

LABOTORY 6: PYSERIAL SCRIPTS AND CODEPROJECT.AI SERVER

Learning Outcomes

By the end of this laboratory, student should be able to

- Use PySerial Scripts to send AT commands to switch modem protocols in windows and raspberry pi
- Use MBIM protocol to dial up 5G Hat
- Install CodeProject.AI server on any machine to do facial registration and recognition with laptop's webcam

Activities

- Run python scripts to send AT commands to the Quectel RM502Q-AE modem in windows and raspberry pi
- Obtained the wwan0 public IP address after the Raspberry Pi dial up to 5G
- Install CodeProject.AI server on windows and execute python scripts to register and recognize faces with laptop's webcam

Equipment

- Window OS laptop with webcam
- Raspberry Pi 4 Model B
- Quectel RM502Q-AE 5G module
- Waveshare RM502Q-AE 5G Hat
- SD Card containing the Raspberry Pi OS
- USB-C 5V Power Adapter
- Sim card

1. PySerial Python Scripts

In Lab 1 and Lab 3, you have already learnt how to send AT commands to 5G Hat with Windows's QCOM and Raspberry Pi's Minicom. Now, instead of sending AT commands one at a time, you can use the given python scripts which will automatically send all the required AT-Commands at once to the RM502Q-AE 5G Hat.

Windows OS PySerial Script

1. Remove USB connector and connect 5G Hat to your laptop via USB cable. Attach antennas.
2. Download and open the **Lab6_Code** folder in VScode.
3. Setup and launch a virtual environment with a python interpreter.
4. Pip install pyserial.
pip install pyserial
5. Open AT_command.py script and change **COM** port number. Run the script:
.\venv\Scripts\python AT_command.py

This script sends AT commands to the RM502Q-AE module to dial up 5G internet access in Windows using the COM port for serial communication.

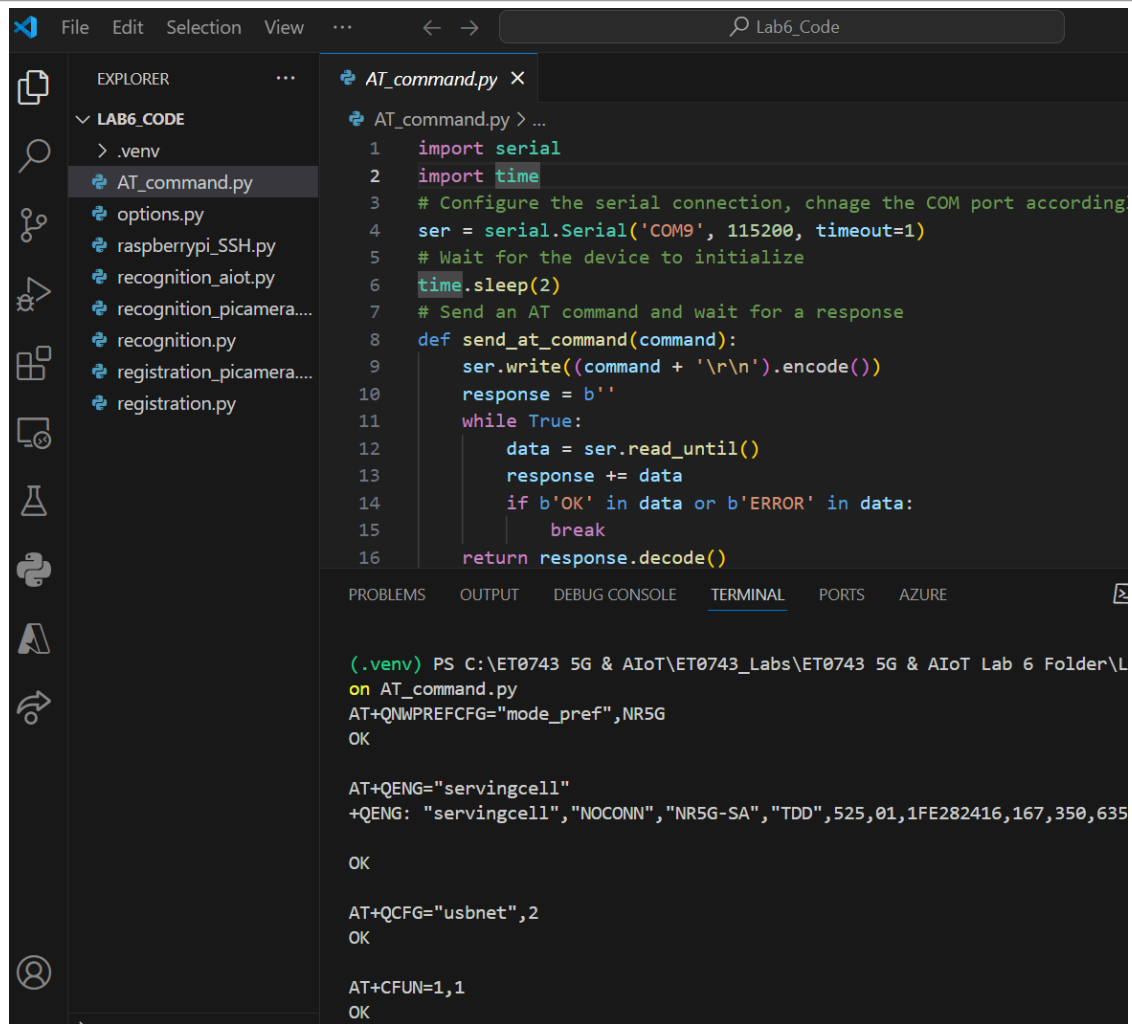
```
import serial
import time
# Configure the serial connection, change the COM port accordingly
ser = serial.Serial('COM9', 115200, timeout=1)
# Wait for the device to initialize
time.sleep(2)
# Send an AT command and wait for a response
def send_at_command(command):
    ser.write((command + '\r\n').encode())
    response = b''
    while True:
        data = ser.read_until()
        response += data
        if b'OK' in data or b'ERROR' in data:
            break
    return response.decode()
# AT Commands usage
response = send_at_command('AT+QNWPRECFG="mode_pref",NR5G')#Force to 5G SA
print(response)
response = send_at_command('AT+QENG="servingcell"')#Query servingcell
print(response)
response = send_at_command('AT+QCFG="usbnet",2') # Switch to MBIM mode
print(response)
response = send_at_command('AT+CFUN=1,1')#Save and reset module
print(response)
# Close the serial connection
ser.close()
```

Pyserial Python Code Explanation:

- **import serial**
This imports the serial module, which provides support for serial connections in Python.
- **import time**
This imports the time module, which is used for time-related functions, such as sleep.
- **ser = serial.Serial('COM9', 115200, timeout=1)**
This line initializes the serial connection. It specifies the AT COM port (COM9 in this case), the baud rate (115200), and the timeout (1 second).
- **time.sleep(2)**
This line waits for 2 seconds, allowing the device connected to the serial port to initialize.
- **def send_at_command(command)**
This defines a function named send_at_command that takes a single argument command, which represents the AT command to be sent.
- **ser.write((command + '\r\n').encode())**
This line sends the AT command to the device. It encodes the command as bytes and writes it to the serial port.
- **response = b''**
This initializes an empty byte string to store the response from the device.
- **while True**
This starts an infinite loop to read the response from the device.
- **data = ser.read_until()**
This line reads data from the serial port until a newline character is encountered (\n).
- **response += data**
This appends the data read from the serial port to the response byte string.
- **if b'OK' in data or b'ERROR' in data**
This checks if the response contains either "OK" or "ERROR". If either of them is found, it breaks out of the loop.
- **return response.decode()**
This returns the response from the device after decoding it from bytes to a string.
- **response = send_at_command('AT+QNWPRECFG="mode_pref",NR5G')**
This sends an AT command to force the device to use 5G Standalone (5G SA) mode and prints the response.
- **response = send_at_command('AT+QENG="servingcell")**
This sends an AT command to query the serving cell and prints the response.
- **response = send_at_command('AT+QCFG="usbnet",2')**
This sends an AT command to switch to MBIM mode and prints the response.
- **response = send_at_command('AT+CFUN=1,1')**
This sends an AT command to save and reset the module and prints the response.
- **ser.close()**
This closes the serial connection once all the commands have been sent and responses received.

