



Module Project Report for  
**CE6023 – Computer Vision Systems**

Student Name : **Harshita Ramesh**

Student ID : **24128953**

Revision Timestamp: 08/12/2024 19:38:57

## Contents

Abstract.....	3
Introduction .....	3
First Task (Setting up Raspberry pi) .....	4
Second Task (Software Installation and Dependencies).....	5
Third Task (Data Collection for face recognition) .....	5
Fourth Task (Load and Configure TFLite Model) .....	6
Fifth Task (Video Saving and End Detection) .....	7
Sixth Task (Issues and Improvements) .....	8
Seventh Task (Performance Across Different Scenarios) .....	9
Discussions .....	9
Conclusion .....	13
References .....	13
Figure 1:Setup of the project .....	4
Figure 2:detecting camera.....	4
Figure 3: Data Collection for facial recognition.....	6
Figure 4: necessary libraries .....	6
Figure 5: load_tflite_model function .....	7
Figure 6: conf_threshold .....	7
Figure 7: plot_decision snippet .....	8
Figure 8: logging issues .....	9
Figure 9: Scenario-1 .....	10
Figure 10: real time detection of scenario-1 from the output video.....	10
Figure 11: Scenario-2.....	11
Figure 12: real time detection of scenario-2 from the output video .....	11
Figure 13: Scenario 3 .....	12
Figure 14: real time detection of scenario-3 from output video .....	12

## Abstract

The idea of this project is to design and implement a computer vision system using Raspberry Pi 4 together with the camera module for detecting and recognizing a dynamic crowd and objects in real time. Within a period of 20 seconds, the system will be able to recognize and follow at least 4 individuals and 1 object and display their labels and confidence level. It assigns tags to people or objects, records entry/exit occurrences, and lists the corresponding degrees of certainty as well as the quality of detections to handle challenges if necessary. OpenCV and dlib, the image analysis and recognition libraries are also included for better image processing in raspberry pi 4 models with 4 GB RAM. One of the most crucial considerations used in the design is the fine-tuning of the camera module taking into account its resolution, exposure controls, and even the conditions in which the camera is being used. These involve issues related to differences in lighting condition where the activities are conducted such as; well-lit indoor environments, outdoor daylight and low light environment. Such conditions are handled through specific algorithms for detection and tracking that enable reliable results. Thus, the emphasis on the ability to process data in real-time mode, on the stability of the algorithm in different conditions, and on the interaction between software and hardware makes the project aim to present a flexible solution for the analysis of dynamic scenes and tracking.

## Introduction

In this report, our primary goal is to develop and introduce an effective computer vision system that will be using a Raspberry Pi 4 alongside a camera module to detect as well as track a dynamic group of individuals and objects in a scene. The task is to construct the system, which would work within the period of 20 seconds, detect at least four individuals and one object, recognize their names, and assign confidence rates to the result, while emphasizing the real-time approach and saving the video record. Some of the objectives include naming or tagging persons and objects in the scene, monitoring on who enters or exits, and a percentage score that goes hand-in-hand with each label. Moreover, the system will also evaluate the quality of the detections and the potential problems that can occur based on factors such as image sensors and/or ISP settings.

The concept behind this system can be complex and consists of several points most related to the characteristics of the hardware and software technologies and environmental conditions that may influence the performance. The core of the system is Raspberry Pi 4 and this will be serving the purpose of the processor of the computer vision of the system. Firstly, Raspberry Pi 4 model comes with a basic configuration of at least 4GB of RAM and has a high processing capabilities which made it possible to process real time object detection and Face recognition. Camera module selection is another critical aspect that determines the system's capabilities, including image quality, frame rate, and light sensitivity. The specific characteristics of the camera, such as the resolution or exposure mode, must be set according to the environment to get the best performance out of it. One must also remember about other components of the system being designed, such as libraries for image

### *Real time detection*

processing and face recognition like OpenCV and dlib. These libraries contain the required algorithms for face detection, motion tracking, and estimation of the confidence level. It will also contain algorithms that can sample performance in various lighting conditions like bright rooms, outdoors, strong back lighting and so on to evaluate how different environmental conditions influence the detection capabilities of the system. Another issue that needs to be addressed in this project is how the proposed system will behave when exposed to different conditions within the environment. For instance, the lighting in the scene can greatly influence the accuracy of the camera module and this requires testing various scenarios such as bright indoor environments, outdoor during the day, or during the night. Furthermore, how the camera module performs in terms of image capturing, sharpness, and its ability for image recognition especially under such conditions as backlighting, is going to be a key factor towards the success of the system.



Figure 1: Setup of the project

## First Task (Setting up Raspberry pi)

Set up the Raspberry Pi 4 by assembling the hardware components, including the camera module, and connecting peripherals like the keyboard, mouse, and monitor. Install the operating system via the welcome screen and configure network settings for both Ethernet and WiFi. Enable SSH for remote access, and ensure the camera module is properly connected. Once setup is complete, configure the system for remote desktop access or use realVNC and test the camera functionality by capturing still images or video. Instead we can also use the command “vcgencmd get\_camera” in terminal to know if the camera is connected.

```
rprdp953@raspberrypi:~ $ vcgencmd get_camera
supported=1 detected=1
rprdp953@raspberrypi:~ $
```

Figure 2: detecting camera

## Second Task (Software Installation and Dependencies)

In the second task we need to install the necessary software libraries, including the Raspberry Pi operating system, OpenCV, dlib, and any additional tools needed for image processing, face recognition, and system monitoring.

