Internship Project Report

Automated A.C. Vent Controlling System

Submitted by

Harsh Kapoor
Electronics and Communication Engineering
International Institute of Information Technology - Hyderabad

Under the guidance of

Kartar Singh
Deputy General Manager
Tata Motors, Engineering Research Centre

Varnit Vats
General Manager
Tata Motors, Engineering Research Centre

TATA MOTORS
ENGINEERING RESEARCH CENTRE

Summer Internship 2025

Contents

Chapter 1

Objective

The primary objective of this project was to automate the operation of air-conditioning (A.C.) vents in electric commercial vehicles, thereby enhancing energy efficiency and passenger comfort. The task involved designing a smart control system that could dynamically manage the A.C. vents based on passenger presence. To achieve this, the following goals were set:

- **Passenger Detection Pipeline:** Develop a reliable pipeline to detect and localize passengers in the vehicle cabin using appropriate sensing and vision-based techniques.
- **Vent Control Mechanism:** Design and implement a mechanism capable of actuating A.C. vents—opening or closing them—based on control commands.
- **System Integration:** Seamlessly integrate the passenger detection system with the A.C. vent control mechanism to allow real-time, responsive operation.
- **Proof of Concept (PoC):** Build a functional proof of concept simulating various components of an electric commercial vehicle to demonstrate the end-to-end working of the proposed solution.

Chapter 2

Introduction

2.1 Vehicle Studies

As part of the internship, I was tasked to study the various electronic sensors and control units used in commercial vehicles. This involved gaining a detailed understanding of the components that contribute to vehicle automation, safety, and control. The Vehicles I studied are -

2.1.1 CNG - 1620 Buro - 6

This was a CNG vehicle equipped with six CNG tanks. The vehicle had dimensions of $a \times b$ where a is the floor height in 100s of mm and b is the bus length from front to back.

The following image is a diagrammatic representation of the vehicle layout. It indicates the placement of the various sensors and electronic controllers throughout the vehicle.

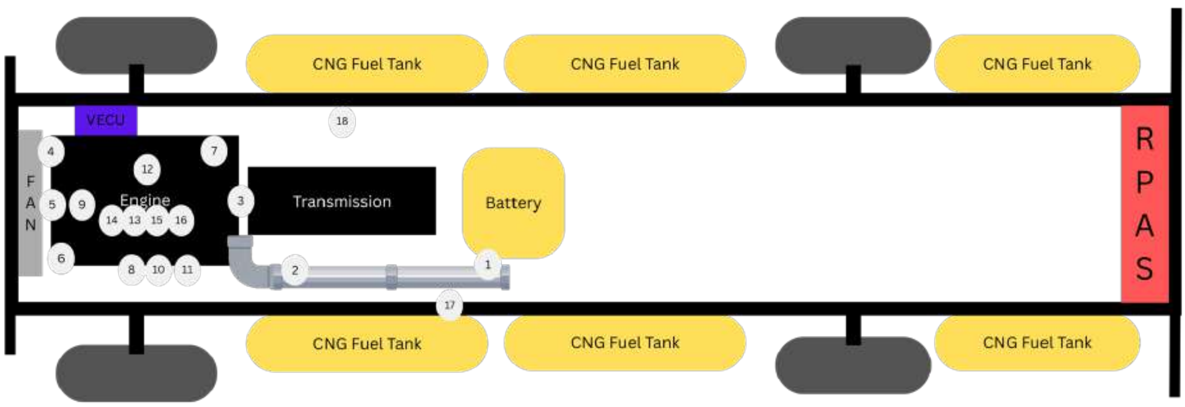

Figure 2.1: Diagrammatic Representation of 1620 CNG Vehicle

The Description of the components (along with the names of the numbered components in the image) are as follows -

S.No	Sensor Name	Sensor Purpose
1	G.P.F.	GPF (Gasoline Particulate Filter) sensor is a component of the exhaust system in gasoline engines designed to capture particulate matter. It works by measuring pressure differences across the GPF, which indicates the amount of soot buildup within the filter. This information is then used by the engine control unit (ECU) to trigger regeneration cycles, which burn off the trapped soot and maintain optimal filter performance
2	Post Oxygen Sensor	Detects the concentration of oxygen molecules in the exhaust air, this combined with the Pre Oxygen measurement tells the ECU the amount of Oxygen being used in engine for fuel combustion, and whether it needs more air to be sucked in or not.
3	Crank Sensor	Monitors the position or rotational speed of the crankshaft (converts linear motion of pistons to rotational motion of wheels)
4	E Viscous Fan	Utilizes electronic control to optimize fan speed and operation for cooling engine. It is controlled by the ECU, and this is based on Engine oil temperature and the coolant temperature. Its speed can be varied instead of being in a binary state (on/off based on bimetallic strip).
5	CAM	Monitors the camshaft (which controls the timing of inlet and exhaust valve operations) position and speed and feeds that data to the vehicle's ECU. The ECU needs this data to control how much fuel enters the combustion chamber and ignition (spark) timing to ignite the fuel.
6	ETB	(Electric Throttle Body) Controls how much air the vehicle's engine receives based on how far the gas pedal is pressed. Controlled by the ECU based on the gas pedal, and the oxygen sensors data.
7	Oil Pressure and Temperature Sensor	Measures the pressure and temperature of the engine oil. This data is fed to the ECU and controls the Fans speed and the amount of coolant intake.
8	Pre Oxygen Sensor	Detects the concentration of oxygen molecules in the intake air and the data is passed on the ECU for the usage in ETB control.
9	Coolant Temperature Sensor	To sense the temperature of the coolant of the engine.
10	Pre Throttle TMAP Sensor	Measures the temperature and pressure of the air before it enters the throttle body (controls air intake) of an engine. Throttle is the valve from which air enters the engine; therefore, the sensor measures the temperature and pressure of the air entering the engine so that the control of intake air can be done.

11	Exhaust Temperature Sensor	To sense the temperature of the exhaust gas of the engine. This helps in optimizing engine performance by changing fuel oxygen fix in engine.
12	Ignition Coils	Transform the low voltage from the battery (here 24V) into a much higher voltage (tens of thousands of volts) needed to create a spark at the spark plug, igniting the fuel-air mixture in the engine.
13	Fuel Injectors	Device for atomizing and injecting fuel into an internal combustion engine
14	TMAP Sensor	Measures the pressure and temperature of the air in an engine. This helps in the efficiency of engine by controlling the combustion by fuel and air by ecu.
15	Fuel Rail Temperature Sensor	Monitors the temperature inside the fuel rail (a metal tube that connects the fuel delivery system to the engine)
16	Gas Rail Pressure Sensor	Monitors the pressure inside the gas (CNG) rail (a metal tube that connects the fuel delivery system to the engine)
17	VECU	Manages and controls various electronic systems using the data from the sensors of the vehicle.
18	HP Cutoff Solenoid Assembly	Used to shut off the flow of High-pressure Fuel at the fuel inlet.
19	RPAS Sensor	Reverse Parking Assistance Systems are automated parking systems that help drivers park with extreme ease and precision.
20	NOX Sensor	Measures the amount of Nitrogen Oxides in the exhaust fumes.

2.1.2 Electric Vehicle - Magna

This was an electric vehicle equipped with four 700kg, 665.6V, 150Ah, 99.84kWh battery packs which are connected in parallel.

The following image is a diagrammatic representation of the vehicle layout. It indicates the placement of the various sensors and electronic controllers throughout the vehicle.