ET0743 - 5G & AIoT APPLICATIONS

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING, SINGAPORE POLYTECHNIC



The screenshot shows a Visual Studio Code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'LAB6_CODE' with several Python files. The active file is 'AT_command.py'. The code in the editor is a Python script that uses the 'serial' library to communicate with an AT module. It configures the serial connection, waits for initialization, and then sends three AT commands: 'AT+QNWPRECFG="mode_pref",NR5G', 'AT+QENG="servingcell"', and 'AT+QCFG="usbnet",2'. The terminal shows the output of these commands, including 'OK' and the response from the AT module.

```
1 import serial
2 import time
3 # Configure the serial connection, chnagne the COM port according
4 ser = serial.Serial('COM9', 115200, timeout=1)
5 # Wait for the device to initialize
6 time.sleep(2)
7 # Send an AT command and wait for a response
8 def send_at_command(command):
9     ser.write((command + '\r\n').encode())
10    response = b''
11    while True:
12        data = ser.read_until()
13        response += data
14        if b'OK' in data or b'ERROR' in data:
15            break
16    return response.decode()
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS AZURE

```
(.venv) PS C:\ET0743 5G & AIoT\ET0743_Labs\ET0743 5G & AIoT Lab 6 Folder\L
on AT_command.py
AT+QNWPRECFG="mode_pref",NR5G
OK

AT+QENG="servingcell"
+QENG: "servingcell","NOCONN","NR5G-SA","TDD",525,01,1FE282416,167,350,635
OK

AT+QCFG="usbnet",2
OK

AT+CFUN=1,1
OK
```

****You may still need to update driver to obtain cellular network on your laptop**

6. Update driver and check if your laptop has cellular connection.

Question 1:

How to check COM port on windows laptop?

Raspberry Pi OS PySerial Script

The second pyserial script is almost the same as the previous one, it sends AT commands to the RM502Q-AE module to dial up 5G internet access in Raspberry Pi using the USB serial port(ttyUSB) for serial communication. The only change is the port, /dev/ttyUSB2 is used instead of COM port.

****This script can only be run inside the Raspberry Pi; you don't have to change anything inside the script.**

```
import serial
import time
# Configure the serial connection
ser = serial.Serial('/dev/ttyUSB2', 115200, timeout=1)
# Wait for the device to initialize
time.sleep(2)
# Send an AT command and wait for a response
def send_at_command(command):
    ser.write((command + '\r\n').encode())
    response = b''
    while True:
        data = ser.read_until()
        response += data
        if b'OK' in data or b'ERROR' in data:
            break
    return response.decode()

# AT Commands usage
response = send_at_command('AT+QNWPRECFG="mode_pref",NR5G')#Force to 5G SA
print(response)

response = send_at_command('AT+QENG="servingcell")#Query servingcell
print(response)

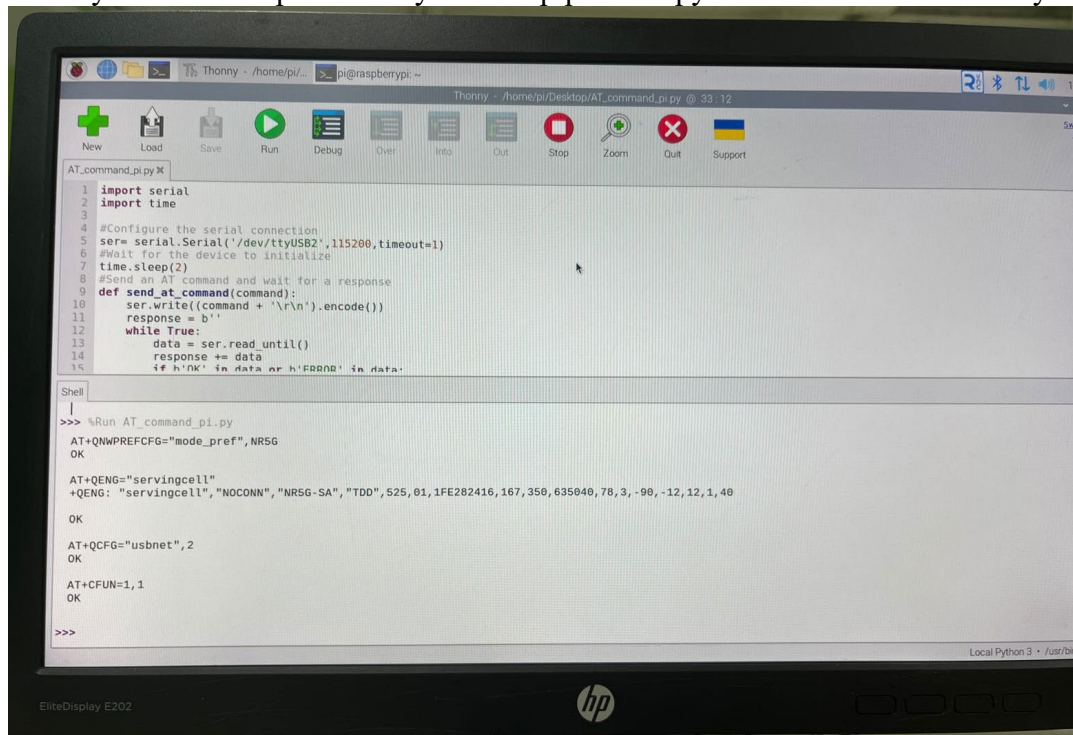
response = send_at_command('AT+QCFG="usbnet",2') # Switch to MBIM mode
print(response)

response = send_at_command('AT+CFUN=1,1')#Save and reset module
print(response)

# Close the serial connection
ser.close()
```

7. Connect monitor, keyboard and mouse to raspberry pi. Power on monitor and pi.
8. Connect the Raspberry Pi and 5G Hat with the USB connector, attach the antennas.
9. Connect to "SPStudent" Wi-Fi Network (refer to Lab 3).

10. Next, run these several commands in a terminal to install MBIM interface.
sudo apt purge modemmanager -y
sudo apt purge network-manager -y
sudo apt update && sudo apt upgrade
sudo apt install libglib2.0-dev libmbim-utils libmbim-glib-dev
sudo reboot
11. After rebooting, right-click on AT_command_pi.py pyserial script and open with Thonny. Run the script. You may have to pip install pyserial if it is not installed yet.



```

1 import serial
2 import time
3
4 #Configure the serial connection
5 ser= serial.Serial('/dev/ttyUSB2',115200,timeout=1)
6 #Wait for the device to initialize
7 time.sleep(2)
8 #Send an AT command and wait for a response
9 def send_at_command(command):
10     ser.write((command + '\r\n').encode())
11     response = b''
12     while True:
13         data = ser.read_until()
14         response += data
15         if b'OK' in data or b'ERROR' in data:
16             break
17
18 #Main function
19 def main():
20     send_at_command('AT')
21     send_at_command('AT+QWPRECFG="mode_pref",NR5G')
22     send_at_command('AT+QENG="servingcell"')
23     send_at_command('AT+QCFG="usbnets",2')
24     send_at_command('AT+CFUN=1,1')
25
26 if __name__ == '__main__':
27     main()

```

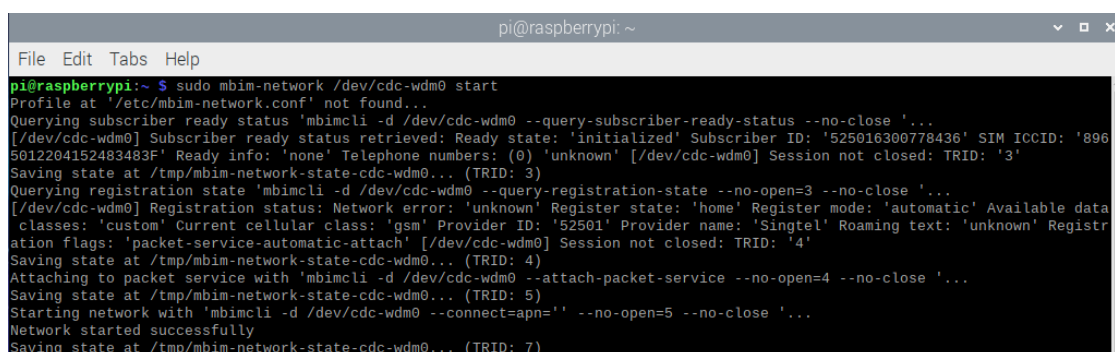
```