We are installing the OpenCV package via command line. We will be using the python 3 version for this laboratory. Enter the following in the command prompt (this could take a few minutes):

- `sudo apt-get install python3-opencv`

after that got installed then enter this command

- `pip3 install face-recognition`

other needed installations:

- `pip install numpy`
- `pip install tf-lite-runtime`
- `pip install matplotlib`

## Third Task (Data Collection for face recognition)

Before initiating the data gathering process of the OpenCV face recognition application on the Raspberry Pi 4, we should open the File Manager and go to final project repository at `~/CE6023/final_project /`. Once there, open the `headshots.py` file in the default Geany editor. Substitute the third line of the file with the corresponding name to your full name or the full name of the person, first name and the last name included and then use the 'Ctrl+S' button to save before closing the editor. Next, navigate to the subfolder named `dataset` in the very same directory as you were working in; create here a new folder and name it after your first and last names. The folder with the photos of the face detector training will be created here. Following this, navigate back to the terminal and type the command `cd ../` or `cd ~/CE6023/final_project/`. Click the command line interface of your system and type `python3 headshots.py` then the camera goes active. Place the Raspberry Pi camera in front of the special faces to capture the images and then press the space bar to make the shots. Take between 10 to 20 pictures from different orientations removing glasses but not hats as they may hinder recognition. Press 'Esc' when done. Last of all, when you are through with the photos take and view them by going to the `dataset` folder then the folder of your photographs. Make sure they are

saved

in

JPEG

files.

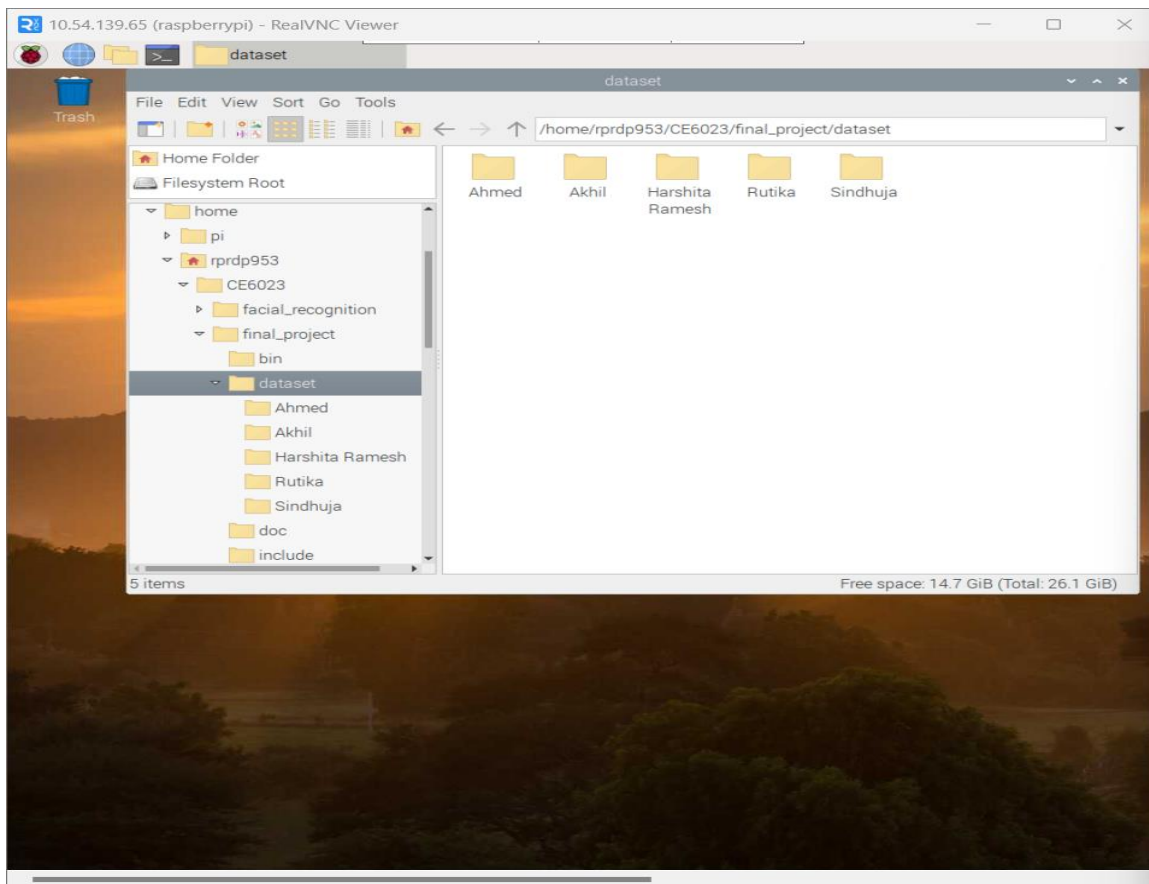


Figure 3: Data Collection for facial recognition

## Fourth Task (Load and Configure TFLite Model)

After the dataset is pre-processed and the required libraries imported next that should be done is setting the object detection model. The real-time object detection applied in the project is based on TensorFlow Lite (TFLite) model. We have to make sure that we have the required TFLite model file running this example (ssd\_mobilenet\_v2.tflite) and its corresponding label map file (labelmap.txt). These files should be placed in the final\_project path. The model, and labels are loaded with the load\_tflite\_model() function and the TFLite interpreter is initialized. This will enable making object detection on the Raspberry Pi run in a very efficient manner.



