HOCHSCHULE RHEIN-WAAL

FACULTY OF TECHNOLOGY & BIONICS

Bachelor Thesis

Development of attendance tracking system using computer vision based AI on an embedded system

Anmol Singh

Germany, 2022

HOCHSCHULE RHEIN-WAAL

FACULTY OF TECHNOLOGY & BIONICS

Bachelor's Thesis in Mechatronic Systems Engineering in the fields of Computer Vision, Artificial Intelligence, Embedded Systems and Programming

Development of attendance tracking system using computer vision based AI on an embedded system

Author:	Anmol Singh
Matriculation Nr:	24244
Study Program:	Mechatronic Systems Engineering
Supervisor:	Prof. Dr. Ronny Hartanto
Co-Supervisor:	Prof. Dr. Matthias Krauledat
Start Date:	16 July, 2022
Submission Date:	10 October, 2022
Place:	Robotics Laboratory, HSRW, Kleve, Germany

Preface

This effort of the bachelor thesis is an extension and perpetuation of the alpha version of the attendance tracker system developed in 2019 jointly by two former master's students: Mr. Abir Bhattacharya and Mr. Mamen Thomas Chembakasseril who took the pains to initiate and set this project in motion, and contributed in advancing on the logic and graphical user interface separately. The status of the alpha version was quite disjoint with no clear workflow structure and no integration of the logic and the graphical user interface. The original idea and goal of this project are to get rid of the paper-based attendance sheets from the robotics laboratory and to have "login/logout as contact-free as possible". My desired outcome was to rework the complete project from scratch which includes the logic(backend), GUI(frontend), merging of the frontend and backend, optimisation, and executing the software package on an embedded system(raspberry pi) accompanying a small camera and a touchscreen for the user to interact with the software. I took up this project as it matched the skill set I gained from the internship, my passion, and is also to fulfill the bachelor's graduation requirements at Hochschule Rhein-Waal in Mechatronic Systems Engineering.

Declaration

I confirm that this bachelor's thesis in mechatronic systems engineering in the fields of computer vision, artificial intelligence, embedded systems and programming is my own work and I have documented all sources and material used.

Kleve, Germany, 10 October, 2022 Anmol Singh

Disclaimer

This work of bachelor thesis is written and completed in accordance with the German Federal Data Protection Act a.k.a Bundesdatenschutzgesetz (BDSG) [1]. The sole purpose of the facial recognition encodings to be saved by this attendance tracker in the database is to fulfill the very functionality of this software being used in the robotics laboratory of the Rhine Waal University of Applied Sciences and is not allowed to be used or published anywhere else. Moreover, this software does not save the picture of the users in the database, but rather just the encodings which are not written in a human-readable format as shown in fig: 4.48.

Dedication

My bachelor thesis work has been assuredly devoted to my mom Dr. Jasjeet Kaur, my dad Mr. Satnam Singh and my sister Ms. Gurneet Kaur whose unconditional love and affection, blessings, moral support, and principled guidance strengthened me to pursue my bachelor's degree in Germany. The fruit of this hard work and dedication would have never been possible without them. I owe everything to them. I love you all from the core of my heart.

Thanks to the Almighty who bestowed us this life, with good people, good food, and beautiful nature.

Acknowledgement

This is a very short and intriguing narrative about how I ended up working as a bachelor thesis student at the robotics laboratory of the Rhine Waal University of Applied Sciences. Curious to know my story? If you have already found my thesis relaxing on the professor's desk or nicely stacked on one of the shelves in the library or through a quality search on the internet, then I would suggest you spend a little more time going through this book in one of those beautiful hopes from the author's point of view to generate some spark, interest, and enthusiasm in the reader's mind for computer vision, artificial intelligence, and embedded systems.

