CLASS ATTENDANCE & FACE MASK COMPLIANCE DETECTION SOFTWARE

Technical Specification

Authors

David Weir (19433086)

Cian Mullarkey (19763555)

Supervisor

Hossein Javidnia

Date Finished

04/03/2022

1. Introduction

1.1 Abstract

The general area covered by this project is that of machine vision, in particular facial recognition. The aim of the project is to achieve a functioning classroom attendance log using a camera and a facial recognition algorithm to identify and then log and record students who have attended the particular class. The project is also capable of noting whether a student is wearing a face mask and making note of this along with their attendance.

1.2 Glossary

Machine Vision - Machine learning is the study of computer algorithms that can improve automatically through experience and by the use of data.

Facial Recognition - Facial recognition system is a technology capable of matching a human face from a digital image or a video frame against a database of faces by pinpointing and measuring facial features from a given image.

Dataset: A folder containing a number of subfolders storing the images of students and named after the relevant student.

GUI: A graphical user interface, a menu that allows the user to control the system and its functions.

Feature Extraction: The process of extracting face component features (e.g. eyes, nose, mouth) from the image of a human face.

System Architecture - A system architecture is the conceptual model that defines the structure, behaviour, and more views of a system.

High Level Design (HDL) - Explains the architecture that would be used in the development of a system. Provides an overview of the system identifying the main components of the system

1.2 Overview/motivation

Our motivation to undertake this particular project came about through our research of potential project ideas earlier in the academic year. While neither of us had a particular project or technology that we had our heart set on, after some research we both agreed that machine vision and facial recognition were technologies that appealed to both of us as it was something we both found interesting, we felt it would be a challenge to us, as well as the fact that it’s a relevant topic in the current technical landscape.

We found that this project had practical applications that could be used outside of academia which made the project all the more interesting to us. Many institutions and organisations can always make use of facial recognition software to take attendance in classes or

meetings etc. Additionally, with the prevalent threat of COVID19 the use of machine vision could assist organisations with ensuring compliance to the face mask rules set by the government.