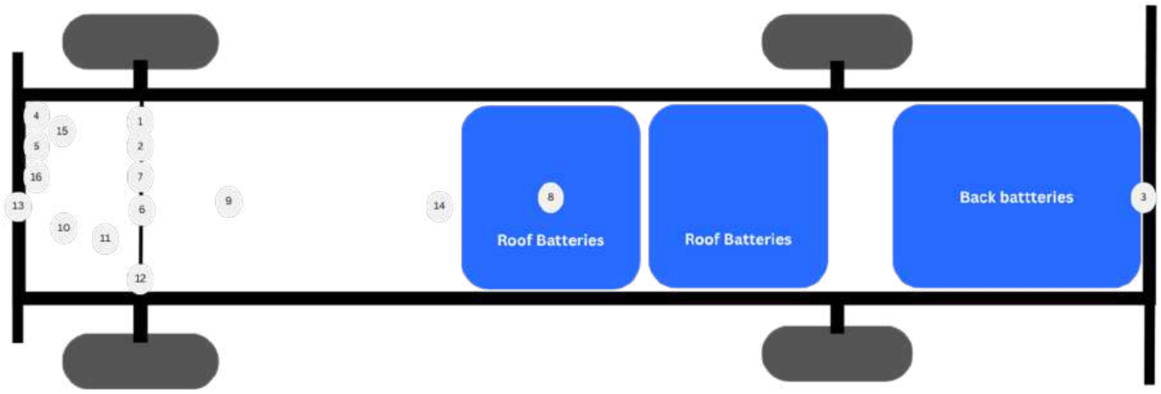

Figure 2.2: Diagrammatic Representation of Magna Vehicle

The Description of the components (along with the names of the numbered components in the image) are as follows -

S.No.	Sensor Name	Sensor Purpose
1	G.D.C.U.	Gateway Domain Control Unit, acts as a hub for data exchange, and connects various parts of the vehicle via signals
2	VECU	Vehicle Electronic Control Unit, used for low voltage controls
3	RPAS	Assists the driver in reversing and parking by using range detection to scan the surroundings. 5 in number – 1 master and 4 slave sensors, all at the back
4	Data Logger	Logs the data of various sensors with time stamps for troubleshooting and sequencing of commands
5	TCU Telematics	Provides telematics data (GPS, IMU, etc.) of the vehicle
6	E.B.S.	Electronic Braking System; helps achieve shorter braking distances using electronic control signals
7	EVCU	Electric Vehicle Control Unit, used in high voltage controls
8	BMS	Battery Monitoring System; manages the health and status of the battery pack
9	MCU	Motor Control Unit; controls speed and gear ratio of the motors
10	DC-DC Converter	Converts DC high voltage to DC low voltage, or vice versa
11	AUX	Converts DC to AC for powering systems like power steering (AUX 1) and the AC unit (AUX 2)
12	TCP	Traction Cooling Pump; used to control motor traction temperature
13	HVAC	AC Control Unit; controls the blower and compressor settings
14	BCS	Battery Control System; controls the charge/discharge rate of the battery, optimizing performance
15	Steering Angle	Detects the angle to which the steering wheel is turned
16	DMS	Driver Monitoring System; monitors the driver's state and helps keep them alert

2.2 A.C. Vent Fitment

I also visited the Tata Motors Body Solutions Limited (TMBSL) plant to inspect the A.C. vent fitment in the vehicle. The objective of the visit was to understand the feasibility of designing a mechanism that could automate the opening and closing of the A.C. vents using actuators. Images of the vents in their open and closed positions are shown below:

Figure 2.3: Vent in Open and Closed Position

I also Calculated the area and the number of AC Vents as a part of the project so that the Compressor and Blower in the vehicle can be adjusted based on the number of closed/open vents so that energy can be saved. The Detailed Calculations are given in the following sections.

2.3 Passenger Detection Techniques

To enable automatic opening and closing of the A.C. vents based on passenger presence, I proposed the following methods of passenger detection, with their respective advantages and limitations-

2.3.1 Pressure Sensor

The idea is to have pressure sensors in seat to detect the weight of a person on the seats. It is similar to the seatbelt warnings in passenger vehicles. A certain threshold of weight will be required for the vents to respond to.

Advantages:

- The main advantage is that it is very easy to interface, as one sensor controls one vent.
- It is already used in Passenger Vehicles so the complexity of ideation is excluded.

Disadvantages:

- Will be needing at-least one per seat, the wiring will increase.
- This method not be able to distinguish between Humans and other objects kept on seat.

2.3.2 Infra-Red/Temperature Sensor

The idea is to have I.R. Sensors pointing at seats to detect human body temperatures. This method will turn on the AC Vents if the human body temperature is detected and will turn it off if it is not detected for a prolonged period of time.

Figure 2.4: Images from an IR Camera Distinguishing Human Temperatures

Advantages:

- The main advantage is that it is very easy to interface, as one sensor controls one vent.
- It is better suited for human detection than pressure sensor as it relies on body temperature.

Disadvantages:

- Will be needing at-least one per seat, the wiring will increase.
- This method not very reliable for Indian subcontinent, as the ambient temperature is already high.

2.3.3 Ultrasonic/Range Sensor

This method utilizes the difference of distances between two seats in the presence or absence of person. We place an ultrasonic sensor at the back of the seat in front and then measure the distance. In the presence of a human the distance is bound to decrease.

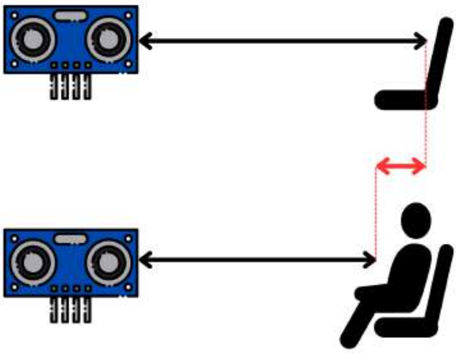

Figure 2.5: Ultrasonic Sensor Working, Red line depicts the difference in measurements

Advantages:

- The main advantage is that it is very easy to interface, as one sensor controls one vent.
- It is already used in automatic light and fans in rooms.

Disadvantages:

- Will be needing at-least one per seat, the wiring will increase.
- This method not be able to distinguish between Humans and other objects kept on seat.

2.3.4 Capacitive Detection

The idea is to setup Electric fields in the vicinity of the seats and detect interference in the fields as humans have a characteristic capacitance which can be detected accurately.

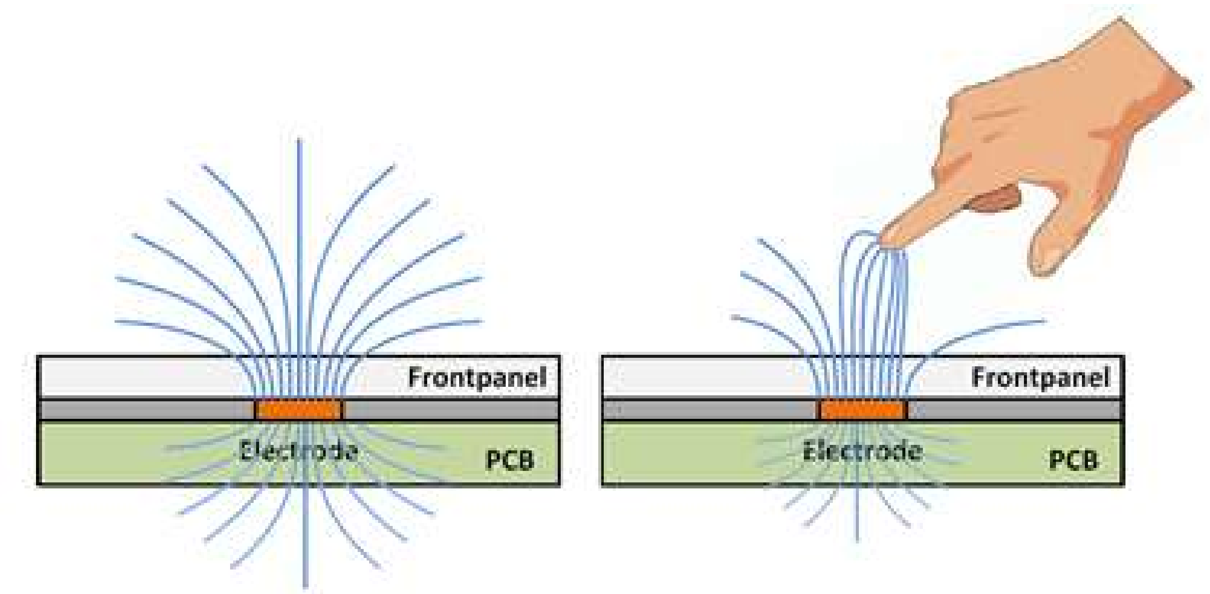

Figure 2.6: Working of Capacitive Detection

Advantages:

- Can accurately detect human presence despite motion, clothing, position, etc.
- It is already used in safety equipments like chainsaws, blades etc.