My cruise has been under construction since 2019 when I first started building the vessel with the most fundamental building block: "learning Python". The ship made its maiden voyage last year in September when I started as an intern at a startup called IRS under the mentorship of Mr. Giridhar Bukka. His project aim; to build an autonomous drone for conducting visual inspections of wind turbine blades was my first real-world problem challenge where I contributed my skillset in building a tracker code based on computer vision and AI models for the manipulation of the UAV during the flight. I would like to thank and express my gratitude to Mr. Bukka for welcoming me on board in his project , igniting my passion to follow these fields, and for his constant support throughout my internship.

I wanted to continue my thesis in the same field ever since; when Prof. Dr. Matthias Krauledat and Prof. Dr. Ronny Hartanto offered me a golden opportunity to start my thesis at the robotics laboratory of HSRW. I am incredibly thankful and indebted to them for giving me a chance to work on this project, their expertise, motivation, and indispensable feedback helped me push forward through this thesis topic to make it a success.

I would also like to extend my sincere thanks to Mr. Mamen Thomas Chembakasseril who stood by my side, guided and motivated me throughout my thesis project with his past experiences in python, CV, and AI.

I would like to thank all my friends and especially my best friend Pulkit Mahajan for being in my life. His rocksteady relentless support, funny sense of humor, and constructive criticism gave meaning to my life and helped me sail through all the ups and downs seamlessly.

Finally and most importantly, I would like to convey my deep regards and love to my mom, my dad, and my sister for their love and constant support in every possible way throughout my stay abroad for my bachelor study program in Mechatronic Systems in Germany.

Abstract

The scope of this thesis presents the development of an attendance tracker system from scratch using computer vision based AI deployed on an embedded system. As students frequently forget to fill out the user attendance list and the carbon footprints need to be physically stored for records, conventional techniques of attendance monitoring such as signing attendance sheets or calling out names of each student by the professor are becoming woefully out of date, repetitive, and dull. This is where the project comes into play. It aims to automate the entire system by implementing a quicker and more secure method of recording attendance that uses face recognition to authenticate the user and gesture recognition to identify the time of entry and exit of the lab, which are alluded to as login and logout. The face recognition has been implemented using the face-recognition python library whereas the hand detection and gesture recognition has been implemented using the MediaPipe library. The system has been meticulously packaged into a complete software that includes an interactive graphical user interface supporting all the key functionalities aside from user login, including: registering a new user in the database, deleting the user from the database, and changing other crucial system parameters from system preferences. This has been done to facilitate and mesmerize user experience. Python has been used only in the writing and development of the software because the goal of this thesis is to create a quick prototype. The finished product has a touchscreen, camera, and Raspberry Pi 4 inside a Polylactic Acid (PLA) case that was 3D printed, making it portable, lightweight, and simple to assemble.

Contents

1. Introduction

1.1. Background

Attendance tracking is a form of authentication system to assert the verification of an individual and maintain a record of time information for every person. Attendance systems exist to safeguard access to repositories and protected information to authorised personnel only and to monitor the employees working hours.

The most traditional method used to take attendance(especially in schools and colleges) is by calling out the names of every student by the professor to mark as present or absent respectively or to pass an attendance sheet so that every student can fill up their credentials such as Unique Identification Number (UID), signature along with their name and time details. These methods are also known as manual methods and are very vulnerable as the credentials can be misused. With the introduction of computers, the world started drifting towards automation rapidly as they proved to be very reliable and efficient to carry out extremely complicated tasks quickly and accurately. Similarly, attempts were made to even automate the attendance systems by incorporating various technologies at hand and getting rid of storing carbon footprints.

Several papers exist which explain the usage of different technologies and concepts to monitor attendance effectively. In the research paper:[2], proposes to automate the system using Quick Response (QR) code. The idea is that every student scans the QR code with a specific mobile application. For their identity check, the application captures image of the student during the scan and sends the information to the server module to run identity checks and confirm the attendance. However, the drawback of this system is if the wifi or the server is down, then the system won't work.

