Read Ultrasonic sens and calculate distance
  duration = pulseIn(front_echo, HIGH);
  return (duration / 50);
}
```

Electrical Schematic



Prototype Images