Shell
>>> %Run AT_command_pi.py
AT+QWPRECFG="mode_pref",NR5G
OK
AT+QENG="servingcell"
+QENG: "servingcell", "NDCONN", "NR5G-SA", "TDO", 525, 61, 1FE282416, 167, 356, 635040, 78, 3, -90, -12, 12, 1, 40
OK
AT+QCFG="usbnets",2
OK
AT+CFUN=1,1
OK
>>>

```

12. Wait **10** seconds for the 5G Hat to reboot, then run this command to start the MBIM network.

sudo mbim-network /dev/cdc-wdm0 start



```

pi@raspberrypi: ~
File Edit Tabs Help

pi@raspberrypi:~ $ sudo mbim-network /dev/cdc-wdm0 start
Profile at '/etc/mbim-network.conf' not found...
Querying subscriber ready status 'mbimcli -d /dev/cdc-wdm0 --query-subscriber-ready-status --no-close ...'
[/dev/cdc-wdm0] Subscriber ready status retrieved: Ready state: 'initialized' Subscriber ID: '525016300778436' SIM ICCID: '896
5012204152483483F' Ready info: 'none' Telephone numbers: (0) 'unknown' [/dev/cdc-wdm0] Session not closed: TRID: '3'
Saving state at /tmp/mbim-network-state-cdc-wdm0... (TRID: 3)
Querying registration state 'mbimcli -d /dev/cdc-wdm0 --query-registration-state --no-open=3 --no-close ...'
[/dev/cdc-wdm0] Registration status: Network error: 'unknown' Register state: 'home' Register mode: 'automatic' Available data
classes: 'custom' Current cellular class: 'gsm' Provider ID: '52501' Provider name: 'Singtel' Roaming text: 'unknown' Registr
ation flags: 'packet-service-automatic-attach' [/dev/cdc-wdm0] Session not closed: TRID: '4'
Saving state at /tmp/mbim-network-state-cdc-wdm0... (TRID: 4)
Attaching to packet service with 'mbimcli -d /dev/cdc-wdm0 --attach-packet-service --no-open=4 --no-close ...'
Saving state at /tmp/mbim-network-state-cdc-wdm0... (TRID: 5)
Starting network with 'mbimcli -d /dev/cdc-wdm0 --connect=apn=' --no-open=5 --no-close ...'
Network started successfully
Saving state at /tmp/mbim-network-state-cdc-wdm0... (TRID: 7)

```

13. Next, use this command to test 5G network:

ping google.com -I wwan0