Another paper [3], prefers to automate the system using Radio Frequency Identification (RFID) technology. Each student gets a card incorporated with the RFID tag, which is unique to each user. In order to make the attendance, the UID card containing the tag needs to be scanned in-front of the RFID reader which reads out information. The reader is in turn connected to a computer which stores the attendance in the database as soon as the tag is scanned. Barcodes [4] and magnetic stripe scannable cards [5] as attendance systems also work similar to RFID tags. They need to be scanned by a barcode scanner and swiped through a reading machine respectively which extracts the unique encrypted details. However, the biggest disadvantage with them are that they can be easily passed on to another person to mark a fraudulent attendance.

Reference [6], comes up with fingerprint technology for attendance system. The portable device can be passed to every individual to mark their attendance. The device communicates with a host computer via Universal Serial Bus (USB) having a Graphical User Interface (GUI) application to manage the device and attendance. The downside is that it is not a contact free technique which is a bad option to be used in a pandemic situation as well as requires additional hardware.

Other way of attendance solution is using speech biometrics [7]. The users access the system by making a call to a tollfree number from pre-decided phones. This call basically connects to the server computer. An Interactive Voice Response (IVR) system then guides the user for enrollment or attendance procedure. Registration and verification is done by the user by speaking some text. This makes the process very slow. Moreover, a malfunctioning telephone network can cause entire system failure. User identity verification can also be carried out simply by logging in the system with your username and password which is very easy to be misused and tapered with.

Reference [8] prefers solving the problem by introducing location as the key aspect during attendance. The coordinates of the company are matched with the coordinates of the employee extracted through Global Positioning System (GPS) while trying to log in. If the match is true, then the attendance is marked. The attendance subsequently gets stored in the database. Iris biometric recognition [9] is also a way to computerise the attendance system. Iris, being unique to every individual is scanned by high resolution iris scanners and if the match against the stored database is true, attendance is marked. But this system requires additional hardware of having a good quality iris scanner which makes it expensive.

Introduction of Computer Vision (CV) and Artificial Intelligence (AI) opened up new boundaries which is evident from the papers: [10, 11, 12] which elaborates in automating the system by running face recognition to mark the attendance of the students. [13] shows, how different hand gestures can be used to control a GUI contact free which needs a face verification to grant the access to the GUI.

I chose to use the combination of face detection, face recognition and gesture recognition to develop the attendance tracker system. The reasons to choose have been mentioned briefly in section 1.4: Proposed solution.

1.2. Problem Statement, Motivation & Description

Good engineers always tend to be slothful and are doctorates in the field of procrastination. This is the basic ideology behind the development of this project. The former students working in the laboratory seemed very lazy in filling out the pen and paper-based attendance sheet while entering and exiting the lab. At this point, the project sprang to life with the idea of getting rid of carbon footprints and having a high-tech attendance tracker system that could recognise the face and fulfill the logging procedure by storing the time stamps of login and logout of a unique individual autonomously thus creating a digital attendance sheet. Moreover, a digital logger opens boundaries to add more functionalities in the future such as having a monthly backup of digital attendance logs online on a server, and remote access to the attendance log from anywhere in the world by connecting it via the internet.

With the introduction of the covid-19 virus as a pandemic, this project became even more crucial to have a contact-free attendance system to stop the spread of the virus whereas the paper-based attendance system was a total disaster. As per the article published online by cdc.gov[14], the covid virus can be transmitted by contacting infected surfaces, although not being the main source of virus spread and transmission as the risk is low compared to other major routes such as aerosols. This risk can be mitigated by introducing proper hand hygiene, disinfection of surfaces, and introducing contact-free techniques to discharge daily chores. The proposal to develop an automated attendance tracker system attempts to make the whole process: "as contact- free as possible", thus taking into account the health of the students and the staff working in the lab.