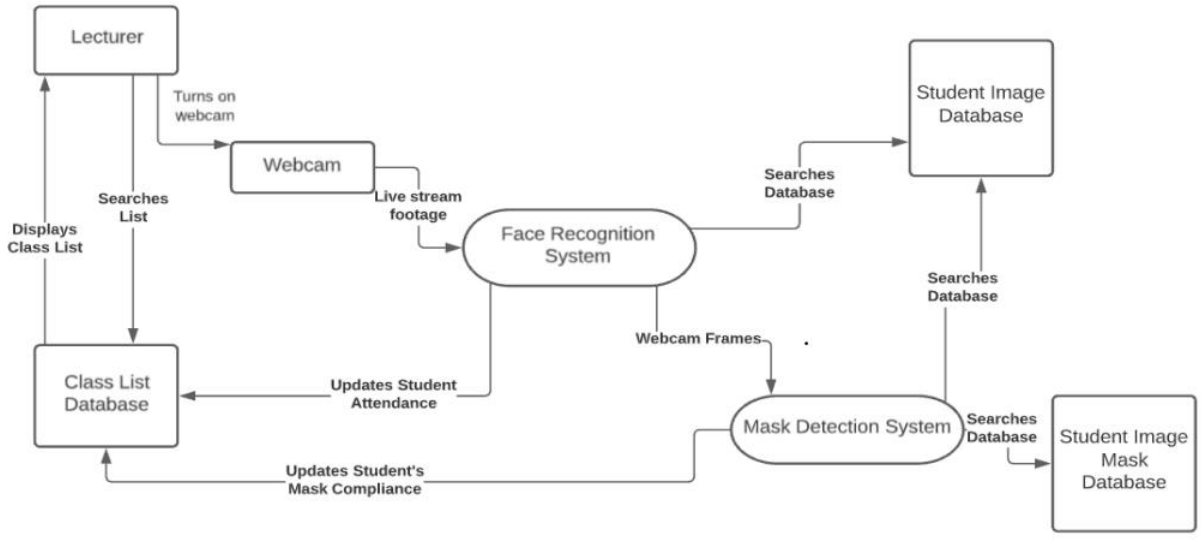
2. High-level design

2.1 High Level Design (HLD) Description

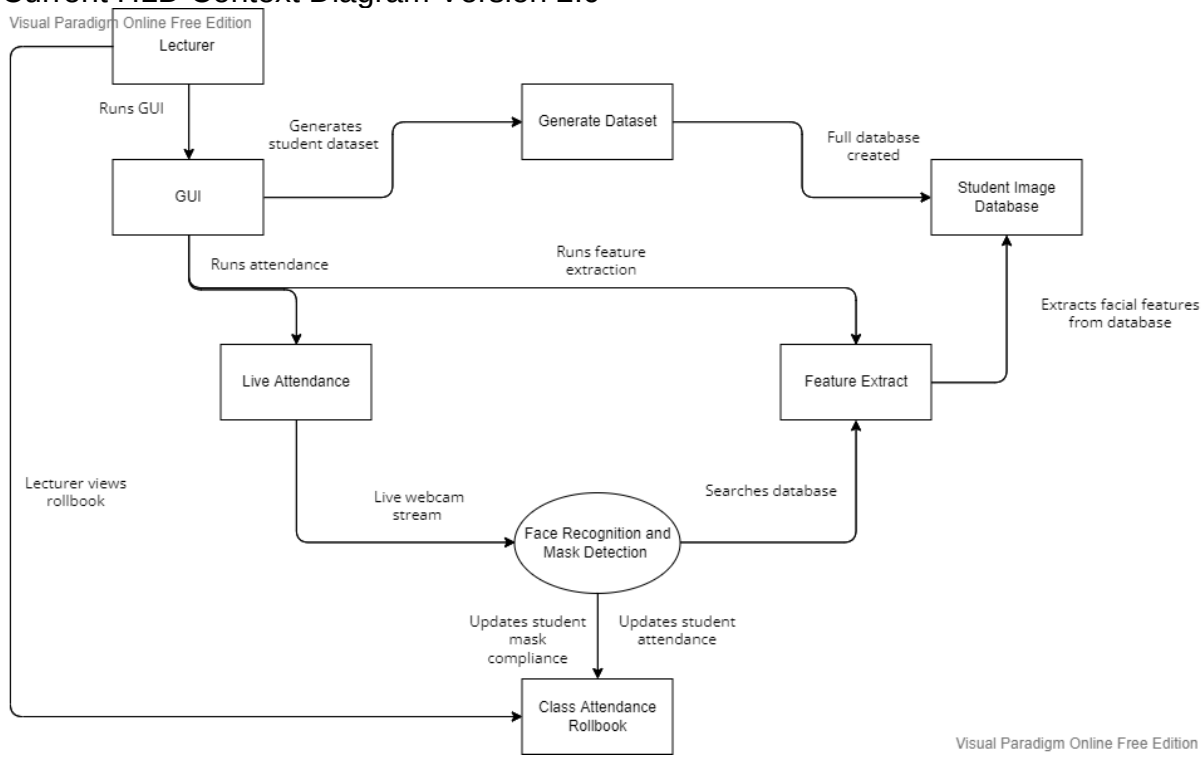
- Lecturer runs the application GUI.
- The students first and last name are entered and dataset generator function is run, capturing images of their face which are entered in a database.
- Lecturer runs feature extraction function which creates a recognition model for use during attendance taking. It only needs to be run once after all students have database entries and anytime a student has been added or removed from the database.
- Lecturer runs attendance function to begin a live webcam of the class that is sent to the facial recognition system.
- The system takes the frames from the live webcam and using the faces from each frame matches the face to images from the student image database.
- The system updates the student's attendance based on finding a match or not.
- The mask detection system then determines if the student is wearing a mask.
- The student's mask compliance is updated on the class list database by the system.