Figure 4: necessary libraries

```

30 # FUNCTION: LOAD TFLITE MODEL
31 def load_tflite_model(model_path, label_path):
32     interpreter = Interpreter(model_path=model_path)
33     interpreter.allocate_tensors()
34     input_details = interpreter.get_input_details()
35     output_details = interpreter.get_output_details()
36
37     # Load labels
38     with open(label_path, "r") as f:
39         labels = [line.strip() for line in f.readlines()]
40     return interpreter, input_details, output_details, labels
41

```

Figure 5: load\_tflite\_model function

This will enable making object detection on the Raspberry Pi run in a very efficient manner. The general working of the system is based on real-time detection, which includes exporting frames of the video, face detection, object detection and the drawing of a bounding box around the detected object or face. In this system, the video frames are captured using Raspberry Pi camera. For each frame, it captures the image then scans for faces by using face\_recognition. After that, the faces are matched with the encodings stored in the file encodings.pickle to determine the person. If the match is found, details of the person are displayed along with confidence level. Must be identified as "Unknown" if none was matched to all of the characteristics on the above list.

Along with face detection, the system also drives the object detection using the model TFLite. The detect\_objects\_tflite() process for each frame, finds different objects like cars, people, animals, etc and shows the bounding boxes and confidence level of objects in it. The objects detected are passed through a confidence value where objects that have the confidence threshold are only displayed. Since that is the case, we shall establish the confidence threshold at 0.6 because if the confidence is way low, the empty spaces and the low level noises can be mis interpreted as some objects. Additionally, the system checks the quality of the image by detecting if it is blurry, using the variance of Laplacian, and logs warnings if the image is found to be blurry.

```

41
42 # Function: Detect Object Detection
43 def detect_objects_tflite(frame, interpreter, input_details, output_details, labels, conf_threshold=0.6):
44     frame_shape =
45     input_shape = input_details[0]['shape'][1:3]
46
47
48     # Resize frame to match input shape
49     resized_frame = cv2.resize(frame, (input_shape[1], input_shape[0]))
50     # Convert frame to numpy array and cast to uint8

```

Figure 6: conf\_threshold

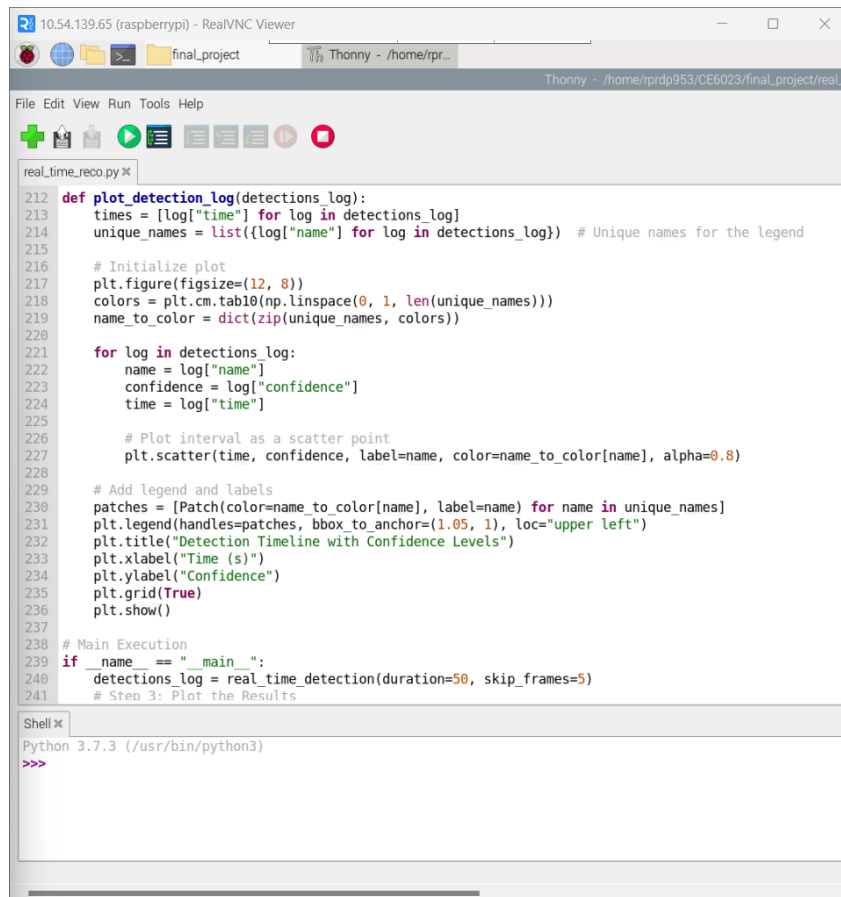
## Fifth Task (Video Saving and End Detection)