1.3. Thesis Objective

The eventual focus of the author of this bachelor thesis is to develop a complete software package from scratch equipped with an interactive graphical user interface that fulfills and executes the entire logic with all the necessary functionalities as the backend, locally on an embedded system thus attaining the desired outcome of having a standalone software application running on an embedded system as a unit device installed at the entrance of the laboratory.

The long-term objective of robotics lab supervisor Prof. Dr. Ronny Hartanto is to make a smart high-tech laboratory that can be controlled remotely and this attendance tracker system is also a part of that prolonged target.

1.4. Proposed Solution

This thesis essentially aims to develop a software capable of running on an embedded system. After obtaining the software requirement specifications, it branches out into 2 parts: developing front-end and the back-end. Once both of them are completed individually, the project enters the integration phase where the GUI and the logic are consolidated together. Then the project sets foot in the testing and optimisation phase before deploying it finally on the embedded system. The general thesis workflow has been visualized graphically in Figure 3.1.

The proposed solution for the software workflow has been conceptualized concretely in the Figure 3.2. It predominantly consists of 2 blocks of logic being handled and executed by the

GUI: user login and admin controlled system preferences. The conventional way to log in/out of the lab is to fill in the attendance sheet with username, matriculation number, signature, start time, end time and the project. Evidently, this is not at all a secure method for attendance logging as the user credentials can be misused. Additionally, due to the introduction of the pandemic the old method has to be discontinued. So, face detection, face recognition as well as gesture recognition was particularly chosen as a solution to this problem as face recognition system helps in identifying a unique user, is more secure than the conventional solution, discards any lame justification for the lethargic students of not logging in/out as the process is completely automated and most importantly, minimizes contact which is a perfect remedy during pandemic situation. The admin controlled system preferences are rudimentary indispensable features to complete this software package which provides the admin an edge to register or delete a user and also change some system settings through the GUI itself. Moreover, it runs completely locally on the embedded system without any need to rely on internet or external servers/clouds which prevents any possibility of data breach, can run in the absence of professor once registered, user login/logout is very fast and accurate, no requirement of any fancy hardware unlike iris or fingerprint scanners and also no likelihood of fake proxy from another location as the device itself is internet free and installed at the lab entrance. So any sort of location coordinates are not required.

1.5. Tools Used

During the span of the development phase of the project and documentation of the thesis, several tools were used as an aid to manage and complete it. For the software part, Python is the core basic tool used for developing the frontend and backend along with the help of external packages such as face-recognition, opencv-python, numpy, and mediapipe. Amongst other famous tools required by any other software developer are: Visual Studio Code[15] as an Integrated Development Environment (IDE) for the development and management of the attendance tracker code, Github[16] for version control and Continous Integration/Continous Development (CI/CD), Miro[17] as well as app.diagrams.net(formerly known as draw.io)[18] to make intuitive and beautiful flowcharts for visualizing the logic and the graphical user interface.

For the hardware part: raspberry pi 4, a raspberry pi camera and a raspberry pi touchscreen has been used to make it a complete embedded system that has been incorporated glamorously in a 3D printed case, printed with the aid of Prusa i3 MK3. Lastly, the complete documentation of the thesis has been taken care of and organized by overleaf, an online latex editor and Zotero: the referencing management system.

1.6. Thesis Structure Overview

This thesis incorporates the entire content in a total of 8 chapters and 1 appendix with the following structure:

- Chapter 1 talks about the background, problem statement, objective of the thesis and the initial plan of action to solve the problem followed by tools used and a brief overview of the thesis structure.
- Chapter 2 addresses the theoretical background of the different AI models being used and the basic knowledge of how those algorithms work.
- Chapter 3 discusses about the methodology used to solve the problem.
- Chapter 4 discusses about the actual implementation of the code.
- Chapter 5 deals with the optimisation of the software.
- Chapter 6 elaborates the results of optimisation observed.
- Whereas chapter 7 talks about the limitations and the possible improvements possible to this project in the future.
- Last but not least, chapter 8 summarizes the development of this attendance tracker software.
- Appendix A contains the complete code for the attendance tracker system.