```
pi@raspberrypi:~$ ping google.com -I wwan0
PING google.com(sh-in-f102.1e100.net (2404:6800:4003:c1c::66)) from
s
64 bytes from sh-in-f102.1e100.net (2404:6800:4003:c1c::66): icmp_
64 bytes from sh-in-f102.1e100.net (2404:6800:4003:c1c::66): icmp_
64 bytes from sh-in-f102.1e100.net (2404:6800:4003:c1c::66): icmp_
64 bytes from sh-in-f102.1e100.net (2404:6800:4003:c1c::66): icmp_
64 bytes from sh-in-f102.1e100.net (2404:6800:4003:c1c::66): icmp_
^C
--- google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5001ms
rtt min/avg/max/mdev = 20.728/40.159/47.478/9.183 ms
```

14. The pyserial script and the command `sudo mbim-network /dev/cdc-wdm0` start are needed to get the **public IP address** to show up at `wwan0` (Wide-Area Network) interface. If you cannot see it, you will need to run the script and start mbim network command again. The `wwan0` interface has 2 IP addresses, but only one of them is the public IP address which usually starts with 100.

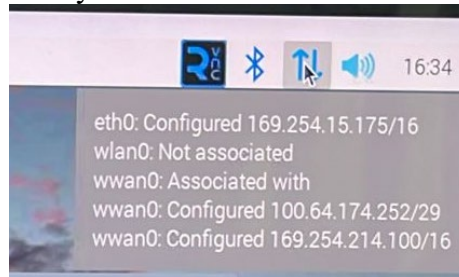


Figure 1: `wwan0`'s public IP address 100.64.174.252

Write down the **public IP address** of `wwan0` (usually starts with 100):

The public IP address is important because it is used in python IP scripts, pi-camera facial recognition script. During the mini project, we are mostly going to be using the public IP address. The reason is because other computers on the same 5G network use the public IP address to communicate with the devices on the network.

2. CodeProject.AI server Introduction

CodeProject.AI Server is an Open-Source Artificial Intelligence server that can be installed locally and self-hosted on any platform and in any programming language. It operates without the need for off-device or out-of-network data transfer and eliminates the complexities of handling dependencies. Moreover, it is compatible with diverse platforms and programming languages, offering the option to run as a Windows Service or a Docker container, ensuring fast and free AI capabilities without compromising data security.

CodeProject.AI Installation (Windows)

1. Go to <https://www.codeproject.com/Articles/5322557/CodeProject-AI-Server-AI-the-easy-way> and download the latest **windows x64 installer**.
2. When running the installer, untick all object detection installation and tick face processing under face recognition.
3. Once the setup is complete, this webpage below will pop up. If it does not, type in localhost:32168 in a web browser to access the dashboard. Wait for it to finish installing.

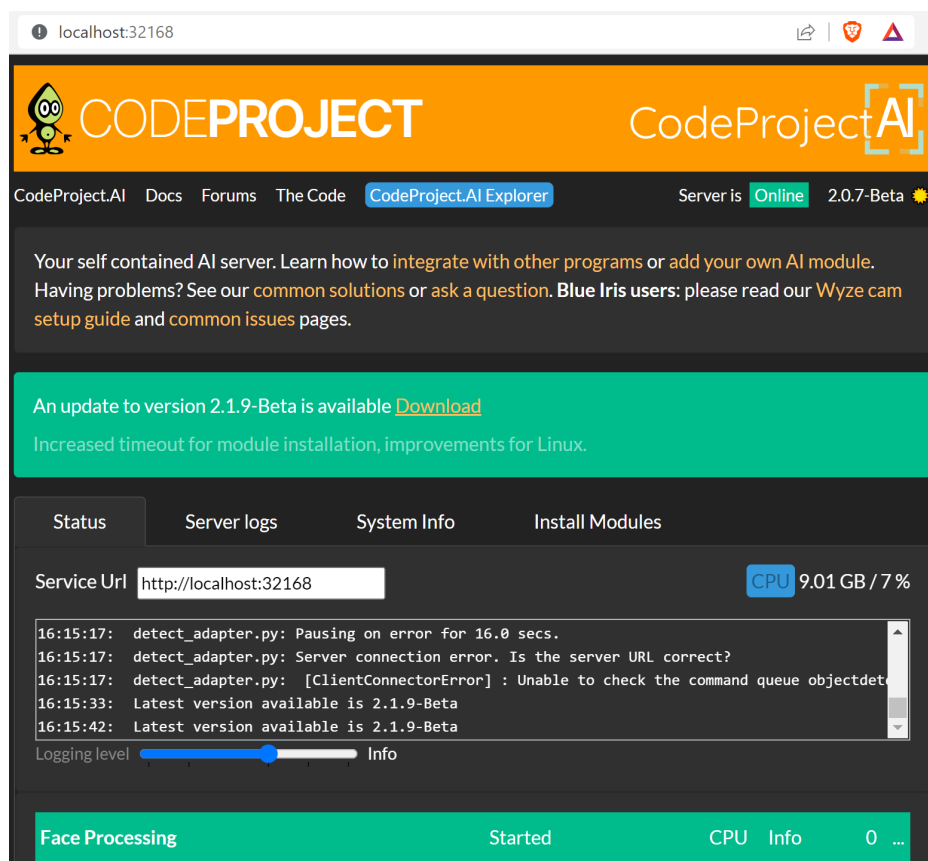


Figure 2: CodeProject.AI server Dashboard

A. CodeProject.AI Functions

- Click on the “CodeProject.AI Explorer” reveals four functions: Face detection, comparison, registration, and recognition.

We are only going to focus on these 3 functions: Face Detection, Face Registration and Face Recognition.

Face Detection

The first function is the Face Detection where the server will try to detect a face (human and animals) within an image.

- Take a photo of your face and under the Face Detection, click “Choose files” and upload your face. Then, click “Detect”.

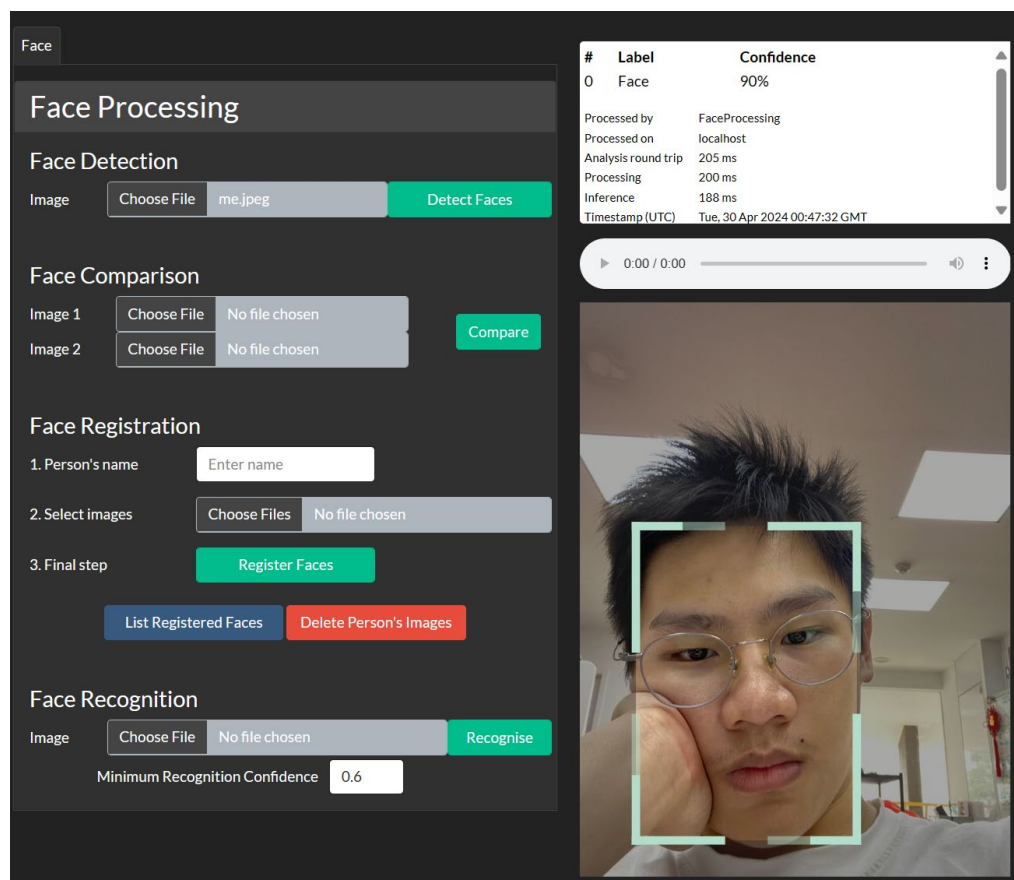


Figure 3: CodeProject.AI's Face Detection function

What you see in the right white box is the results, you will see the label and confidence level. 90% confidence level means the server can a human face in that image 90% of the time because your face may not be clear enough in the image uploaded. If you upload an image that is not clear enough, you will get no predictions returned which means the server cannot detect a face in that image.

Face Registration

Next is the face registration function. It registers one image for a user for recognition. This **trains** the face recognition model and allows the face recognition function to report back a userId based on an image you supply that may or may not contain that user's face.

By uploading an image of our face, the server will only extract unique facial features like mouth, nose, eyes and save it in vectors in a database for future reference. Take note that this function is using the server's Face Detection function to detect a face in an image first before the server can register the face. CodeProject.AI server runs on a local server which means whatever faces you registered on the server will only remain in the laptop which the server is installed. So, if you were to install a new server on another laptop, all the registered faces will be gone.

6. Input your name.
7. Input a picture of your face as in jpeg format.
8. Select "Register Faces".

