We have our object detection in two phase. In phase 1, we will be train to detect the object type and size locally on our PC and in phase 2, we will deploy the model on some server and using esp32 WIFI we will access the model and detect the model without using the local system.

## Phase 1: Detection and training of model on local system

**Step 1: Capture Image from ESP32 + OV7670:** Send grayscale image data from ESP32 to PC via Serial.

```
#include <Wire.h>
#include <OV7670.h>

OV7670 cam;

uint8_t frame1[80][60]; // Downscaled grayscale
uint8_t frame2[80][60];

void setup() {
  cam.begin();
}

void loop() {
  cam.captureGrayscale(frame1);
  delay(500); // Wait a bit
  cam.captureGrayscale(frame2);

  int diff = computeFrameDiff(frame1, frame2);

}

int computeFrameDiff(uint8_t f1[80][60], uint8_t f2[80][60]) {
  int total = 0;
  for (int i = 0; i < 80; i++) {
    for (int j = 0; j < 60; j++) {
      total += abs(f1[i][j] - f2[i][j]);
    }
  }
  return total;
}

```

## Step 2: PC Python Script to Receive & Process Frame

bash Copy Edit

```
pip install pyserial opencv-python numpy
```

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

IMG_WIDTH = 80
IMG_HEIGHT = 60

ser = serial.Serial('COM3', 115200)  #camera port here

def read_image():
    while ser.inWaiting() < IMG_WIDTH * IMG_HEIGHT:
        pass
    data = ser.read(IMG_WIDTH * IMG_HEIGHT)
    img = np.frombuffer(data, dtype=np.uint8).reshape((IMG_HEIGHT, IMG_WIDTH))
    return img

while True:
    img = read_image()

    # Basic object detection
    _, thresh = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if contours:
        largest = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest)
        print(f"Object Width: {w}")

        cv2.rectangle(img, (x, y), (x+w, y+h), 255, 1)

    cv2.imshow("Image", img)
    cv2.waitKey(1)

```

## Phase 2: Server Deployment + ESP32 HTTP Client

### Step 1: Python Server for Image Upload + Detection

```python
from flask import Flask, request
import numpy as np
import cv2
import base64
from io import BytesIO
from PIL import Image

app = Flask(__name__)

@app.route('/detect', methods=['POST'])
def detect():
    data = request.get_json()
    img_data = base64.b64decode(data['image'])
    img_array = np.frombuffer(img_data, np.uint8)
    img = cv2.imdecode(img_array, cv2.IMREAD_GRAYSCALE)

    # Threshold & detect object
    _, thresh = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    result = {"object_detected": False, "width": 0}
    if contours:
        largest = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest)
        result["object_detected"] = True
        result["width"] = w

    return result

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)
```

## Step 2: ESP32 Sends Image Over Wi-Fi (HTTP POST)

```ino
#include <WiFi.h>
#include <HTTPClient.h>
#include "base64.h"

const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";
const char* serverURL = "http://your_server_ip:5000/detect";

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting...");
  }
  Serial.println("Connected!");
}

void loop() {
  uint8_t image[80 * 60];
  captureGrayscaleFrame(image);

  String encoded = base64::encode(image, sizeof(image));
  HTTPClient http;
  http.begin(serverURL);
  http.addHeader("Content-Type", "application/json");
  String payload = "{\"image\":\"" + encoded + "\"}";
  int status = http.POST(payload);

  if (status == 200) {
    String response = http.getString();
    Serial.println(response);
  }
  http.end();
  delay(3000);
}
```

We'll implement a base64 encoding function.

This process will help us to capture and send images to our server using ESP32 through wi-fi. The server detects the object's type and its width. ESP32 gets response and will rotate the servo motor which will open the doors.s

### 💻 Image 1: Tinkercad Circuit Simulation Overview

**Components Used:**

- Arduino UNO
- PIR Motion Sensor
- L293D Motor Driver IC
- DC Motor

🔌 **Connections Explained:**

**PIR Sensor:**

- VCC (Red wire) → Arduino 5V
- GND (Black wire) → Arduino GND
- OUT (Middle wire) → Arduino Digital Pin 2
    - This is the motion-detection signal pin. When motion is detected, it outputs HIGH (1).

**L293D Motor Driver:**

- Pin 1 (Enable 1) → Arduino Digital Pin 9 (PWM control)
- Pin 2 (Input 1) → Arduino Digital Pin 3
- Pin 7 (Input 2) → Arduino Digital Pin 4
- Pin 8 (Vcc2 / Motor Supply) → External Power Source or 5V
- Pin 16 (Vcc1 / Logic Power) → Arduino 5V
- Pin 4, 5, 12, 13 (GND) → Common GND
- Pin 3 & 6 (Output 1 & 2) → Connected to DC Motor Terminals

**DC Motor:**

- Connected to Output 1 and 2 of L293D (Pins 3 and 6).
- The motor spins when motion is detected by the PIR.

🔬 **Image 2: Real Hardware with Oscilloscope**

This setup displays the real-world connection of a PIR sensor and monitoring using an oscilloscope.

**Components:**

- PIR Sensor:
    - Red wire → VCC
    - Black/White wire → GND
    - Blue wire → Signal Output

- Arduino UNO (with a prototyping shield)
- DS212 Oscilloscope
    - Monitors the digital signal output from the PIR.

**Oscilloscope Screen Insight:**

- A flat line (low signal) indicates no motion.
- When motion is detected, the signal spikes HIGH, visible as a digital pulse.

👷 **System Working Summary:**

- PIR detects motion → Output goes HIGH
- Arduino reads the HIGH signal and sends signals to the L293D
- L293D activates the DC motor (e.g., to open a door or trigger a fan)
- Oscilloscope confirms proper signal response from the PIR sensor

📈 **Signal Conditioning for Doppler Sensor using LM358 Op-Amp**

**Why LM358 is Needed:**

✅ **1. Signal Amplification:**

- Doppler radar sensors output very weak analog signals (millivolts range).
- LM358 amplifies this to detectable voltage levels (0–5V) suitable for microcontrollers.

✅ **2. Waveform Shaping (Comparator Mode):**

- Doppler outputs are often noisy sine waves.
- LM358 used as a comparator converts this into clean digital pulses (HIGH/LOW), making them easier to process.

✅ **3. Noise Filtering:**

- Doppler signals include noise.
- With appropriate RC filtering, LM358 removes unwanted fluctuations for better accuracy.

This makes the LM358 an essential interface between the Doppler sensor and any digital control system like Arduino.

Contribuitons: Our team researched double-door mechanisms, analysing pivot dynamics, torque needs, and sensor integra on while exploring automated single-door conversion. Through collaborate on, we evaluated actuators, control methods, and safety features, establishing key technical parameters for our adapt ve door prototype.

Harsh: Researched on the different types of electronic components required for automa ng double door swing mechanism (PIR sensors), project weekly report and finalise on of materials

Aditya: Researched on the different types of electronic components required for automate ng double door swing mechanism (Microwave sensor, Motor Drivers)

Ayush: Researched on the different types of electronic components required for automate ng double door swing mechanism (Servo and Stepper Motors)

Shreshth: Researched on methods to switch between Double door and Single door mechanism (Image Processing Camera) and purchase of materials