Disadvantages:

- Is not used at this scale and application before, therefore field strength required is not known and hard to calculate.
- The field may interfere with workings of other components and to be isolated and contained.

2.3.5 Computer Vision and Machine Learning

This track requires to have cameras placed in the bus and to detect humans using pre-trained algorithms running on a micro processor. It utilizes already present camera to track humans on the seats using bounding boxes.

Figure 2.7: Passenger Detection using Computer Vision

Advantages:

- The main advantage is that it needs a few cameras to cover the entire vehicle.
- It can be used for counting passengers in vehicle and enhancing security in vehicles.

Disadvantages:

- Need large compute power even if we use pre-trained models for detection.
- This method is not very suitable if seats are high (have head support) as it will block the view of the passengers behind.

After evaluating all possible sensing methods, the final choice was to adopt a **Computer Vision based approach**. This method offers the advantage of requiring only a single sensing element a camera while still being capable of reliably detecting human presence. Its flexibility, ease of integration with modern embedded systems, and potential for future scalability made it the most practical and robust solution for vent automation.

Chapter 3

Work Done

3.1 Vent Opening Mechanism

Automating the A.C. vents involves the design and integration of both mechanical and electronic components. The following key tasks are necessary to realize the following systems -

3.1.1 Actuator Selection

The selection of a suitable actuator depends primarily on three factors: **power consumption**, **ease of availability**, and **ease of integration on production lines**.

- **Servo Motors:** These actuators are energy-efficient and readily available in the market. However, they require precise alignment and mounting, which may pose challenges in large-scale production environments where repeatability and minimal manual intervention are crucial.

- **Solenoid Valves:** Although they consume relatively more power, solenoid valves are easier to fit into existing production setups due to their simpler actuation mechanism and mounting requirements. Their widespread industrial use also makes them easy to source.

3.1.2 C.A.D. of Mechanism

Designing the vent opening mechanism required careful planning to ensure that the actuator could move the vent flaps reliably. For this, a 3-D model of the mechanism was developed using C.A.D. software (Autodesk's Fusion-360). The CAD model helped us:

- Visualize and simulate the motion of the flaps to confirm they could fully open and close.

- Check if the mechanism fits well with the existing vent design, without the need for major modifications.

- Find the best position for the actuator so that it's easy to install and maintain.

Following are the images of the 3-D model and the sectional diagrams of the prototype made.

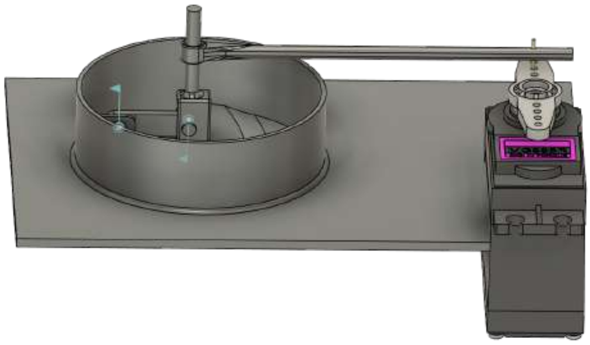

Figure 3.1: Vent Design with additional control

Figure 3.2: Control Mechanism, Vent State: Closed

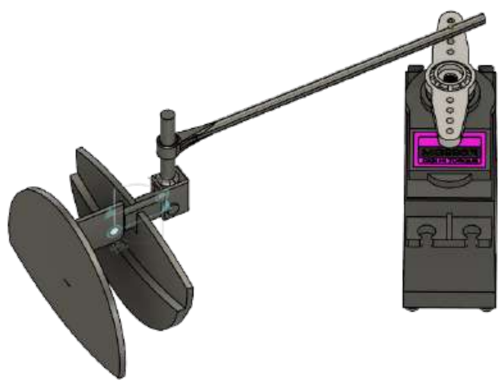

Figure 3.3: Control Mechanism, Vent State: Open

3.2 Automation Algorithm Design

To control the vent opening and closing automatically, a simple yet effective algorithm is designed. The logic is based on inputs, such as passenger presence, and user preferences. These inputs are processed to determine the optimal vent status. The decision-making process is illustrated in the flow diagram below.