The screenshot shows a web application titled 'Face Registration'. It has a dark grey background with white text. The form consists of three numbered steps. Step 1, 'Person's name', has a text input field containing 'Sheng Kai'. Step 2, 'Select images', has a 'Choose Files' button and a 'No file chosen' status. Step 3, 'Final step', has a large green 'Register Faces' button. Below the steps, there are two buttons: a blue 'List Registered Faces' button and a red 'Delete Person's Images' button.

Figure 4: CodeProject.AI Face Registration function

You can list and remove registered faces saved in the server using the blue and red buttons.

For your information, the facial features are stored in a database file in a hidden folder called ProgramData (e.g. C:\ProgramData\CodeProject\AI) and the database file default name is called faceembedding. If you have installed SQLite3 extensions in VSCode, you can view the database file in SQLite editor. Unfortunately, you cannot do anything like delete or add userid to the database because you do not have access to the database file. Therefore, you need to use the CodeProject.AI Dashboard to delete a registered person's face.

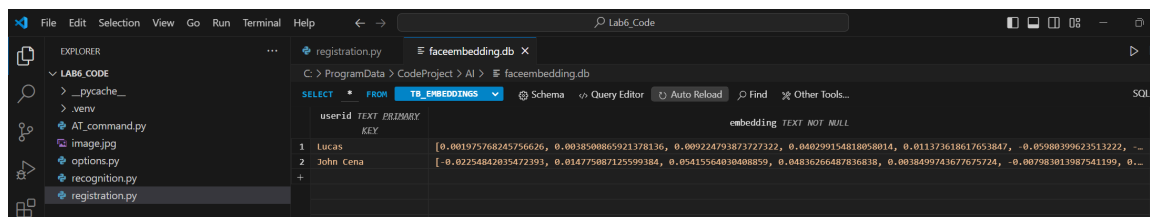


Figure 5: faceembedding.db file in SQLite editor in VSCode

Face Recognition

Next is the face recognition function. It recognizes all faces in an image and returns the userId and coordinates of each face in the image. If a new (unregistered) face is detected, then no userid for that face will be returned. By uploading an image of our face, the server will perform facial recognition based on the faces saved in the server's database using the face registration function. The confidence level means the level of accuracy which the server can recognize your face. For facial recognition function to recognize your face more accurately, you need to register many images of your face.

9. Upload your face and select "Recognise". You can adjust the minimum recognition confidence level from **0.0 - 1.0** where higher value will make the face recognition function harder to recognize your face but more accurate.

Face Processing

Face Detection

Image: No file chosen

Face Comparison

Image 1: No file chosen

Image 2: No file chosen

Face Registration

1. Person's name:

2. Select images: No file chosen

3. Final step:

Face Recognition

Image: me.jpeg

Minimum Recognition Confidence:

#	Name	Confidence
0	Lucas	100%

Processed by: FaceProcessing
 Processed on: localhost
 Analysis round trip: 1287 ms
 Processing: 1280 ms
 Inference: 1181 ms
 Timestamp (UTC): Tue, 30 Apr 2024 01:38:12 GMT

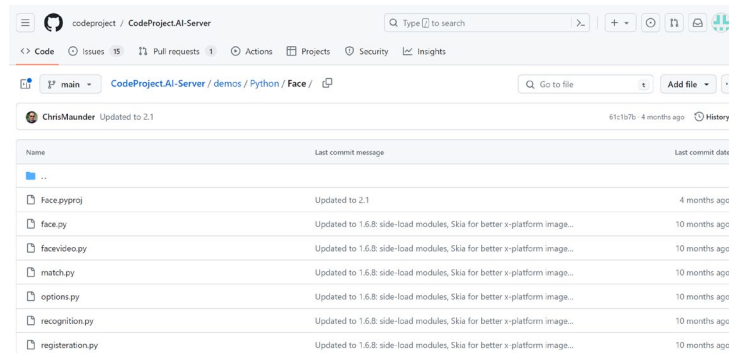
0.00 / 0.00

Figure 6: CodeProject.AI's Face Recognition function

B. CodeProject.AI Facial Registration/Recognition Scripts (webcam)

CodeProject.AI can also make use of python scripts to perform the functions stated above. The available Python script functions include face registration and face recognition. The base Python scripts can be found in the CodeProject.AI GitHub repository under the 'demos' directory in the 'python face' folder. The GitHub repository can be found at:

<https://github.com/codeproject/CodeProject.AI-Server>



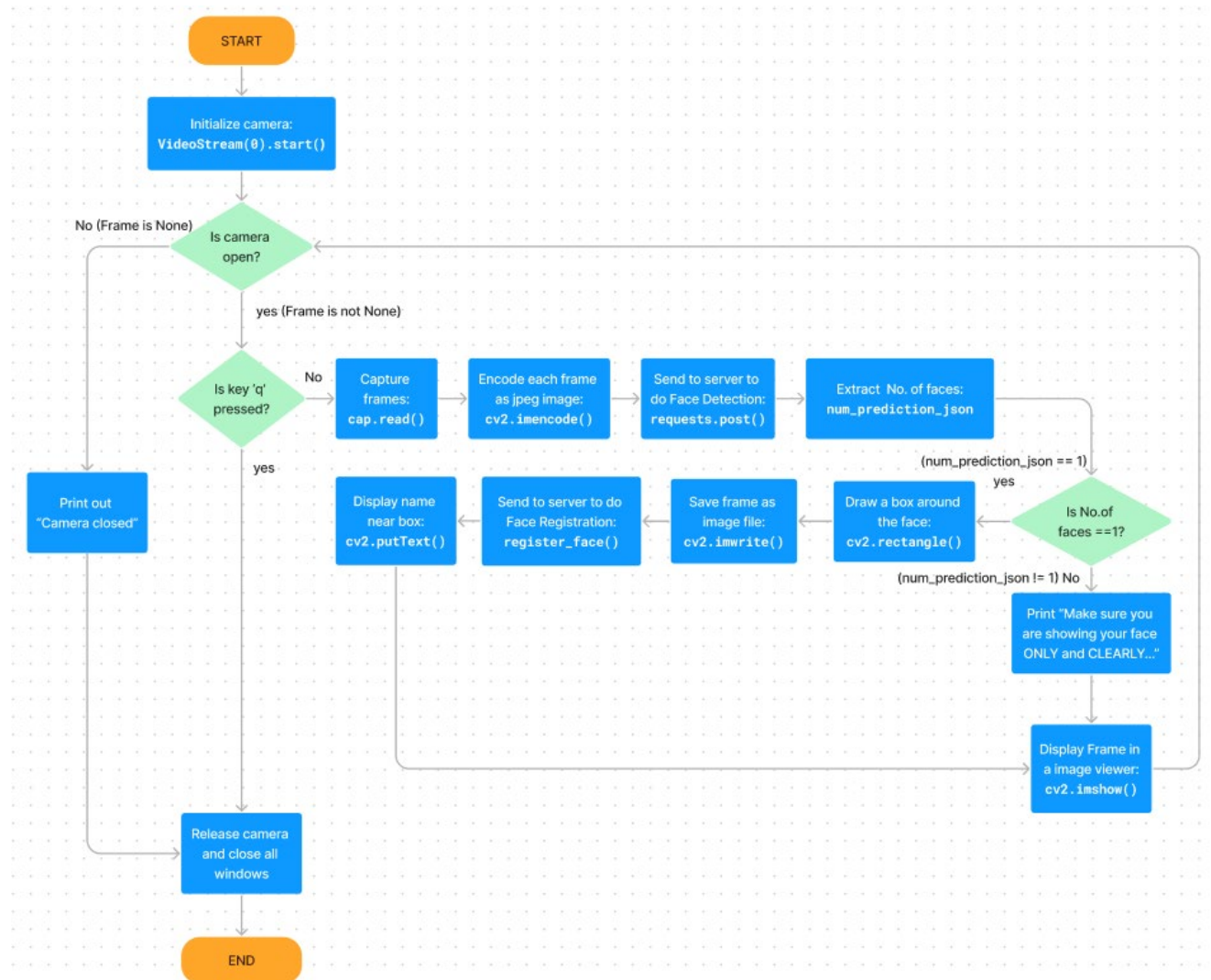
Name	Last commit message	Last commit date
..		
Face.pyproj	Updated to 2.1	4 months ago
face.py	Updated to 1.6.8: side-load modules, Skia for better x-platform image...	10 months ago
facevideo.py	Updated to 1.6.8: side-load modules, Skia for better x-platform image...	10 months ago
match.py	Updated to 1.6.8: side-load modules, Skia for better x-platform image...	10 months ago
options.py	Updated to 1.6.8: side-load modules, Skia for better x-platform image...	10 months ago
recognition.py	Updated to 1.6.8: side-load modules, Skia for better x-platform image...	10 months ago
registration.py	Updated to 1.6.8: side-load modules, Skia for better x-platform image...	10 months ago

Figure 7: CodeProject.AI Github Repository

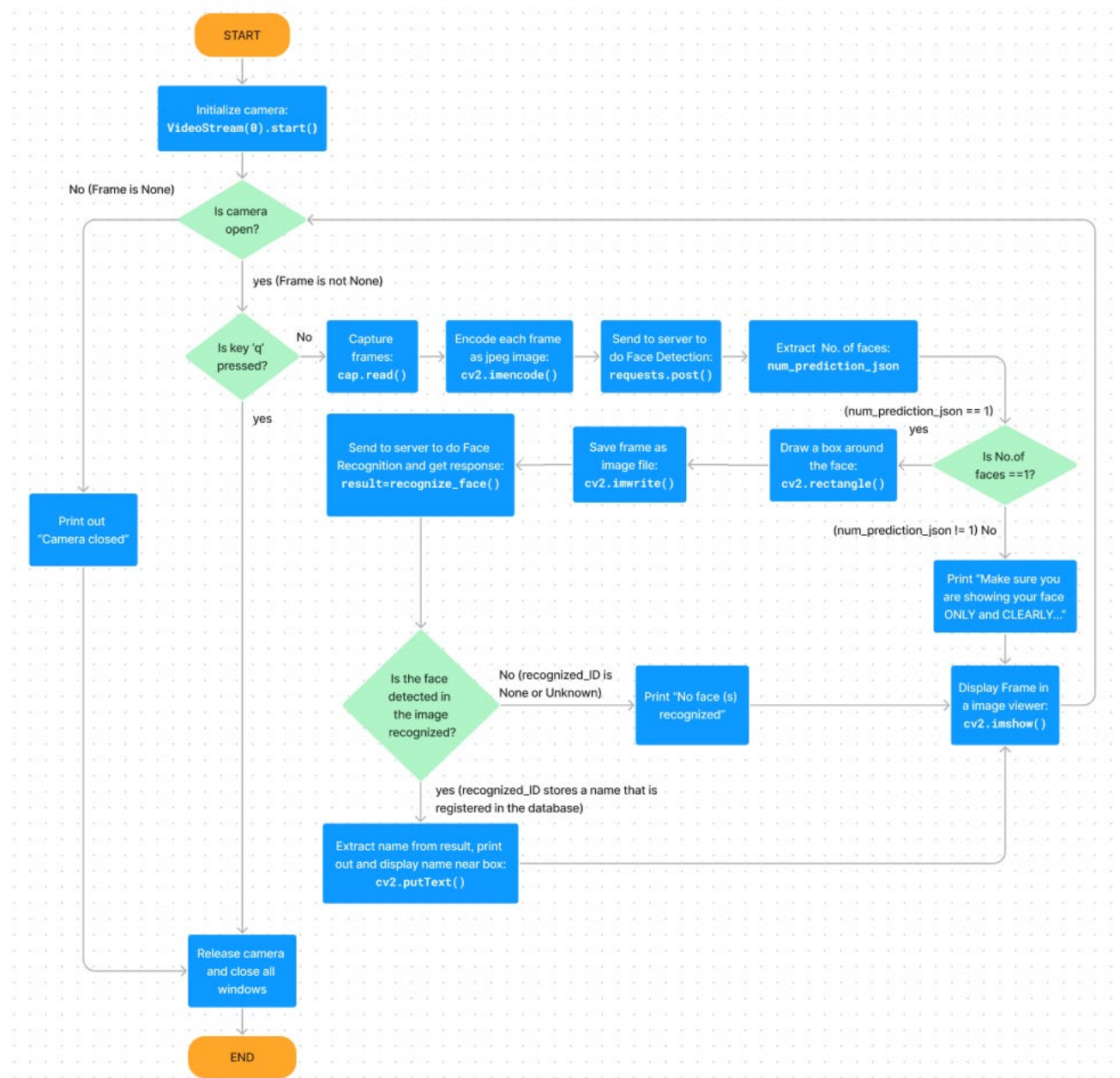
The most important script is options.py as it sets up the local host and image directory. It needs to be placed in the same folder as the other two scripts, registration.py and recognition.py. registration.py and recognition.py will capture frames from your laptop's webcam. Therefore, it is important that your laptop have a webcam installed.

The flowcharts of both registration.py and recognition.py are quite similar except when a face is detected, a different function register_face() or recognize_face() will be called and do either face registration or recognition.

Flowchart of registration.py



Flowchart of recognition.py



12. Go back to VScode, make sure you open the correct folder and are in the virtual environment already.
13. Pip installs opencv, requests and imutils.
pip install opencv-python
pip install requests
pip install imutils
14. Before you run registration.py, replace the variable name with your OWN name. Run registration.py:
.\venv\Scripts\python registration.py