To track the detected objects and faces, the system takes screenshots of record the video stream with bounding boxes and labels. Then we use the cv2.VideoWriter() function to save the video into an .avi file so that for subsequent viewing of the monitored objects and faces in the captured video. It is shown that the video file is saved under the given path and then the system continues to run for the specified time (for instance, 20 seconds) then it halts. The video capture and video writer objects are, therefore, released, and the video stream comes to a close. At the end of the detection session, the system produces a plot of the detection over time. The plot\_detection\_log() is a function that comes



### Real time detection

in handy in the plotting of a graph that captures the magnitude of confidence in every detected faces and objects against the time they were detected. Specific colors are assigned for each identified name or object and a legend is generated so that each of them can be distinguished. This enabled graphical study of how the confidence levels changed over time to make an evaluation of the performance of the system during the detection session.

The image shows a screenshot of a Thonny IDE window. The title bar indicates the connection is to a Raspberry Pi at 10.54.139.65. The file explorer shows a project named 'final\_project'. The main editor displays a Python script named 'real\_time\_reco.py'. The script defines a function 'plot\_detection\_log' that takes a 'detections\_log' list as input. It extracts 'time', 'name', and 'confidence' from the log entries. It initializes a plot with a figure size of (12, 8) and assigns unique colors to each object name using a colormap. The plot is a scatter plot where the x-axis is 'Time (s)' and the y-axis is 'Confidence'. Each point is labeled with the object name. A legend is added in the upper left corner. The main execution block calls 'real\_time\_detection' with a duration of 50 and skip\_frames of 5, then calls 'plot\_detection\_log' to display the results. The shell window at the bottom shows the Python 3.7.3 prompt is ready for input.

```
212 def plot_detection_log(detections_log):
213     times = [log["time"] for log in detections_log]
214     unique_names = list({log["name"] for log in detections_log}) # Unique names for the legend
215
216     # Initialize plot
217     plt.figure(figsize=(12, 8))
218     colors = plt.cm.tab10(np.linspace(0, 1, len(unique_names)))
219     name_to_color = dict(zip(unique_names, colors))
220
221     for log in detections_log:
222         name = log["name"]
223         confidence = log["confidence"]
224         time = log["time"]
225
226         # Plot interval as a scatter point
227         plt.scatter(time, confidence, label=name, color=name_to_color[name], alpha=0.8)
228
229     # Add legend and labels
230     patches = [Patch(color=name_to_color[name], label=name) for name in unique_names]
231     plt.legend(handles=patches, bbox_to_anchor=(1.05, 1), loc="upper left")
232     plt.title("Detection Timeline with Confidence Levels")
233     plt.xlabel("Time (s)")
234     plt.ylabel("Confidence")
235     plt.grid(True)
236     plt.show()
237
238 # Main Execution
239 if __name__ == "__main__":
240     detections_log = real_time_detection(duration=50, skip_frames=5)
241     # Step 3: Plot the Results
```

Figure 7: plot\_decision snippet

## Sixth Task (Issues and Improvements)

While running the system, certain issues may arise that need attention. For example, if the detection confidence is low, the system logs warnings about the low confidence and abnormal bounding box sizes. If the system detects blurry images (using the Laplacian variance), a warning is logged, suggesting improvements in focus or lighting. These issues are logged to ensure that the user is aware of potential problems in the system's performance and can take corrective actions.



### Real time detection

```
# Function: Check if Image is Blurry
def is_image_blurry(image, threshold=100):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    variance_of_laplacian = cv2.Laplacian(gray, cv2.CV_64F).var()
    return variance_of_laplacian < threshold

# Function: Log Detection Issues
def log_detection_issues(confidence, box_size, image_blur):
    if confidence < 0.5:
        print(f"Warning: Low detection confidence: {confidence}")
    if box_size < 0.05 or box_size > 0.5:
        print(f"Warning: Bounding box size is abnormal: {box_size}")
    if image_blur:
        print("Warning: The image seems blurry. Try improving the focus or lighting.")
```

Figure 8: logging issues

## Seventh Task (Performance Across Different Scenarios)

The above task is repeated in the different Scenarios to identify the performance and confidence score.

## Discussions

When comparing the performance across different scenarios,

**Brightly Lit Room with backlighting:** I placed the setup in the living room where it was well lit coz the balcony door, and big windows. The facial recognition system had an impressive performance during bright lighting conditions, with very slight facial images not being recognized out rightly. The object detection model also gave high accuracy, identifying normal objects with accuracy of above 0.6. Nevertheless, there was a common problem regarding positioning bounding boxes. In this case the bounding boxes that refer to the detected objects are placed at the wrong area of a frame or drawn with the wrong scale. The bounding box scaling function, which was intended to map the normalized coordinates back to pixel values, was improperly calculated, resulting in bounding boxes being placed incorrectly. confidence levels and object detection performance remained strong. In some cases we can see that the Warnings were issued for the abnormal size of bounding box, this was handled by the ISP part. The confidence was on average of 65-70% all the time. The confidence level was maintained in average eventho the persons were little afar from the camera.