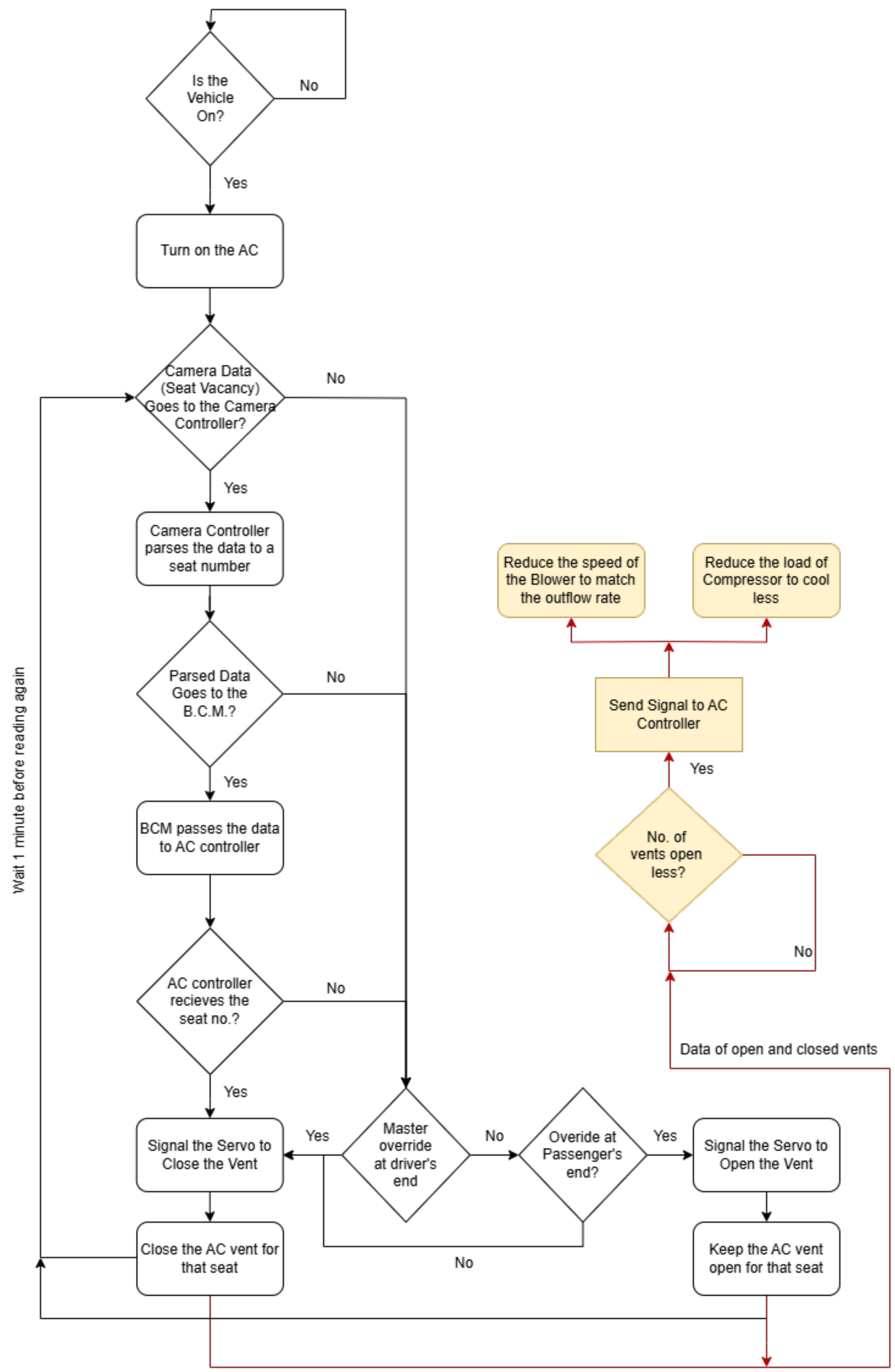

Figure 3.4: Flow diagram representing the logic for Vent Automation

The automation algorithm begins by checking whether the vehicle is powered on. Once the vehicle is on, the system proceeds to activate the air conditioning. The next step involves detecting passenger presence.

The core logic of the algorithm is built around the assumption that vents should remain open by default. This approach prioritizes passenger comfort by placing constraints only on vent closure. In the event of a malfunction in any subsystem, the default open state ensures that airflow is not interrupted.

Passenger detection is carried out using a camera-based system, which processes visual data to determine seat occupancy. Based on this information, the system identifies which vents need to be closed. This seat-specific data is then transmitted to the A.C. Controller via the Body Control Module (BCM).

Importantly, if the data transmission fails at any stage, the vents remain open by design. This prevents erroneous closures, such as a vent shutting off when a passenger is actually present.

Once the A.C. Controller receives the occupancy data, it toggles the vents accordingly. A master control is available to the driver, allowing them to override the system and open or close all vents simultaneously, a useful feature for cooling shuttle buses preemptively before passengers board. Additionally, passengers are given manual control to close their individual vents or open them if they remain closed unintentionally.

Moreover, the data regarding the number of open and closed vents can be fed back to the A.C. Controller. This enables dynamic adjustment of blower speed and compressor settings, leading to optimized energy consumption and improved system efficiency.

The algorithm currently runs at a rate of 1 frame per minute. This is done to minimize the computing power required to run the pretrained model on an edge computing device, like the Camera Controller. This also makes sure that the vents remain open even if passenger moves for a while.

3.3 A.C. & Blower Control

The A.C. vents are designed to operate optimally at an airflow rate between 5 to 7.5 m/s. However, when multiple vents are closed — either due to unoccupied seats or manual passenger inputs — the airflow dynamics change. This can result in increased flow velocity through the remaining open vents, potentially compromising passenger comfort.

To address this, the blower speed must be dynamically adjusted based on the number of open vents. By reducing the blower speed when fewer vents are open, we can maintain consistent airflow across all active vents, ensuring comfort and reducing unnecessary energy consumption. Following table contains the information of all the vents present in a vehicle.

Vent Type	**Vent Length** (mm)	**Vent Breadth** (mm)	**Vent Area** (cm^2)
Circular (controllable vent)	45	45	15.89625
Rectangular (small and always open)	92	5	4.6
Rectangular (bigger and always open)	130	40	52

From this, using fluid dynamics we can calculate what is the total outflow rate of the vents and the total inflow rate from the blowers. Using this data we can control the blowers' speed and also save on power.

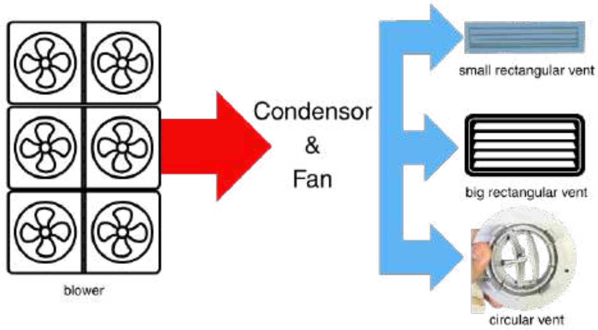

Figure 3.5: Air Flow diagram of AC

The total volumetric flow rate from all the vents, assuming there are n_c circular vents, n_{rs} small rectangular vents and n_{rb} big rectangular vents with the flow rate being $7.5m/s$ is -

$$(15.9n_c + 4.6n_{rs} + 53n_{rb})\frac{7.5}{10^4}\ m^3/s$$

If there are n_b blowers and each blower has an area of $A_b\ m^2$ with flow rate of $f_b\ m/s$, the total volumetric flow rate from blowers is -

$$n_b A_b f_b\ m^3/s$$

At all vents open, the total flow rate out from vents should equal the total flow rate in from the blowers as no air remains in the ducts. This means -

$$(15.9n_c + 4.6n_{rs} + 53n_{rb})\frac{7.5}{10^4} = n_b A_b f_b$$

If we assume $n_{c_{closed}}$ number of controllable vents are closed, the number of blowers required are n'_b, then -

$$(15.9(n_c - n_{c_{closed}}) + 4.6n_{rs} + 53n_{rb})\frac{7.5}{10^4} = n'_b A_b f_b$$

$$(15.9n_c + 4.6n_{rs} + 53n_{rb})\frac{7.5}{10^4} - n_{c_{closed}}\frac{15.9 \times 7.5}{10^4} = n'_b A_b f_b$$

$$n_b A_b f_b - n_{c_{closed}}\frac{15.9 \times 7.5}{10^4} = n'_b A_b f_b$$

$$n_b - n_{c_{closed}}\frac{15.9 \times 7.5}{10^4 A_b f_b} = n'_b$$

Therefore, the number of Blowers that can be closed $= \left\lfloor n_{c_{closed}}\frac{1.2}{100A_b f_b}\right\rfloor$, where $\lfloor x \rfloor$ means the floor value of x. In case we want to consider the difference of densities of the hot air being flowed in and the cold air being flowed out, we can use -

$$\left\lfloor n_{c_{closed}}\frac{1.2\rho_{cold}}{100A_b f_b \rho_{hot}}\right\rfloor$$

3.4 A.C. Vent Automation Pipeline

3.4.1 Component Selection

The P.O.C. required the simulations of the various components which are present in the vehicle. For cost savings and ease of use the following components were selected along with rationale and requirements -

S.no	Component	Requirements	Selected Device	Rationale
1.	Camera	Image to be clear, and use CSI protocol	Raspberry Pi Camera	Checked the requirements and is of Raspberry Pi, ease in interfacing
2.	Camera Controller	Can run light weight human detection algorithms, Can forward a seat number ahead	Raspberry Pi Zero 2 W	Good compute power, basically a computer in a package
3.	Body Control Module	To take input a digital bitstream and to forward it to the AC Controller	ESP32 WROOM-32	Easy to work with, can take in data well
4.	A.C. Controller	Have multiple GPIO pins to toggle the various Vents. To be able to receive a bitstream and send signal to the actuator	ESP32 WROOM-32	Easy to work with, can take in data well, and multiple output pins with good enough compute to calculate the change in Fan speed etc
5.	5V Buck Converter	Needed to change voltage from 24V of bus to 5 needed for the Controller	LMR51420 - 5V	Cheap
6.	3.3V Buck Converter	Needed to change voltage from 24V of bus to 3.3V needed for the Controller	LMR51420 - 3.3V	Cheap
7.	Actuators	Not consume much power. Have enough length when toggled to the extended position.	MG-996R Servo Motor	Cheap, Available in offline stores