```
# Import necessary modules and packages
import os
import cv2
import requests
import datetime
from options import Options
from imutils.video import VideoStream

# Create an instance of the Options class
opts = Options()

#Put your name
name = "Lucas"

# Define function to register a face !!!
def register_face(img_path, user_id):
    # This function sends the face image and the user id to the server to
    register !!!

    # Build the file path to the image
    filepath = os.path.join(opts.imageDir, img_path)

    # Read the image data as bytes
    image_data = open(filepath, "rb").read()

    try:
        # Send the image & user_id to the server to register !!!
        response = requests.post(opts.endpoint("vision/face/register"),
                                files={"image": image_data},
                                data={"userid": user_id}).json()

        # Print the response received from the server !!!
        print(f"Registration response: {response}")
    except requests.exceptions.RequestException as e:
        raise SystemExit(e)

# Define the main function
def main():
```

```
# Captures the video frames from the camera,
# detects the faces in the frames,
# and sends them to the server with user ids for registration. !!!

# Create an instance of the Options class
opts = Options()

# Initialise capturing video from the default camera
# 0 for default camera, 1 if you have installed third-party webcam apps
cap = VideoStream(0).start()

# Initialize variables to keep track of frame count, predictions, and
frame skipping
frame_index = 0
predictions = {}
skip_frame = 5

# Begin the main loop to process frames from the camera
while True:
    # Check for the 'q' key to exit the program
    if cv2.waitKey(1) & 0xFF == ord('q'):
        print("Stop capturing...")
        break

    # Capture a frame from the camera
    frame = cap.read()
    if frame is None:
        print("Camera closed")
        break

    # Increment the frame count
    frame_index += 1

    # Skip some frames to reduce the processing load
    if skip_frame > 1:
        if frame_index % skip_frame != 0:
            continue

    # Get the current timestamp
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Add the timestamp to the frame
    cv2.putText(frame, timestamp, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 255, 0), 1, cv2.LINE_AA)

    # Encode the frame as a JPEG image
    retval, new_frame = cv2.imencode('.jpg', frame)
```



```

try:
    # Send the frame to the server to detect the face(s) in the frame
    response = requests.post(opts.endpoint("vision/face"),
                              files={"image": new_frame}).json()
except requests.exceptions.RequestException as e:
    raise SystemExit(e)

# Extract the detections from the response
predictions = response['predictions']

# Get the number of predictions (i.e. faces detected)
num_prediction_json = len(predictions)

# Loop over the predictions and draw a rectangle around each face in
the frame
if num_prediction_json == 1:
    for i in range(num_prediction_json):
        blue, green, red = 0, 0, 255
        frame = cv2.rectangle(frame,
                               (predictions[i]['x_min'],
                                predictions[i]['y_min']),
                               (predictions[i]['x_max'],
                                predictions[i]['y_max']),
                               (blue, green, red), 2)
        # Save the frame as an image file named "image.jpg"
        cv2.imwrite('image.jpg', frame)

        # Register the face with the user ID
        register_face("image.jpg", name)

        # Add result right next to the rectangle
        cv2.putText(frame, name, (predictions[i]['x_min'],
                                predictions[i]['y_min']-10), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2,
                                cv2.LINE_AA)
    else:
        print('Make sure you are showing your face ONLY and CLEARLY...')

cv2.imshow('Image Viewer', frame)

# Release the camera and close all windows
cap.stop()
cv2.destroyAllWindows()

# Call the main function if this script is being run directly
if __name__ == "__main__":
    main()

```

registration.py python script running:

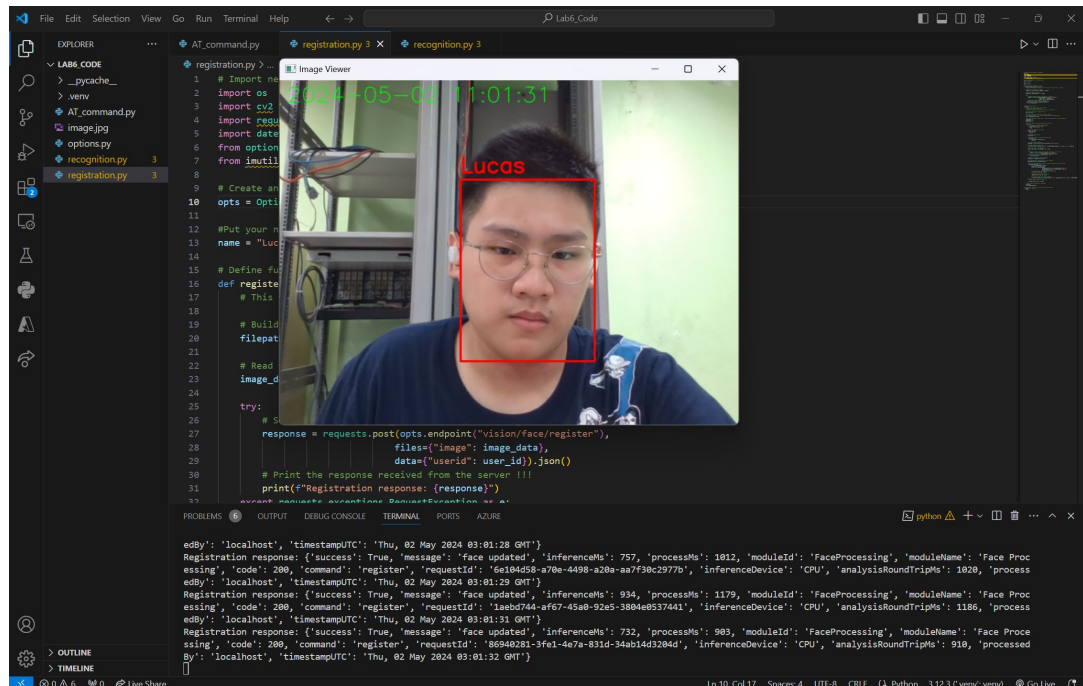


Figure 8: Registration.py running on windows

**If the script does not run, check this line `cap = VideoStream(0).start()`, replace the 0 with 1.

15. Press 'Q' to quit and run recognition.py:
`.\venv\Scripts\python recognition.py`

```
# Import necessary modules and packages
import os
import cv2
import requests
import datetime
from options import Options
from imutils.video import VideoStream

# Create an instance of the Options class
opts = Options()

# Define function to recognize a face !!!
def recognize_face(img_path):
    # This function sends the image to the server to detect the faces in the
    # image
    # and returns the user id(s) of the recognized face(s). !!!
    # Create an instance of the Options class
    opts = Options()

    # Build the file path to the image
    filepath = os.path.join(opts.imageDir, img_path)
```

```
# Read the image data as bytes
image_data = open(filepath, "rb").read()

try:
    # Send the image to the server to recognize the face(s) !!!
    response = requests.post(opts.endpoint("vision/face/recognize"),
                              files={"image": image_data},
                              data={"min_confidence": 0.6}).json()

    return response
except requests.exceptions.RequestException as e:
    raise SystemExit(e)

# Define the main function
def main():
    # Captures the video frames from the camera,
    # detects the faces in the frames,
    # and sends them to the server for recognition. !!!