## Real time detection

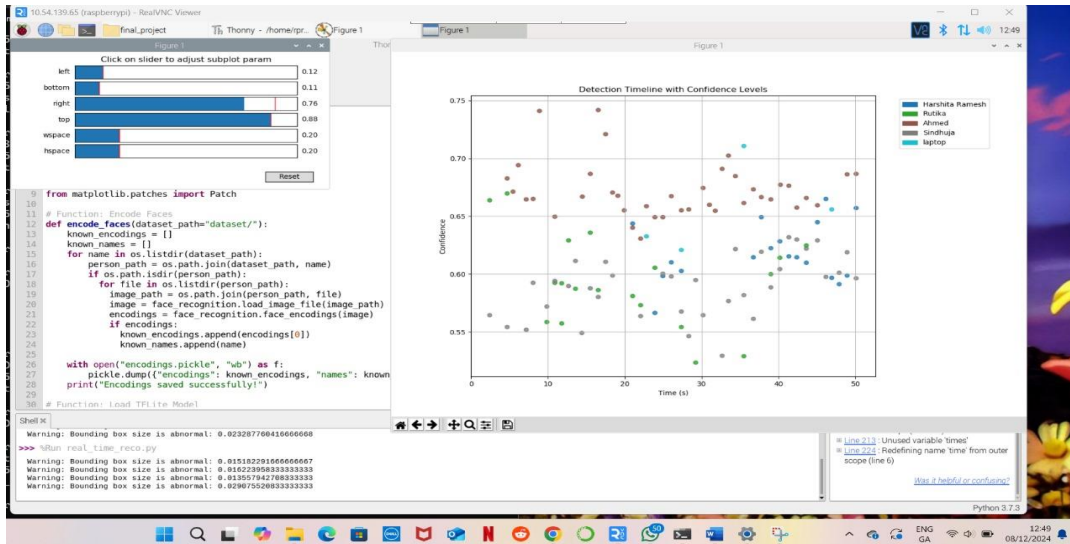


Figure 9: Scenario-1

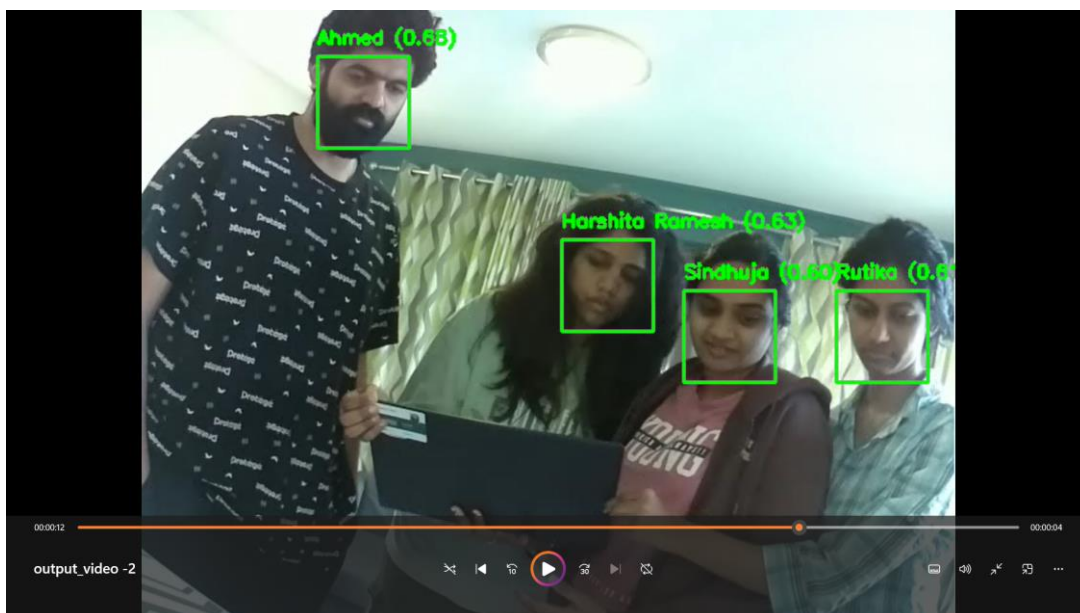


Figure 10: real time detection of scenario-1 from the output video

**Backlighting (Kitchen):** The kitchen had irregular lighting system, with some part brightly lit and other part poorly lit. This led to occasional errors in face detection, more so on areas where direct light was not revealed on the faces. Sometimes the system was unable to identify the face because it was in a shadow or unable to track the face when moving it through regions with scarce lighting. These problems were most apparent at the peripheral areas and in the distant area away from the camera of the room with lesser lighting quality. The object detection system worked reasonably well, but sometimes it confused objects when the contrast of some objects in the scene was low and the lighting in the scene was inconsistent. At the end of the video capturing at one point for a sec a misidentification occurred and the empty space/ some other object was confused to be a cat.