3.4.2 Pipeline Overview

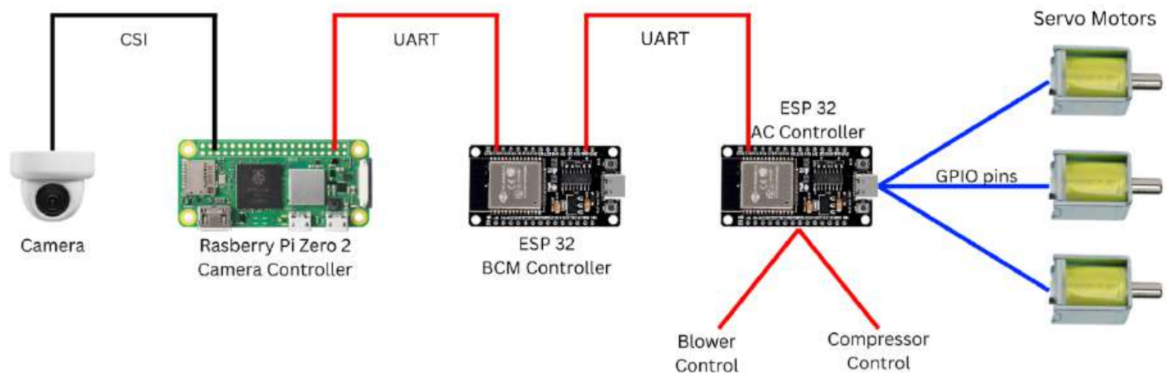

Figure 3.6: Pipeline Overview

The above image shows the pipeline with the electronics modules as the components in the vehicle. The first leg of the pipeline has the camera and the camera controller. The camera takes a picture at regular intervals and sends it to the camera controller via a CSI (Camera Serial Interface) port on the Raspberry Pi. The Raspberry Pi runs the human detection algorithm and determines the seat numbers of the vacant and occupied seats.

This seat occupancy data is then passed to the Body Control Module (BCM), which in our implementation is an ESP-32 microcontroller. The communication between the Raspberry Pi and the BCM is handled via a UART (Universal Asynchronous Receiver/-Transmitter) protocol. UART was chosen for its simplicity and ease of implementation in prototyping stages. However, the system is modular and can later be upgraded to CAN (Controller Area Network) for better noise immunity and scalability.

The final stage of the pipeline is the AC Controller module. Communication between the BCM and the AC Controller can be done via UART protocols. For ESP32-to-ESP32 communication, it was chosen due to its superior efficiency in multi-device setups. Based on the received data, the corresponding actuator is activated to open the respective AC vent. Additionally, the AC Controller manages the Blower and Compressor units by dynamically adjusting their states depending on environmental and occupancy conditions. It also has the 30 second timer which closes the vent.

3.5 Hardware Design and Integration

I used the Arduino IDE for programming the ESP32 modules. The download link for the IDE is provided in the References section. To upload code to the ESP32, connect it to your laptop or computer using a micro-USB cable.

In the Arduino IDE, first install the ESP32 Board Package by Espressif Systems (or the Arduino AVR Boards package, which is for AVR-based boards like the Arduino Uno). You can do this via Tools => Board => Boards Manager, then search for **ESP32** and install it.

Once the package is installed, go to Tools => Board and select ESP32 Dev Module. Also, ensure that the correct COM port is selected (usually COM3, COM4, or COM5 depending on your system).

Now you're ready to upload codes to the ESP32 connected to the device. For Raspberry Pi Setup I have attached a video link in the References Section, Which Installs Pi OS on the board following which one can install the necessary packages, and IDE to run a python script.

3.5.1 Circuit Digrams

Following are the images of the circuit Diagram and the actual Circuit made -

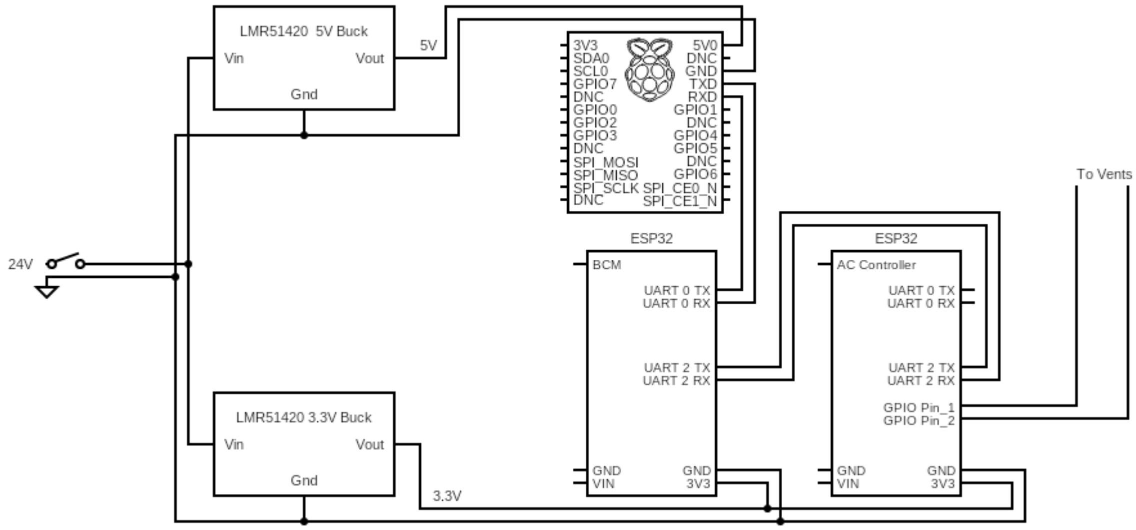

Figure 3.7: Circuit Diagram

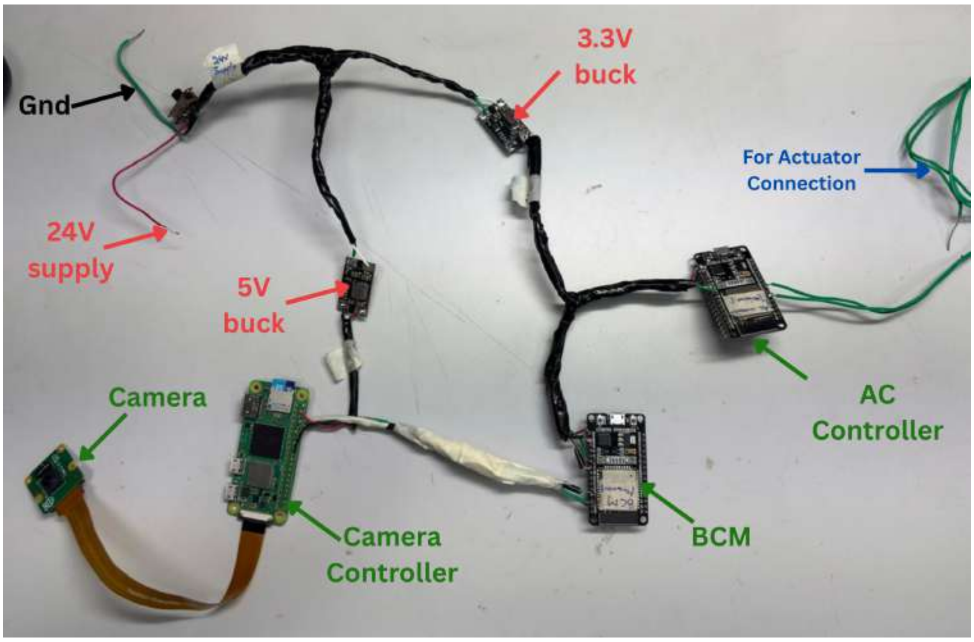

Figure 3.8: Hardware Realization Of Circuit

3.5.2 Actuator Mechanism Design

Following are the images of the Hardware model of the actuation system made in the proto-shop of E.R.C. -