    # Initialise capturing video from the default camera
    # 0 for default camera, 1 if you have installed third-party webcam apps
    cap = VideoStream(0).start()

    # Initialize variables to keep track of frame count, predictions, and
    # frame skipping
    frame_index = 0
    predictions = {}
    skip_frame = 5

    # Begin the main loop to process frames from the camera
    while True:

        # Check for the 'q' key to exit the program
        if cv2.waitKey(1) & 0xFF == ord('q'):
            print("Stop capturing...")
            break

        # Capture a frame from the camera
        frame = cap.read()
        if frame is None:
            print("Camera closed")
            break

        # Increment the frame count
        frame_index += 1

        # Skip some frames to reduce the processing load
        if skip_frame > 1:
```

```

        if frame_index % skip_frame != 0:
            continue

    # Get the current timestamp
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Add the timestamp to the frame
    cv2.putText(frame, timestamp, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 255, 0), 1, cv2.LINE_AA)

    # Encode the frame in JPEG format
    retval, new_frame = cv2.imencode('.jpg', frame)

    try:
        # Send the frame to the server to detect the face(s) in the frame
        response = requests.post(opts.endpoint("vision/face"),
                                files={"image": new_frame}).json()
    except requests.exceptions.RequestException as e:
        raise SystemExit(e)

    # Extract the predictions from the response
    predictions = response.get('predictions', [])
    # Get the number of predictions (i.e. faces detected)
    num_prediction_json = len(predictions)

    # Loop over the predictions and draw a rectangle around each face in
the frame
    if num_prediction_json == 1:
        for i in range(num_prediction_json):
            blue, green, red = 0, 0, 255
            frame = cv2.rectangle(frame,
                                (predictions[i]['x_min'],
predictions[i]['y_min']),
                                (predictions[i]['x_max'],
predictions[i]['y_max']),
                                (blue, green, red), 2)
            # Save the frame as an image file named "image.jpg"
            cv2.imwrite('image.jpg', frame)

            # Get response
            result = recognize_face("image.jpg")

            # Check if any detected faces are recognized
            if "predictions" in result:
                # Get userid
                for user in result["predictions"]:
                    recognized_ID=user["userid"]
                    #Check if userid is None or unknown

```

```

        if recognized_ID and recognized_ID != "unknown":
            #Print out name of recognized face
            print(f'Recognized as: {user["userid"]}')
            # Add name right next to the rectangle
            cv2.putText(frame, recognized_ID,
(predictions[i]['x_min'], predictions[i]['y_min']-10),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
        else:
            print("No face(s) recognized!!!")

    else:
        print('Make sure you are showing your face ONLY and CLEARLY...')

    cv2.imshow('Image Viewer', frame)

# Release the camera and close all windows
cap.stop()
cv2.destroyAllWindows()

# Call the main function if this script is being run directly
if __name__ == "__main__":
    main()

```

recognition.py python script running:

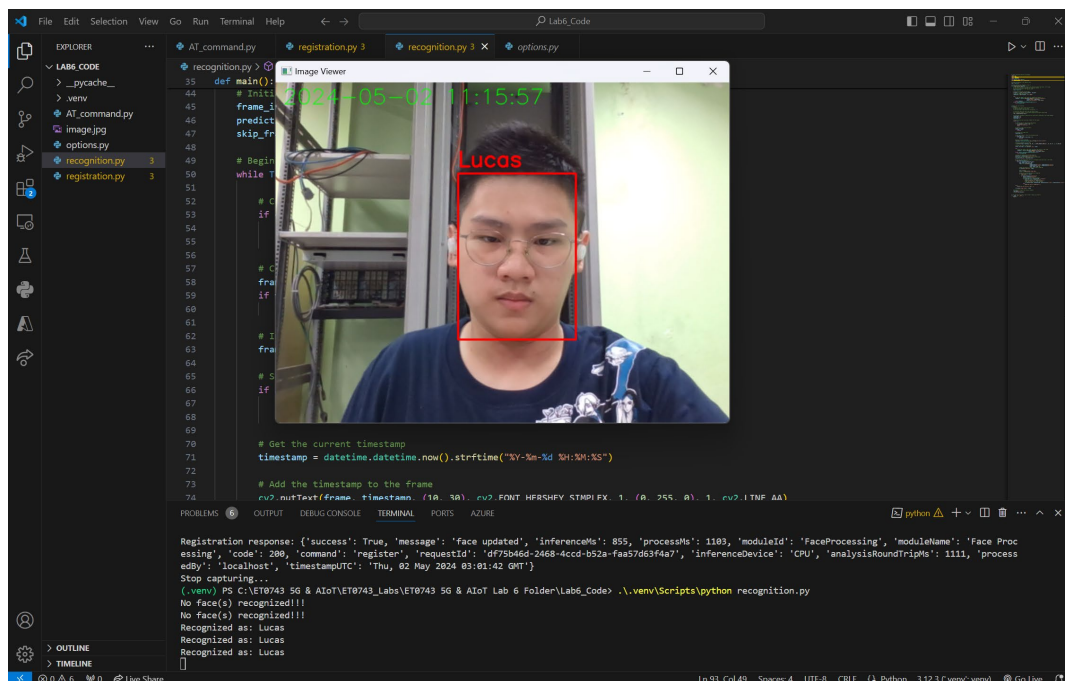


Figure 9: My face is recognized and printed out.

CodeProject.AI server Python Code Explanation:

All the code explanations are already done inside the scripts themselves, however there are still a few lines of code that need more explaining.

- **cap = VideoStream(0).start()**

The VideoStream function from the `imutils.video` module is a convenient utility for capturing frames from a video stream, such as a webcam or an IP camera. It provides a simplified interface for accessing video streams and is often used in computer vision and video processing applications. There are also additional arguments for this function like adjusting resolution and framerate:

cap = VideoStream(src=0, resolution=(640, 480), framerate=30).start()

Do note that the VideoStream function from `imutils.video` can also take in parameters like Real-Time Streaming Protocol (RTSP) or Hyper Text Transfer Protocol (HTTP) URLs, which are commonly used for streaming video from IP cameras, Pi Camera and other networked sources. If your stream the pi camera with RTSP, you can use the generated RTSP URL like this:

cap = VideoStream("rtsp://100.126.0.9:8080/").start()

The `imutils` python library heavily depends on `opencv` as it is just a simpler, beginner-friendly version of `opencv`. However, `imutils VideoStream` function serves different purposes as compared to `opencv VideoCapture` function. VideoStream from `imutils.video` provides a simplified interface for accessing video streams, including support for threading and simplified frame grabbing. While `cv2.VideoCapture` from OpenCV (`cv2`) is a class that provides access to video capturing devices, such as cameras and video files. It offers more extensive functionality, including the ability to set properties like frame width, height, and framerate, as well as more control over video capture settings. The reason why `imutils VideoStream` function is used to capture the stream is because it already has built-in threading while `opencv VideoCapture` does not, therefore, there is a decrease in latency (less lagging) using `imutils VideoStream`.

- **response = requests.post(opts.endpoint("vision/face/recognize"), files={"image": image_data}, data={"min_confidence": 0.8}).json()**

This sends an image to the CodeProject.AI server endpoint, along with some additional parameters like minimum confidence level, and expects a JSON response containing recognition results. This part of the line below allows you to adjust the minimum confidence level required for the facial recognition algorithm to recognize you:

data={"min_confidence": 0.8}

By setting a higher minimum confidence level, you can ensure that only predictions with a high level of confidence are accepted, potentially reducing false positives at the expense of possibly missing some correct predictions. Conversely, setting a lower minimum confidence level may result in more predictions being accepted, but also increases the risk of accepting incorrect or unreliable predictions. Adjusting this parameter allows you to fine-tune the balance between precision and recall in your recognition system, depending on your specific requirements and use case.

Troubleshooting:

- Make sure that your laptop has a camera.
- If you have installed a third-party camera on your computer, replace “cap = cv2.VideoCapture(0)” with “cap=cv2.VideoCapture(1)” instead.
- Inside CodeProject.AI server Dashboard, go to the “Install Modules” tab. Under Face Recognition, uninstall and reinstall Face Processing.
- Uninstall and reinstall CodeProject.AI server.

Question 2:

What are the lines of code that can increase or decrease the accuracy of the face recognition function by adjusting the minimum confidence level?