2. Theoretical Background

This section briefly delves in the fundamental principles of the AI models used in this project namely: face detection to find a face in the frame, face recognition to identify the authenticity of the user and gesture recognition to identify the corresponding gestures, thumbs-up for login and thumbs-down for logout as referred to in the section 1.4: Proposed Solution. No details are mentioned as it is outside the scope of this bachelor thesis.

2.1. Face Detection

Face detection refers to a presence of a human face amongst many objects in a picture, video or a live camera feed[19]. It is basically a form of object detection.

2.1.1. Traditional Face Detection Model: HOG based

[20] HOG is also known as "Histogram of Oriented Gradients", a concept which originated in the 2005, being used for object detection in the field of CV.

To find faces in the digital data, HOG needs a black and white image. Subsequently, each and every pixel of the image is inspected one at a time and the darkness of the current pixel is compared to the immediate surrounding pixels as shown in Figure 2.1.

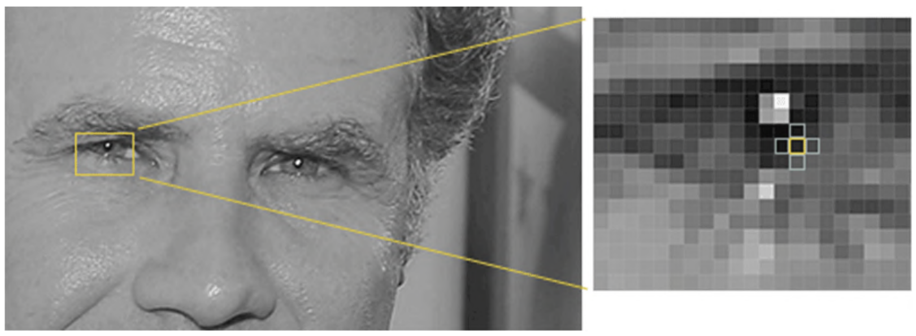


Figure 2.1.: HOG pixel inspection. [20].

Eventually, a vector is drawn from lighter pixel to the darker pixel as shown in Figure 2.2. Iterating this step for all pixels results in vectors over every pixel also known as gradients, essentially describing the flow from light to dark regions of the image. The reason to use gradients for the direction change in the brightness levels is that all sorts of images (dark/bright) would always end up with the same result using this idea.

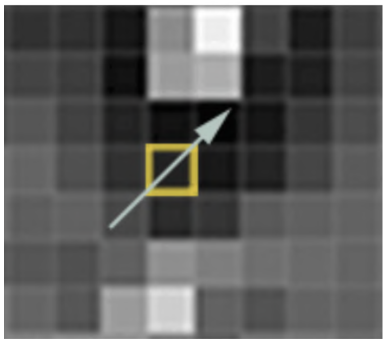


Figure 2.2.: HOG vector gradient. [20].

This procedure is done slightly on an upper level(means a bunch of pixels being considered as a single unit and a single vector is drawn over them) to just notice the fundamental pattern of the image. Once the HOG image is generated, it is compared to the pre-built HOG face image which was generated from a number of other face images, shown in Figure 2.3. With this method, face detection can be carried out very easily as in Figure 2.4.