2.2 High Level Design Context Diagram

Initial HLD Context Diagram Version 1.0

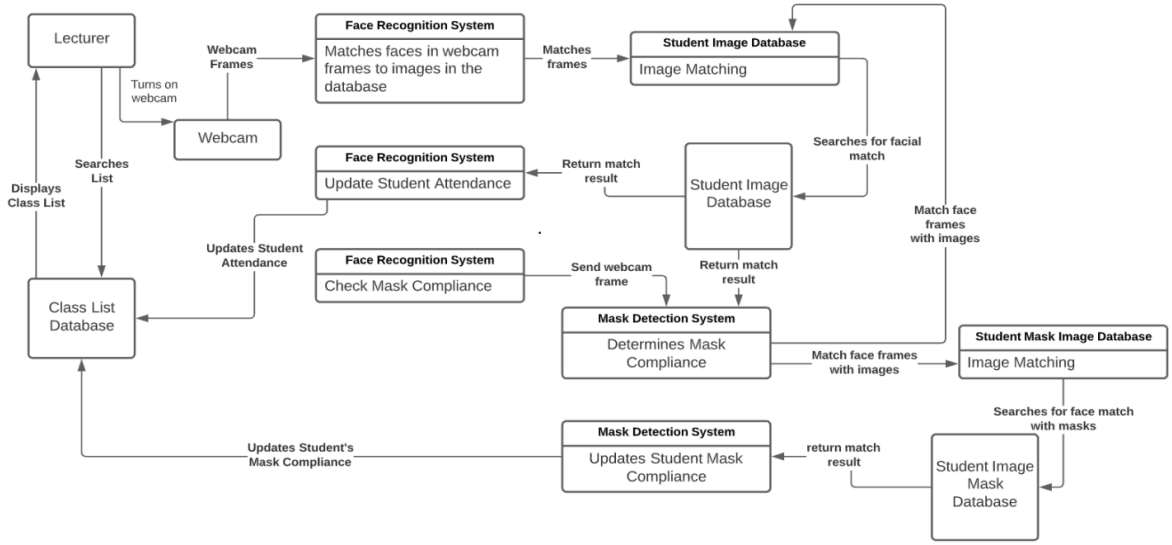


Current HLD Context Diagram Version 2.0

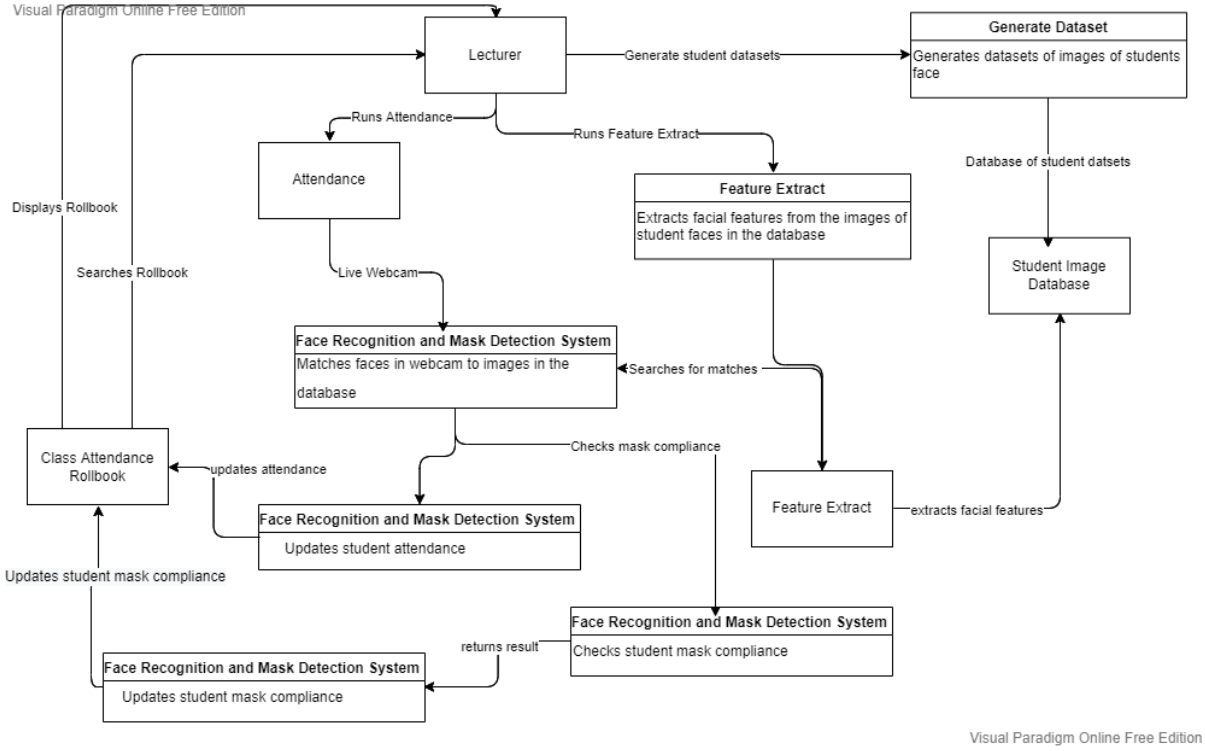


2.3 Data-Flow Diagram

Initial Data-Flow Diagram Version 1.0