## Real time detection

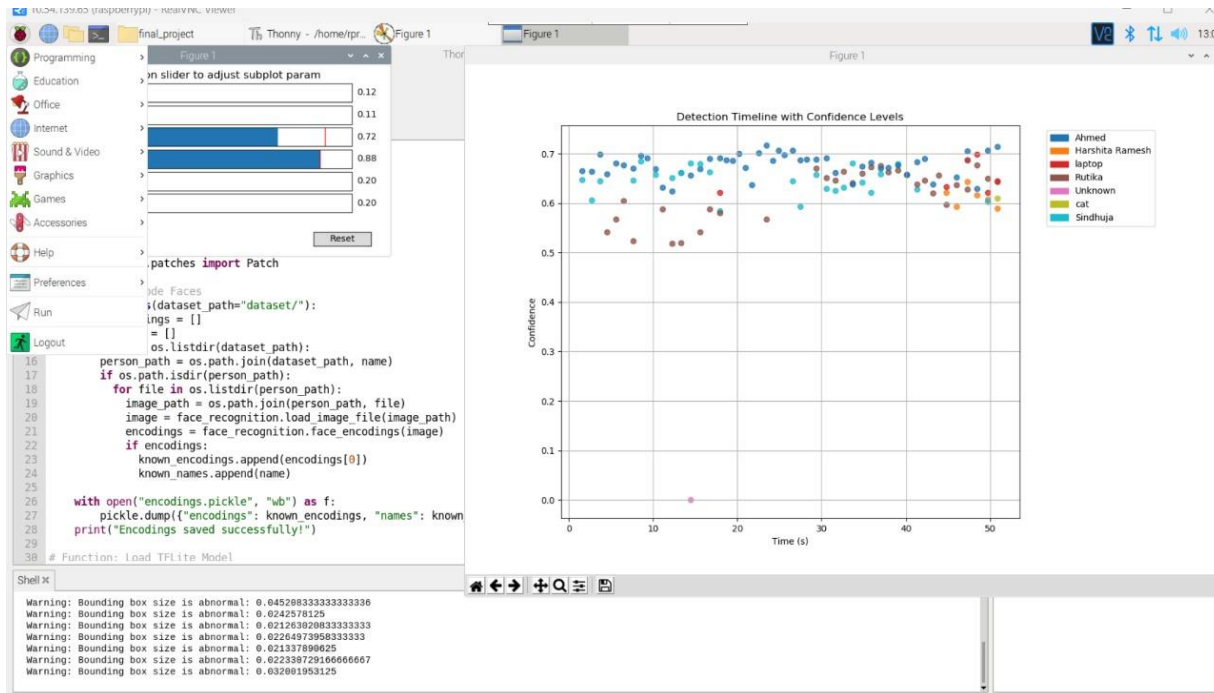


Figure 11: Scenario-2

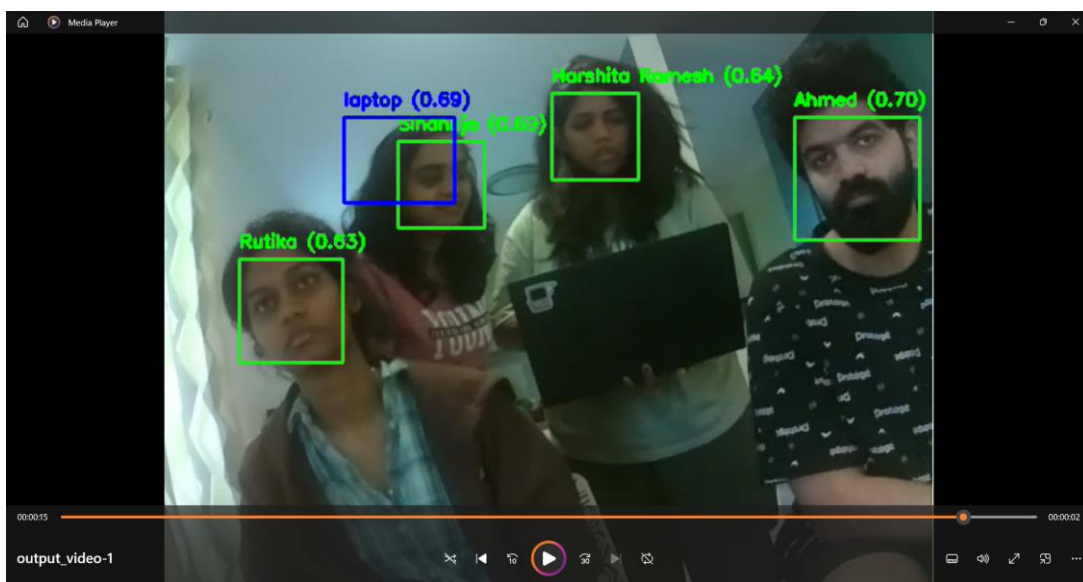


Figure 12: real time detection of scenario-2 from the output video

**Dimly lit room:** In the dimly lit bedroom, where the lighting was weak, significant issues arose. The object detection system struggled the most in this environment, frequently identifying empty spaces as objects 2–3 times during the detection period. The system's confidence scores for these false detections were low, but the system still labelled these empty areas/stickers in the walls as objects, likely due to a combination of insufficient lighting and low contrast. Additionally, the bounding boxes were misaligned, often placed outside of the object areas or showing areas with no objects at all. This behaviour was exacerbated by the ISP processing, which couldn't adjust properly to such low-light conditions, resulting in incorrect bounding box sizes. The warning related to bounding box abnormalities ("Bounding box size is abnormal:") was triggered frequently, as the system detected large bounding boxes around empty spaces, further indicating issues with the ISP configuration and the image quality in such conditions. The confidence score increased to 80% when the

### Real time detection

person[ Ahmed] and object was near to the camera where there was good lightening when compared to other spots.