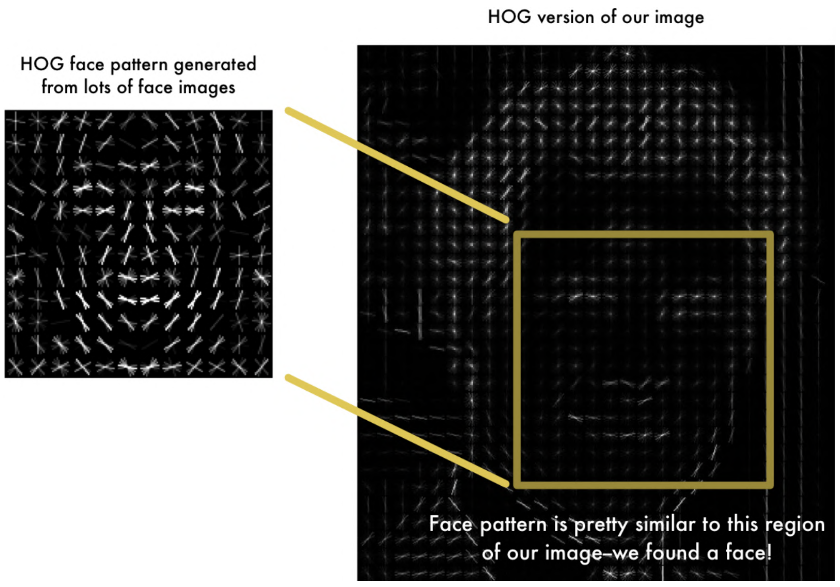

HOG version of our image
HOG face pattern generated from lots of face images
Face pattern is pretty similar to this region of our image–we found a face!

Figure 2.3.: HOG image comparison to detect face. [20].

Figure 2.4.: Detected face using HOG technique. [20].

2.1.2. Deep Learning Based Face Detection Model

The model used to detect faces in this project is known as BlazeFace, developed by Google Research and published in 2019. Blazeface is an extremely fast and lightweight model designed to run on mobile Graphical Processing Units (GPUs) and deliver inferences super realtime. It can perform single or multiple face detections in a single frame with Frames Per Second (FPS) values ranging from 200 to 1000 depending upon the device specifications. BlazeFace has been developed on the foundation of Single Shot Multibox Detector (SSD), an object detector with an evident improvement of having a better feature extractor with a particular convolutional unit called Blazeblock as shown in Figure 2.5. The anchor computation is shown in Figure 2.6. [21, 22, 23] (**No details mentioned: outside the scope of this thesis**).

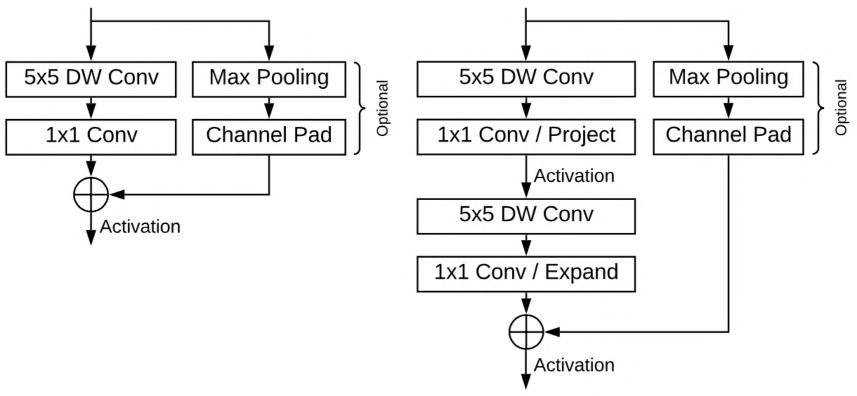


Figure 2.5.: BlazeBlock (left) and double BlazeBlock [22].

Brief specifications of the model are as follows[24]:
Model Type: Convolutional Neural Network (CNN)
Model Architecture: Convolutional Neural Network SSD-like

Model Input: Red Green Blue (RGB) image (from a video frame) resized to 128x128x3 pixels.
Model Output:

- Face bounding box coordinates.
- Detection Confidence Score
- 6 facial keypoint coordinates also known as landmarks: Left eye, right eye, nose tip, mouth, left eye tragion, right eye tragion.