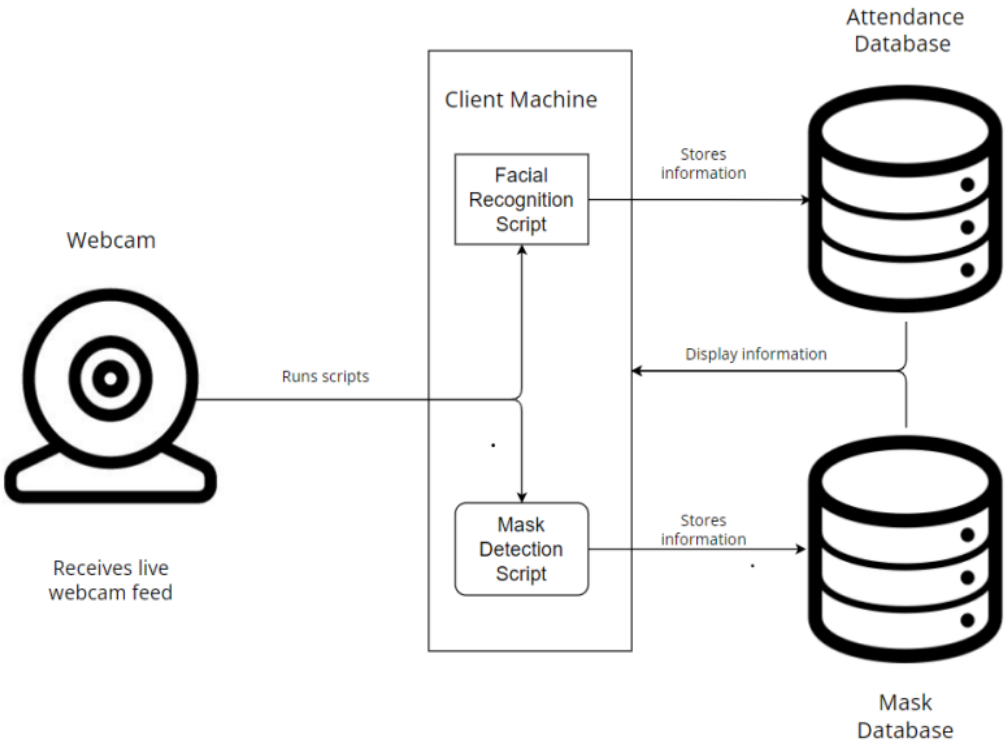


Current Data-Flow Diagram Version 2.0

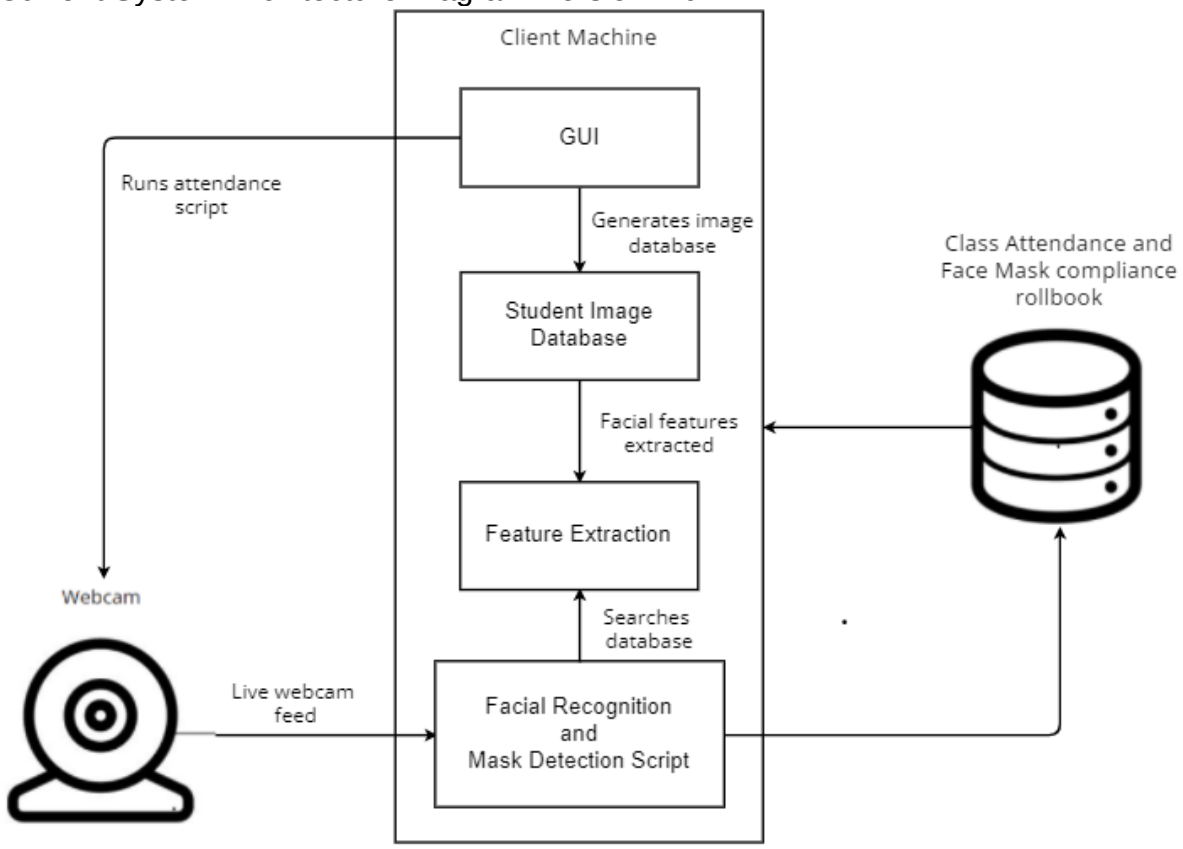


3. System Architecture

Initial System Architecture Diagram Version 1.0



Current System Architecture Diagram Version 2.0



4. Implementation

‘dataset_generator.py’

Our implementation begins with the ‘dataset_generator.py’ file. This is the opening stage of the implementation as this file is the generates the images of the individual which are later used to build the facial recognition model.