Figure 3.9: Vent Actuation System; Configuration - Close

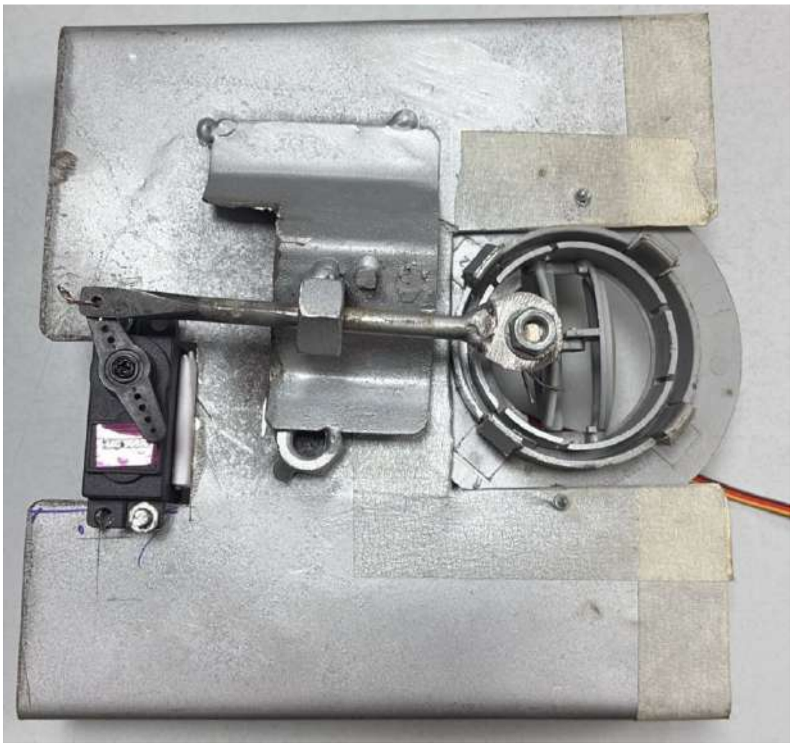

Figure 3.10: Vent Actuation System; Configuration - Open

3.5.3 Controller Codes

Following are the codes written for the various modules. The Descriptions and languages are mentioned below, and a GitHub link to the repository with all the documentation.

Camera Controller

The Code was written in Python and later ported to PI OS for Pi Zero 2 W. This code lets us draw the Areas of Interest (AOIs), and then detects humans in those Boxes.

```python
import cv2
import numpy as np
import serial
import time

rois = []
drawing = False
ref_point = []

# --- Detection Parameters ---
DETECTION_PARAMS = {
    "winStride": (5, 4),
    "padding": (16, 16),
    "scale": 1.05,
    "hitThreshold": -0.35,
}

# --- Mouse Callback Function ---
def draw_roi(event, x, y, flags, param):
    global ref_point, drawing, rois, frame

    if event == cv2.EVENT_LBUTTONDOWN:
        ref_point = [(x, y)]
        drawing = True

    elif event == cv2.EVENT_MOUSEMOVE and drawing:
        frame_copy = frame.copy()
        cv2.rectangle(frame_copy, ref_point[0], (x, y), (0, 255, 0), 2)
        cv2.imshow("Human Detector - ROI Selection", frame_copy)

    elif event == cv2.EVENT_LBUTTONUP:
        ref_point.append((x, y))
        drawing = False
        x1, y1 = ref_point[0]
        x2, y2 = ref_point[1]
        rois.append((min(x1, x2), min(y1, y2), max(x1, x2), max(y1, y2)))
        print(f"ROI added: {rois[-1]}")

# --- Main Function ---
def main():
    global frame

    # --- Serial Port Setup ---
    try:
        # Adjust 'COM3' to your specific port if necessary it can be viewed in the Arduino IDE or Device Manager.
        # For Windows, it might be something like 'COM3' or 'COM4'
        # For Linux, it might be something like '/dev/ttyUSB0'
        # For macOS, it might be something like '/dev/cu.usbmodemXXXX'
```

```
        ser = serial.Serial('COM3', 115200, timeout=1)
        time.sleep(2)  # Allow time for Arduino or device to initialize
        print("Serial connection established on COM Port")
    except Exception as e:
        print(f"Warning: Could not open serial port. {e}")
        ser = None

    # --- HOG Detector Setup ---
    hog = cv2.HOGDescriptor()
    hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

    # --- Video Capture Setup ---
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Could not open camera.")
        return


    window_name = "Human Detector - ROI Selection"
    cv2.namedWindow(window_name)
    cv2.setMouseCallback(window_name, draw_roi)

    detection_mode = False

    # --- Instructions ---
    print("Welcome to the Human Box Detector!")
    print("--- ROI Selection Mode ---")
    print("Press 'd' to start drawing a new box.")
    print("Press 'c' to clear all boxes.")
    print("Press 's' to save ROIs and start detection.")
    print("Press 'q' to quit.")

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Error: Failed to capture frame.")
            break

        frame = cv2.flip(frame, 1)
        key = cv2.waitKey(1) & 0xFF

        if key == ord('q'):
            break
        elif key == ord('s') and not detection_mode:
            if not rois:
                print("Warning: No ROIs defined. Press 'd' to draw at least one.")
            else:
                detection_mode = True
                print("\n--- Detection Mode Started ---")
                cv2.destroyWindow(window_name)
                window_name = "Human Detector - Running"
                cv2.namedWindow(window_name)
        elif key == ord('c'):
            rois.clear()
            print("All ROIs cleared.")
        elif key == ord('d'):
            print("Ready to draw. Click and drag your mouse on the window.")

```

```python
        if not detection_mode:
            display_frame = frame.copy()
            cv2.putText(display_frame, "Mode: ROI Selection", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
            cv2.putText(display_frame, "d: Draw | c: Clear | s: Start | q: Quit", (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
            for (x1, y1, x2, y2) in rois:
                cv2.rectangle(display_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.imshow(window_name, display_frame)

        else:
            display_frame = frame.copy()
            cv2.putText(display_frame, "Mode: Detection | q: Quit", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
            status_report = []

            for i, (x1, y1, x2, y2) in enumerate(rois):
                roi_frame = frame[y1:y2, x1:x2]
                if roi_frame.size == 0:
                    continue

                detections, _ = hog.detectMultiScale(roi_frame, **DETECTION_PARAMS)
                roi_has_human = len(detections) > 0
                roi_color = (0, 255, 0) if roi_has_human else (0, 0, 255)

                status_report.append(f"ROI #{i+1}: {'DETECTED' if roi_has_human else 'Clear'}")

                cv2.rectangle(display_frame, (x1, y1), (x2, y2), roi_color, 2)
                cv2.putText(display_frame, f"ROI {i+1}", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, roi_color, 2)

                for (x, y, w, h) in detections:
                    abs_x, abs_y = x + x1, y + y1
                    cv2.rectangle(display_frame, (abs_x, abs_y), (abs_x + w, abs_y + h), (255, 0, 0), 2)

                # --- Serial Output on Detection ---
                if roi_has_human and ser:
                    message = f"DETECTED in Frame {i+1}\n"
                    ser.write(message.encode('utf-8'))

            print(' | '.join(status_report), end='\r')
            cv2.imshow(window_name, display_frame)

    # --- Cleanup ---
    cap.release()
    if ser:
        ser.close()
        print("\nSerial connection closed.")
    cv2.destroyAllWindows()
    print("\nApplication closed.")

# --- Main Entry Point ---
if __name__ == '__main__':
    main()
```