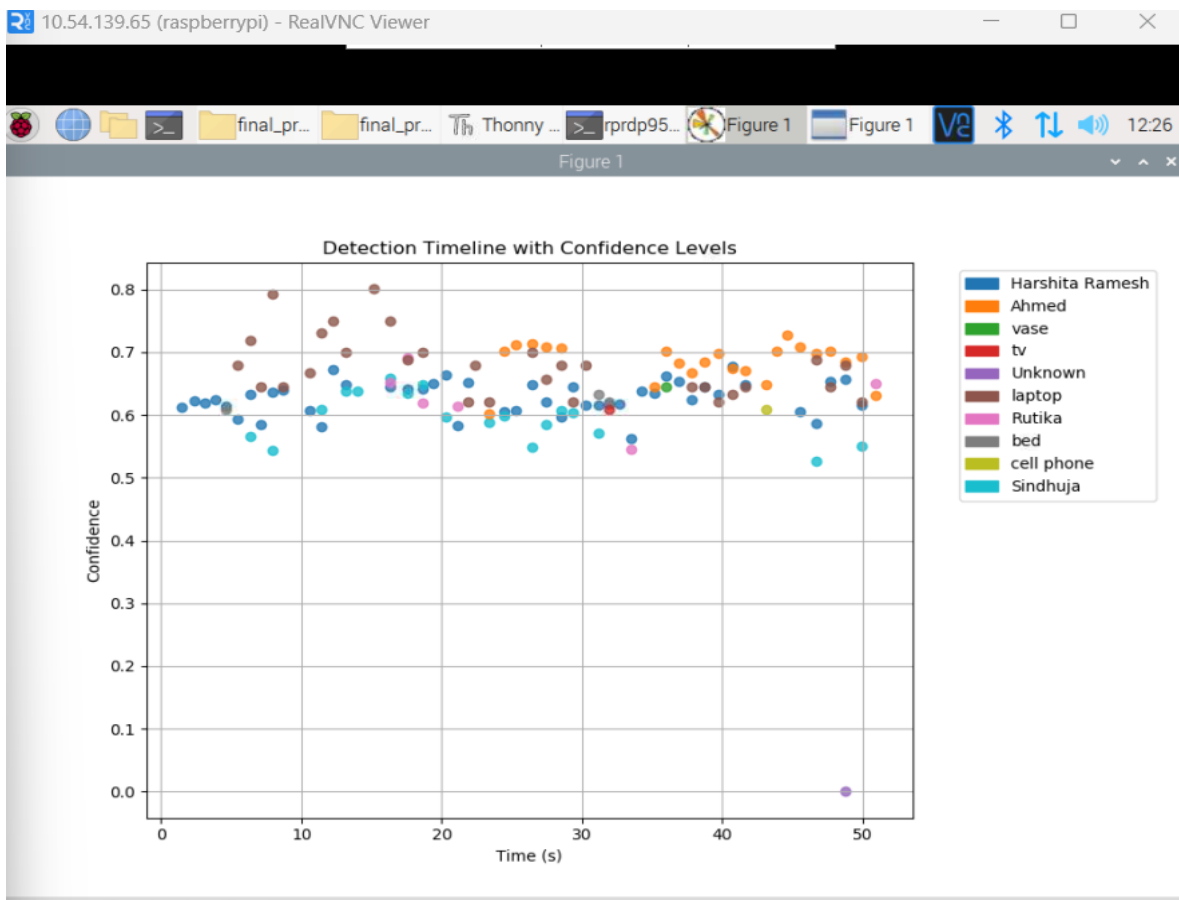


Figure 13: Scenario 3

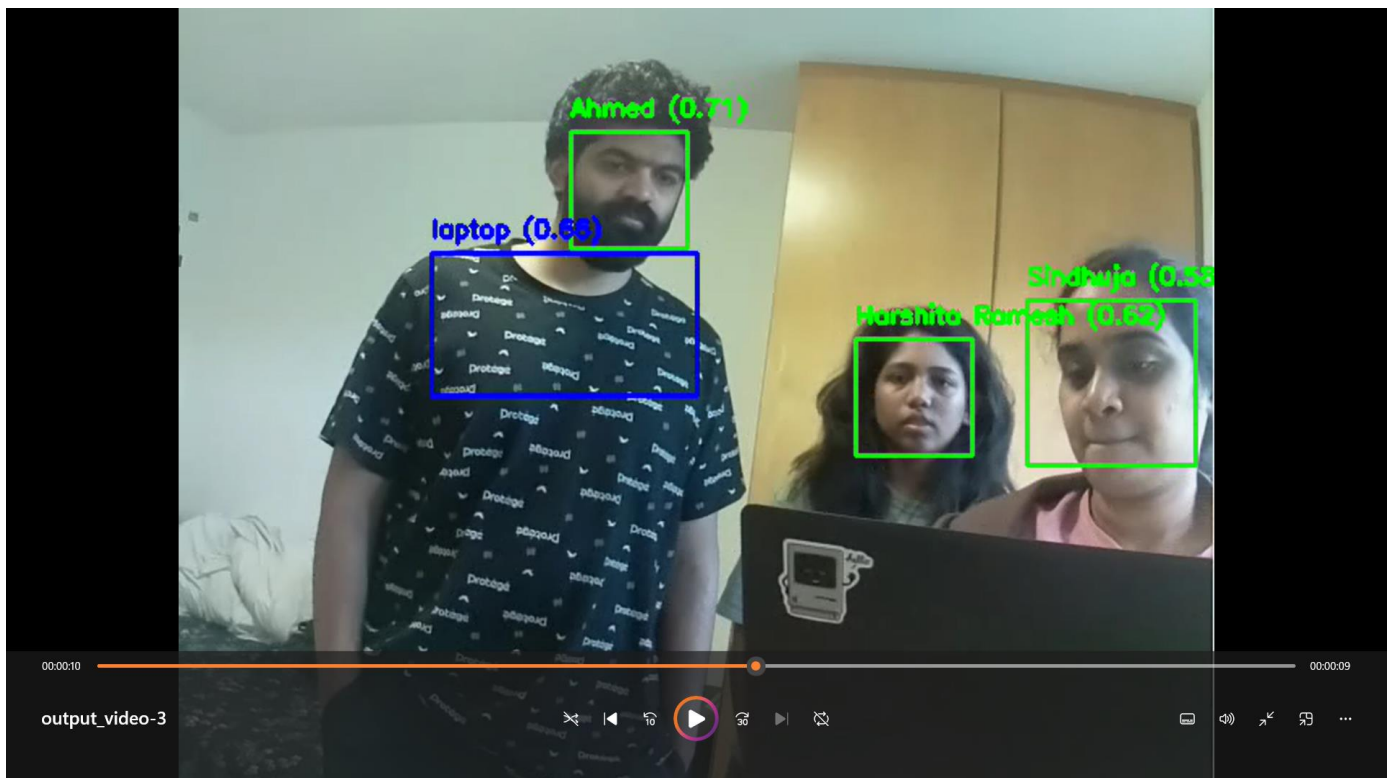


Figure 14: real time detection of scenario-3 from output video

## Conclusion

Overall, the system established a real-time face and object detection architecture with the help of Raspberry Pi 4, TFLite model for object detection and face\_recognition Python library for face recognition. Real-time detection of individuals and objects, displaying their names along with the confidence score in the form of bounding boxes is also accomplished along with its log results over the time. Further, a plot was derived from the output of the system which represents the confidence levels of the detected objects and faces in the given 20-seconds of the recorded video. The validity of the system is strong when faces are clearly illuminated and objects are static or in motion slightly. The performance of the recognition was highly comparable to the images of the person in the dataset set and the confidence level was generally above 60 to 70 percent most of the time. The core functionality was achieved, including identifying individuals and objects in real-time, displaying names and confidence scores with bounding boxes, and logging these results over time. Additionally, the system generated a plot showing the confidence levels of detected objects and faces over the duration of the 20-second video. The validity of the