After creating the dataset folder using the user inputted individuals first and last name, a haar cascade is loaded in. This is an XML file containing data which is used to detect data.

```
# haar cascade - create a cascade initialized with the face cascade
# Loads the face cascade (XML file containing data to detect faces) into memory
faceCascade = cv2.CascadeClassifier('./haar_cascades/haarcascade_frontalface_default.xml')
```

Following this we have three different functions, front, right and left. The purpose of these functions is very similar, firstly to take 30 images of the front of the user’s face, followed by 5 of the left of their face and 5 of the right. Whilst we originally implemented just 1 function which took 30 photos of the user’s face, we implemented these extra 2 functions based on supervisor feedback in order to increase the applications recognition ability. The functions run in order and are separated by a ‘sleep(5)’ call to give the user time to move their head.

```
# fucntion calls
front(faceCascade, height, width, path)
time.sleep(5)
right(faceCascade, height, width, path)
time.sleep(5)
left(faceCascade, height, width, path)
```

Each function sets the default webcam as the video source then oops the program until it captures the required number of images, whilst also drawing an identifying rectangle around the users face.

```
    img_count = 1
    while img_count <= 30:
        (_, image) = webcam.read()
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        faces = faceCascade.detectMultiScale(gray, 1.3, 4)

        # draw a identifying rectangle around the faces
        for (x, y, w, h) in faces:
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
            face = gray[y:y + h, x:x + w]
            face_resize = cv2.resize(face, (height, width))
            cv2.imwrite('% s/% s.png' % (path, img_count), face_resize)
            img_count += 1
```

The images taken are stored in the previously created datasets

‘feature_extract.py’

The next file in our implementation is ‘feature_extract.py’. This file trains the recognition model by extracting facial features from the images in each dataset sub folder generated by the previous file.

This file begins by retrieving the path of each sub-data folder in the datasets folder.

```
# the datasets folder contains sub-folders containing images of various people
imgPaths = list(paths.list_images('datasets'))
```

It then loops through the images and extracts each user’s name for recognition from the path. The images are converted from OpenCV ordering BGR to dlib ordering RGB in order to allow the program to locate the faces using the facial recognition library.

```
# load the input image: convert it from OpenCV ordering BGR to dlib ordering RGB
image = cv2.imread(imgPath)
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# locate faces using the face_recognition library
boxes = face_recognition.face_locations(rgb, model='hog')
```

Facial embeddings of the face are then computed and saved, along with the names, in a data dictionary saved to the ‘face_enc’ file.

‘webcam_recogntion.py’

The core of the system’s implementation is based in this webcam recognition file. This is where the actual face recognition, mask detection and attendance recording are done.

To begin we load a pre-trained model using the load_model function from Keras.

```
model = load_model("./mask_detector.model")  # load trained model
```

This model is used to detect faces that are wearing masks (which is otherwise not possible using only a Haar Cascade) and used in deciding if someone is wearing a mask.

We set 2 dictionaries which are simply used for labelling as we can see below.

```
# labels and colour dictionaries
labels_dict = {0: 'Mask', 1: 'No Mask'}
colour_dict = {0: (0, 255, 0), 1: (0, 0, 255)}
```

After this we load are Haar cascade into the classifier and load our face_enc file (file created with the face embeddings of our dataset of people) and begin streaming. Initially, we stream from the machine’s pre-set main camera however, this can be changed to a different camera output if necessary.

```
# load the haar cascade in the cascade classifier
faceCascade =
cv2.CascadeClassifier('./haar_cascades/haarcascade_frontalface_alt2.xml')
# load the known faces and embeddings
data = pickle.loads(open('face_enc', "rb").read())

print("Streaming...")

# begin webcam feed
webcam = cv2.VideoCapture(0, cv2.CAP_DSHOW)  # main camera capture at 0
```

Once the stream is established, we create our seen and attendance lists which will be used to record student attendance information and, inside a while loop, continuously grab each frame from the video stream. Using detectMultiScale we can detect faces and return a set of 4 coordinates that create a box around the detected face.

```
seen = []
attendance = []  # list of student attendance and times

# loop over frames from the video file stream
while True:
    # grab the frame from the threaded video stream
    ret, frame = webcam.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,  # greyscale image
        scaleFactor=1.1,  # compensates for the different distances from
the camera of the faces
        minNeighbors=5,  # defines how many objects are detected near the
current object before
```

```
        # declaring a face is detected
        minSize=(60, 60),  # size of each window
        flags=cv2.CASCADE_SCALE_IMAGE
    )
```