Body Control Module

The Code was written in Arduino's Language which very closely resemble with C.

```
#include <Arduino.h>

// UART2 Pins for forwarding data
#define UART2_TX 17
#define UART2_RX 16

// Status LED (optional)
#define STATUS_LED 2

// Communication parameters
const int USB_BAUD = 115200;    // USB Serial baud rate (matches Python)
const int UART2_BAUD = 9600;    // UART2 baud rate (matches receiver
    ESP32)

// Timing and status variables
unsigned long lastDataTime = 0;
unsigned long ledBlinkTime = 0;
bool ledState = false;
bool debugMode = true;
int messageCount = 0;

void setup() {
  // Initialize USB Serial (Serial0) - for receiving from Python
  Serial.begin(USB_BAUD);
  delay(1000); // Give serial time to initialize

  Serial2.begin(UART2_BAUD, SERIAL_8N1, UART2_RX, UART2_TX);
  Serial.println("UART2 initialized on pins TX:17, RX:16");

  // Initialize status LED
  pinMode(STATUS_LED, OUTPUT);
  digitalWrite(STATUS_LED, LOW);
}

void loop() {
  // Check for data from USB Serial (Python code)
  if (Serial.available()) {
    String receivedData = Serial.readStringUntil('\n');
    receivedData.trim();

    if (receivedData.length() > 0) {
      lastDataTime = millis();
      messageCount++;

      // Process and forward the message
      String forwardMessage = processMessage(receivedData);

      if (forwardMessage.length() > 0) {
        Serial2.println(forwardMessage);
      }
    }
  }

  delay(10); // Small delay to prevent excessive CPU usage
}

// Process incoming message and format for UART2
```

```
String processMessage(String input) {
  String output = "";

  // Look for detection messages from Python
  if (input.indexOf("DETECTED in Frame") != -1) {
    // Extract frame number
    int frameStart = input.indexOf("Frame ") + 6;
    int frameEnd = input.indexOf(" ", frameStart);
    if (frameEnd == -1) frameEnd = input.length();

    String frameNumber = input.substring(frameStart, frameEnd);
    frameNumber.trim();

    // Format message for receiver ESP32
    output = "Frame " + frameNumber;

    Serial.print("[PROCESS] Detection in Frame ");
    Serial.print(frameNumber);
    Serial.println(" - Forwarding to receiver");

  } else if (input.indexOf("Frame") != -1) {
    // Direct frame message - forward as is
    output = input;
    Serial.println("[PROCESS] Direct frame message - Forwarding");

  } else {
    // Unknown message format
    if (debugMode) {
      Serial.print("[PROCESS] Unknown format: ");
      Serial.println(input);
    }
  }

  return output;
}
```

A.C. Controller

The Code was written in Arduino's Language which very closely resemble with C.

```
#include <Arduino.h>

// UART Pins
#define UART_TX 17
#define UART_RX 16

// Servo PWM GPIOs
#define VENT1_SERVO_PIN 32  // Control for Frame 1
#define VENT2_SERVO_PIN 25  // Control for Frame 2

// Servo parameters for MG996R
const int pwmFreq = 50;              // 50Hz
const int pwmResolution = 16;        // 16-bit resolution
const int maxDuty = 65535;           // 2^16 - 1

// Servo angles
const int VENT_OPEN_ANGLE = 220;      // Open at 90
const int VENT_CLOSE_ANGLE = 0;      // Closed at 0

// Timing variables
```

```
unsigned long lastReceivedTime1 = 0;
unsigned long lastReceivedTime2 = 0;

//Default State of vents
bool vent1Closed = false;
bool vent2Closed = false;

// Debug flag
bool debugMode = true;

void setup() {
  Serial.begin(115200);
  delay(1000); // Give serial time to initialize

  Serial.println("=== ESP32 Servo Controller Starting ===");

  // Initialize UART2
  Serial2.begin(9600, SERIAL_8N1, UART_RX, UART_TX);
  Serial.println("UART2 initialized on pins RX:16, TX:17");

  // PWM channel setup with new API (ESP32 Arduino Core v3.x)
  if (!ledcAttach(VENT1_SERVO_PIN, pwmFreq, pwmResolution)) {
    Serial.println("ERROR: Failed to setup PWM for Servo 1");
  } else {
    Serial.println("PWM for Servo 1 setup successful");
  }

  if (!ledcAttach(VENT2_SERVO_PIN, pwmFreq, pwmResolution)) {
    Serial.println("ERROR: Failed to setup PWM for Servo 2");
  } else {
    Serial.println("PWM for Servo 2 setup successful");
  }

  // Initialize vents as open with delay
  Serial.println("Initializing servos to OPEN position...");
  moveServo(VENT1_SERVO_PIN, VENT_OPEN_ANGLE);
  delay(1000); // Give servo time to move
  moveServo(VENT2_SERVO_PIN, VENT_OPEN_ANGLE);
  delay(1000);

  // Initialize timing
  lastReceivedTime1 = millis();
  lastReceivedTime2 = millis();

  Serial.println("=== Setup Complete - Waiting for UART2 data ===");
  Serial.println("Expected format: 'Frame 1' or 'Frame 2'");
  Serial.println("Timeout: Frame 1 = 15s, Frame 2 = 15s");
}

void loop() {
  // Check for UART data
  if (Serial2.available()) {
    String received = Serial2.readStringUntil('\n');
    received.trim();

    if (received.length() > 0) {
      if (debugMode) {
        Serial.print("[UART] Received: '");
        Serial.print(received);
        Serial.println("'");
```

```
      }

      // Check for Frame 1
      if (received.indexOf("Frame 1") != -1) {
        lastReceivedTime1 = millis();
        Serial.println("[FRAME1] Signal detected - Opening vent 1");

        if (vent1Closed) {
          moveServo(VENT1_SERVO_PIN, VENT_OPEN_ANGLE);
          vent1Closed = false;
          Serial.println("[FRAME1] Vent 1 opened");
        } else {
          Serial.println("[FRAME1] Vent 1 already open");
        }
      }
      // Check for Frame 2
      else if (received.indexOf("Frame 2") != -1) {
        lastReceivedTime2 = millis();
        Serial.println("[FRAME2] Signal detected - Opening vent 2");

        if (vent2Closed) {
          moveServo(VENT2_SERVO_PIN, VENT_OPEN_ANGLE);
          vent2Closed = false;
          Serial.println("[FRAME2] Vent 2 opened");
        } else {
          Serial.println("[FRAME2] Vent 2 already open");
        }
      }
      else {
        Serial.print("[WARNING] Unknown message: ");
        Serial.println(received);
      }
    }
  }

  // Timeout logic for Frame 1 (15 seconds) Change 15000 to desired time in ms
  if (!vent1Closed && (millis() - lastReceivedTime1 > 15000)) {
    Serial.println("[TIMEOUT] Frame 1 timeout - Closing vent 1");
    moveServo(VENT1_SERVO_PIN, VENT_CLOSE_ANGLE);
    vent1Closed = true;
  }

  // Timeout logic for Frame 2 (15 seconds) Change 15000 to desired time in ms
  if (!vent2Closed && (millis() - lastReceivedTime2 > 15000)) {
    Serial.println("[TIMEOUT] Frame 2 timeout - Closing vent 2");
    moveServo(VENT2_SERVO_PIN, VENT_CLOSE_ANGLE);
    vent2Closed = true;
  }

  // Debug status every 10 seconds
  static unsigned long lastDebugTime = 0;
  if (debugMode && (millis() - lastDebugTime > 10000)) {
    printStatus();
    lastDebugTime = millis();
  }

  // Check for serial commands for testing
  if (Serial.available()) {
```

```
    String command = Serial.readStringUntil('\n');
    command.trim();
    command.toLowerCase();

    if (command == "test1") {
      Serial.println("[TEST] Testing Servo 1");
      moveServo(VENT1_SERVO_PIN, VENT_CLOSE_ANGLE);
      delay(1000);
      moveServo(VENT1_SERVO_PIN, VENT_OPEN_ANGLE);
    }
    else if (command == "test2") {
      Serial.println("[TEST] Testing Servo 2");
      moveServo(VENT2_SERVO_PIN, VENT_CLOSE_ANGLE);
      delay(1000);
      moveServo(VENT2_SERVO_PIN, VENT_OPEN_ANGLE);
    }
    else if (command == "status") {
      printStatus();
    }
    else if (command == "debug") {
      debugMode = !debugMode;
      Serial.print("[DEBUG] Debug mode: ");
      Serial.println(debugMode ? "ON" : "OFF");
    }
  }

  delay(50); // Small delay to prevent excessive CPU usage
}

// Improved servo control function for MG996R
void moveServo(int pin, int angle) {
  // Constrain angle
  angle = constrain(angle, 0, 180);

  // Calculate duty cycle for MG996R (1ms-2ms pulse width)
  // For 16-bit resolution at 50Hz:
  // 1ms = 3277, 1.5ms = 4915, 2ms = 6553
  int minDuty = 3277;  // 1ms pulse width (0 degrees)
  int maxDuty = 6553;  // 2ms pulse width (180 degrees)

  int duty = map(angle, 0, 180, minDuty, maxDuty);

  ledcWrite(pin, duty);

  // Debug output
  Serial.print("[SERVO] Pin ");
  Serial.print(pin);
  Serial.print(" set to ");
  Serial.print(angle);
  Serial.print("   (duty: ");
  Serial.print(duty);
  Serial.println(")");
}

// Status printing function
void printStatus() {
  Serial.println("=== STATUS ===");
  Serial.print("Vent 1: ");
  Serial.print(vent1Closed ? "CLOSED" : "OPEN");
  Serial.print(" | Last signal: ");
```

```
    Serial.print((millis() - lastReceivedTime1) / 1000);
    Serial.println("s ago");

    Serial.print("Vent 2: ");
    Serial.print(vent2Closed ? "CLOSED" : "OPEN");
    Serial.print(" | Last signal: ");
    Serial.print((millis() - lastReceivedTime2) / 1000);
    Serial.println("s ago");

    Serial.print("Free heap: ");
    Serial.print(ESP.getFreeHeap());
    Serial.println(" bytes");
    Serial.println("=============");
}
```