```
response = requests.post(opts.endpoint("vision/face/recognize"),  
files={"image": image_data}, data={"min_confidence": 0.6}).json()
```

Appendix

CodeProject.AI Facial Registration/Recognition Scripts (Pi-Camera)

This section is to prepare you for the mini project. Repeat everything in section B except the webcam is replaced by a pi-camera and its video stream will be captured with a RTSP URL.

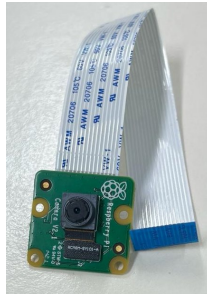


Figure 10: Raspberry Pi Camera module V2

Pi-camera is a raspberry pi camera module that can be connected to your raspberry pi through CSI port to take pictures and record videos. The sensor has 5-megapixel native resolution in still capture mode. In video mode it supports capture resolutions up to 1080p at 30 frames per second. It is also light-weight and small, making it the ideal camera to be used in mini projects.

1. Disconnect the 5G hat and connect pi-camera to your raspberry pi's CSI port.

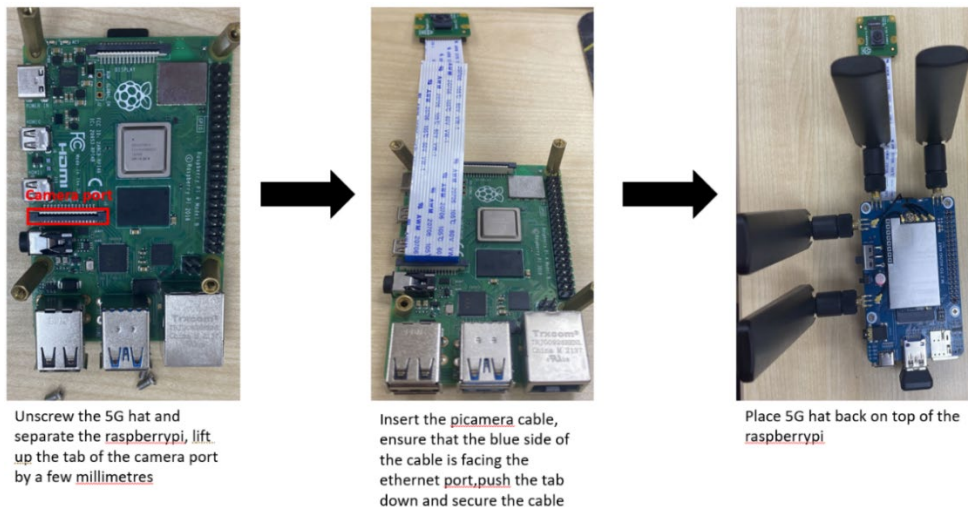


Figure 11: Connect Pi-Camera to Raspberry Pi

2. Connect monitor, keyboard and mouse to raspberry pi. Power on everything. Enable legacy camera using `sudo raspi-config`.
3. Execute this command `raspistill -o Desktop/image.jpg` in a terminal to test pi-camera. View the captured image.jpg in the Desktop folder in the raspberry pi.
4. Download `mediamtx_v1.1.1_linux_arm64v8.tar.gz` and transfer this file to the pi. Extract everything inside the pi.

5. Execute this command in a pi terminal to stream the pi-camera.

```
./mediamtx &
v4l2-ctl --set-ctrl video_bitrate=500000 &
ffmpeg -input_format h264 -f video4linux2 -video_size 1280x720 -framerate 60 -i
/dev/video0 -c:v copy -an -f rtsp rtsp://localhost:8554/feeder -rtsp_transport tcp
```

6. Copy the code below and named the python script as livestream.py
7. Pip install imutils and opencv-python if you have not done so
8. Because a pi-camera is stream over RTSP, the stream from the pi-camera can be captured with a RTSP URL which is set only when you execute the command in 3. The RTSP URL is in the format of **rtsp://localhost:8554/feeder** where localhost is the IP address of the raspberry pi's eth0, wlan0 or wwan0. Since we are not using 5G for this section, use the IP address of eth0.

livestream.py

```
import cv2
import imutils
from imutils.video import VideoStream

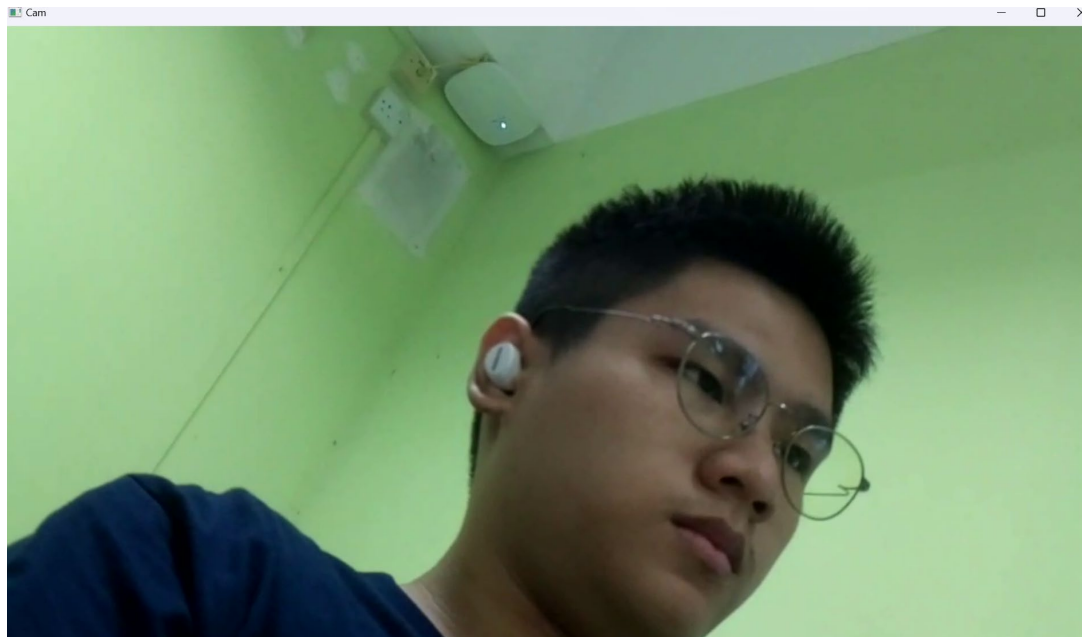
rtsp_url = "rtsp://169.254.15.175:8554/feeder"

def main():
    vs = VideoStream(rtsp_url).start() # Open the RTSP stream
    while True:
        # Grab a frame at a time
        frame = vs.read()
        if frame is None:
            continue
        # Resize and display the frame on the screen
        frame = imutils.resize(frame, width=1280,height=720)
        cv2.imshow('Cam', frame)

        # Wait for the user to hit 'q' for quit
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            break
    # Clean up
    cv2.destroyAllWindows()
    vs.stop()

if __name__ == "__main__":
    main()
```

A Live Video will pop out in windows!



9. Edit registration.py and recognition.py.
10. Change this line **cap = VideoStream(0).start()**, replace 0 with your own RTSP URL.
cap = VideoStream(rtsp://169.254.15.175:8554/feeder).start()