We then convert the input frame from the camera from BGR to RGB and using the face_encodings functions from face_recognition, create a 128-dimension face encoding for each face found in that frame.

```
# convert the input frame from BGR to RGB
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

# the facial embeddings for the face in input
encodings = face_recognition.face_encodings(rgb)
names = []
```

We use this to loop over each embedding for each face and compare these encodings with the encodings stored in the face_enc file. Using compare_faces, we create arrays with TRUE and FALSE values depending on if the encodings from both sources match. TRUE if they match and FALSE otherwise. If no encoding match exists we set the name to unknown.

```
# loop over the facial embeddings
# this loops over the embeddings of each face we have in the datasets
folder
# it allow us to recognise all the faces on the web feed (assuming there is
more than 1 at any time)
for encoding in encodings:

    # Compare encodings with encodings in data["encodings"]

    # matches will be an array of True boolean values for images that match
the data
    # and False boolean values for images that are unknown or do not match
    matches = face_recognition.compare_faces(data["encodings"], encoding)

    # set name as unknown if no encoding matches exist
    name = "Unknown"
```

For every match we get, we find and store the position in the matches array where we found a True. With each of these indexes, we find the name at that index in our face_enc file and increment our count of that name. The name we accept as having recognised is the one with the highest count, which is appended to our list of names.

```
# checks for found matches
if True in matches:

    # Find positions at which we get True and store
    matchedIds = [i for (i, b) in enumerate(matches) if b]
    counts = {}

    # loop over the matched indexes and maintain a count for each
recognized face
    for i in matchedIds:

        # Check the names at respective indexes we stored in matchedIds
        name = data["names"][i]

        # increase count for that name
        counts[name] = counts.get(name, 0) + 1
```

```
    # set the name which has highest count
    name = max(counts, key=counts.get)

# update the list of names
names.append(name)
```

Finally, we loop over the recognised faces, rescaling the image sizes, and make a prediction of if they are wearing a mask or not.

```
# loop over the recognized faces
for ((x, y, w, h), name) in zip(faces, names):

    # rescale the face coordinates

    # Save just the rectangle faces in SubRecFaces
    # resize images
    face_img = frame[y:y + h, x:x + w]
    resized = cv2.resize(face_img, (224, 224))
    normalized = resized / 255.0
    reshaped = np.reshape(normalized, (1, 224, 224, 3))
    reshaped = np.vstack([reshaped])
    result = model.predict(reshaped)

    label = np.argmax(result, axis=1)[0]  # mask or no mask label
prediction
```

We also take the name, time and prediction of each student, creating a list with each as an element and then adding this list to a greater list of attendance. We also add the student's name to the list of seen so their attendance is not marked more than once.

```
# take attendance info for recognised student, append it to attendance
information list
if name not in seen:
    student = []
    student.append(name)
    student.append(datetime.now().strftime("%H:%M:%S"))
    student.append(labels_dict[label])

    attendance.append(student)
    seen.append(name)  # update list of seen students
```

We also display a rectangle (red or green depending on if a mask) and 2 labels, the name and Mask or No Mask. This is displayed on the user's webcam display where the system has detected a face.

```
# take attendance info for recognised student, append it to attendance
information list
        if name not in seen:
            student = []
            student.append(name)
            student.append(datetime.now().strftime("%H:%M:%S"))
            student.append(labels_dict[label])

            attendance.append(student)
            seen.append(name)  # update list of seen students

        # mark the detected face with a rectangle drawn on the display
        cv2.rectangle(frame, (x, y), (x + w, y + h), colour_dict[label], 2)
```

```
        # put the name of the recognised person/face on the screen
        cv2.putText(frame, name, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
(0, 255, 0), 2)

        # Mask / No Mask Label
        cv2.putText(frame, labels_dict[label], (x + 5, y + h + 25),
cv2.FONT_HERSHEY_DUPLEX, 1,
                    colour_dict[label], 2)

cv2.imshow("Webcam", frame)  # display

# end program if "q" key is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

Lastly, this file create a CSV file called Rollbook with the today's date and writes each student's name, time of attendance and if they are wearing a mask to the file.