Images of Working

The images show the POC running the code for Human detection and Vent Opening.

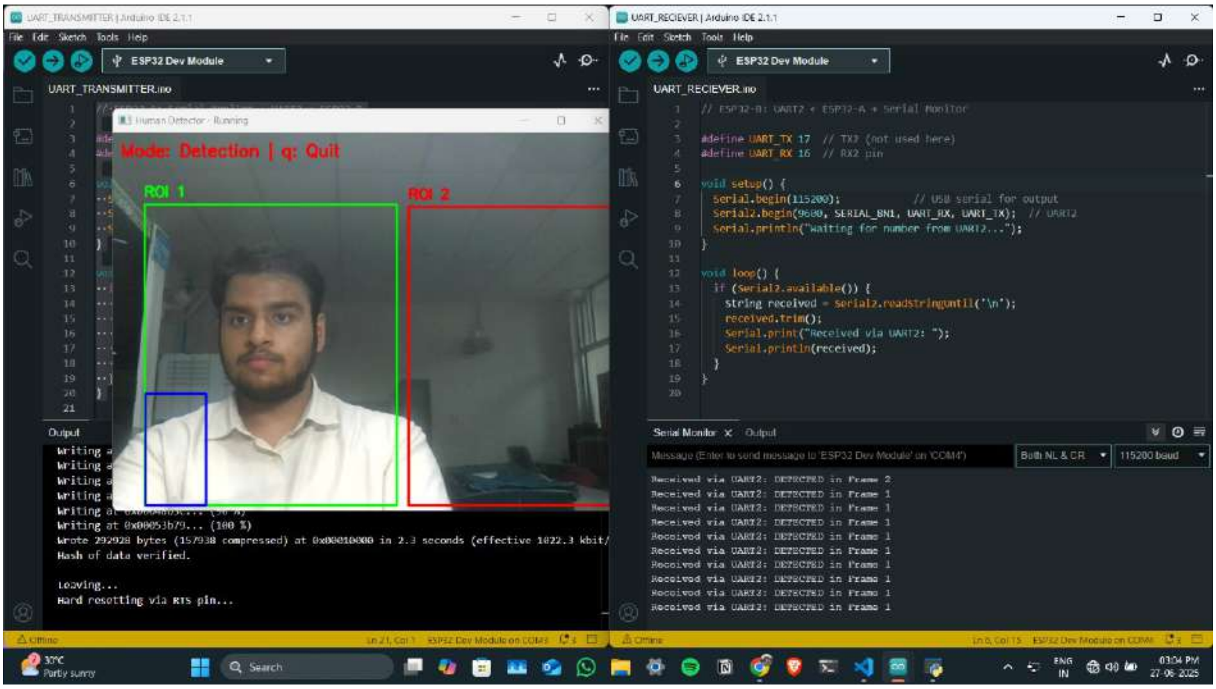

Figure 3.11: Detection in Frame 1

Chapter 4

Future Work

This project lays the foundation for several avenues of future improvements and enhancements. Key directions for advancement, categorized by relevant subsystems, are outlined below.

4.1 Electronics and Controllers

Future iterations of the system can benefit significantly from improved hardware and optimized software. For instance, a Raspberry Pi 5 can be employed as the camera controller to provide greater processing power, while upgrading to Camera Module 3 will enable higher resolution image capture. Additionally, migrating from UART to CAN bus communication would better emulate the architecture of real-world automotive systems, ensuring more robust and scalable communication.

On the software side, the current human detection method based on HOG can be upgraded to more accurate and efficient deep learning-based approaches such as the YOLO (You Only Look Once) object detection model. Furthermore, the system should eventually be transitioned to run on actual vehicle-grade embedded controllers to ensure real-world applicability.

4.2 Actuator Mechanism

The actuator mechanism controlling the AC vents can be significantly enhanced. A more robust and versatile design should be developed, capable of precise control in all required directions of motion. The mechanism must also be manufacturable at scale and designed for long-term reliability. Special focus should be placed on minimizing mechanical failure, simplifying assembly, and ensuring consistency across multiple units.

Acknowledgment

I would like to express my sincere gratitude to all those who supported and guided me throughout the course of this Summer Internship.

First and foremost, I am deeply thankful to **Kartar Sir**, my project mentor, for his invaluable guidance, encouragement, and technical insights. His expertise and feedback were instrumental in shaping the direction of this work. Under his and his team's mentorship, I was able to learn a great deal about the various vehicles being developed and the manufacturing processes involved.

I would also like to thank the Tata Motors Engineering Research Centre for providing the necessary infrastructure and resources to carry out the research and development of the system. The experience was immensely educational and enriching. I would also like to thank **Girish Sir**, **Surabhi Ma'am**, and **Gurvinder Sir**; without their guidance this project would not have been completed on time. I am equally grateful to the **Human Resources Team** for ensuring a smooth transition and for providing me with this opportunity.

A special thanks to my teammates and peers for their constructive discussions and collaboration throughout the project. Their support helped me overcome several challenges during development and testing.

Lastly, I am deeply grateful to my family and friends for their constant encouragement and moral support, which played a significant role in the successful completion of this project.

References

[1] **Noema Technologies**: Real Time Passenger Detection System in Commercial Vehicles

[2] **US Patent**: Using IR Sensor to sweep a passenger vehicle to detect Passenger Temperature

[3] **Chinese Patent**: Multiple IR Sensors to Detect Passenger Presence and control A.C. Systems

[4] **Arduino IDE**: Download link for the Coding platform of ESP-32s

[5] **Raspberry Pi Zero Setup Guide**: Guide to install the appropriate OS on Raspberry Pi Zero and how to set it up.

[6] **GitHub Link**: Repository Link for all the Code files and Documentation related to the project.