system is strong for scenarios where the faces and objects are well-lit, clearly visible, and there is minimal motion. The recognition accuracy was high for individuals whose images were included in the dataset, with confidence levels typically exceeding 60-70% almost all the time. Still, there were several unexpected situations in which the system was unable to recognize the object, like in a poorly illuminated room or when it was partially hidden. The TFLite object detection model had moderate accuracy on frequently detected objects, but the accuracy was low on low-confidence detections and if objects dominated the frame.

## References

- Patrick Denny. (2024, october). *Computer vision systems- Lab 3 Material*. Retrieved from CE6023-: Facial Recognition System [https://learn.ul.ie/content/enforced/45572-CE6023\\_SEM1\\_2024\\_5/Labs/Lab3\\_FacialRecognition/LabEx3\\_CE6023\\_FacialRecognition.pdf](https://learn.ul.ie/content/enforced/45572-CE6023_SEM1_2024_5/Labs/Lab3_FacialRecognition/LabEx3_CE6023_FacialRecognition.pdf)
- Patrick Denny. (2024,November). *Computer vision systems- Lab 5 Material*. Retrieved from CE6023-: TensorFlow and JPEG [https://learn.ul.ie/content/enforced/45572-CE6023\\_SEM1\\_2024\\_5/Labs/Lab5\\_TensorFlowJPEG/LabEx5\\_CE6023\\_TensorFlowJPEG\\_Rev1.pdf](https://learn.ul.ie/content/enforced/45572-CE6023_SEM1_2024_5/Labs/Lab5_TensorFlowJPEG/LabEx5_CE6023_TensorFlowJPEG_Rev1.pdf)