```
# create a rollbook text file with today's data
rollbook_name = "RollBook " + datetime.today().strftime("%d.%m.%y") +
".csv"
rollbook = open(rollbook_name, "w")

# rollbook titles
rollbook.write("Student Name, " + "Time, " + "Mask/No Mask\n")

# add attendance info for each student to a comma delimited text file
for info in attendance:
    rollbook.write(info[0] + ", " + info[1] + ", " + info[2] + "\n")

webcam.release()
cv2.destroyAllWindows()
```

'gui.py'

The final part of our implementation was to create a user interface for our application. While we initially planned to utilise a simple command line interpreter based interface, however as the project developed we felt it would benefit from a proper window application UI.

To create this we utilised the python GUI framework Tkinter to create this simple GUI. This involved standard set up of the window size, initial title, labels and setting of an icon and background image.

Following this we implemented 4 different buttons, the first of which would run the user manual pdf.

```
def user_manual():
    path = "../docs/3-final-reports/User Manual.pdf"
    subprocess.Popen([path], shell=True)
```

The next button was for the dataset generator and involved the creation of entries with sufficient error handling to allow the user to enter the desired name for the dataset.

```
# create labels and entries for fname and lname
first_name_label = tk.Label(root, text='First Name', font=('calibre', 10, 'bold'))
first_name_label.place(relx=0.4, rely=0.6, anchor='center')


first_name_entry = tk.Entry(root, textvariable=first_name_var, font=('calibre', 10, 'normal'))
first_name_entry.place(relx=0.6, rely=0.6, anchor='center')


last_name_label = tk.Label(root, text='Last Name', font=('calibre', 10, 'bold'))
last_name_label.place(relx=0.4, rely=0.65, anchor='center')

last_name_entry = tk.Entry(root, textvariable=last_name_var, font=('calibre', 10, 'normal'))
last_name_entry.place(relx=0.6, rely=0.65, anchor='center')
```

The following button would simply run the feature extract program described above, and this was the same again for the webcam recognition program when it came to the last button.

```
def feat_ext():
    MsgBox = tk.messagebox.askquestion('Feature Extract', 'Do you wish to begin extracting features from the '
                                                          'datasets?\n '
                                                          'This may take up to a minute or two.',
                                       icon='info')
    if MsgBox == 'yes':
        print("Extracting features")
        extract()
        messagebox.showinfo("Feature Extract", "Extracted facial features successfully.")
        print("Extracted facial features successfully")


# calls webcam recognition program
def attend():
    # print("Streaming")
    MsgBox = tk.messagebox.askquestion('Live Attendance',
                                       'Do you wish to begin taking live attendance and mask detection?\n',
                                       icon='info')
    if MsgBox == 'yes':
        # print("Streaming")
        messagebox.showinfo("Live Attendance", "Beginning Stream.\n" "Press 'Q' to end stream.")
        live_attendance()
        messagebox.showinfo("Live Attendance", "Stream Ended.")
        print("Stream Ended")
```

5. Problems solved

Whilst we faced many individual problems over the course of this project, the major problem we faced was implementing the mask detection itself.

6. Future work

If we are to continue to work on this project in the future, we would definitely focus our immediate attention upon improving the recognition model itself. Whilst we are quite satisfied with the levels of recognition that it can currently achieve as we feel it is satisfactory for this project, there are certainly areas of it that could be improved upon.

Obstacles such as glasses, long hair as well as lighting are areas with which our application can have trouble with, and if continuing to work on this project into the future it would be these obstacles that we would try and initially tackle and improve upon.

We also feel that there is a lot of potential for a project such as this to branch out to even further uses than we have designed it for, that being classroom attendance and mask detection and compliance in classrooms. Mask detection in particular could be an interesting area to further explore outside of just the classroom environment, though we are aware of the potential ethical concerns that may prove an expansion like this challenging.