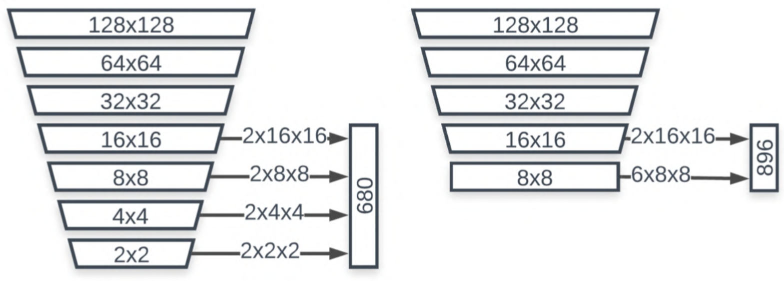


Figure 2.6.: Anchor computation: SSD (left) vs. BlazeFace [22].

2.2. Face Recognition

Face recognition means to identify a person in a given image/video or a live camera feed, which is primarily accomplished by extracting some features from the given input image and comparing it to the already stored features in the database. The image which gets matched in majority is selected as the desired outcome. [19]. The face recognition used in this project has been developed by Adam Geitgey using state of the art face recognition originally by dlib c++ library [25] toolkit built using deep learning neatly combined into a python package/library [26]. Face recognition is achieved in many steps which are as follows [20, 27, 28]:

1. **Locating all faces in the given frame:** This is achieved by face detection algorithm: HOG based as discussed above.
2. **Warping posed and projected faces:** This is done in order to center the facial features example the lips, eyes, nose which makes the subsequent steps easier to process. To accomplish this, 68 unique facial landmarks(Figure 2.8, Figure 2.9) are spotted on each face and then rotation, scaling and shearing is applied get a centered image as shown in Figure 2.10. Example of centered and warped image is shown in Figure 2.7.

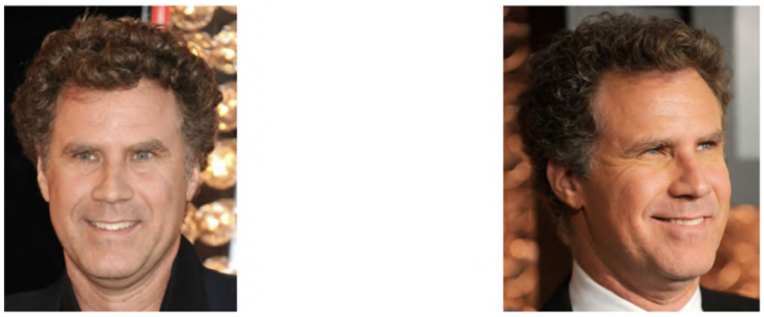


Figure 2.7.: Centered Face(left) and Posed Face(right) [20].

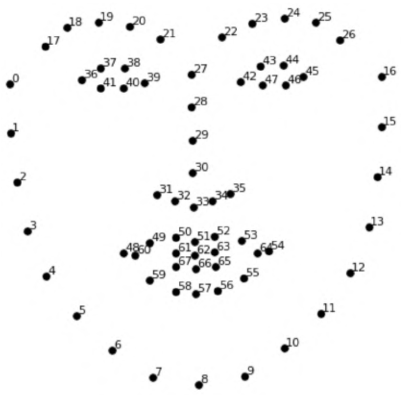


Figure 2.8.: 68 Facial Landmarks [20].

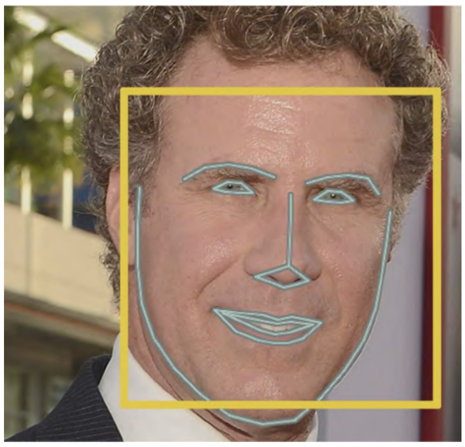
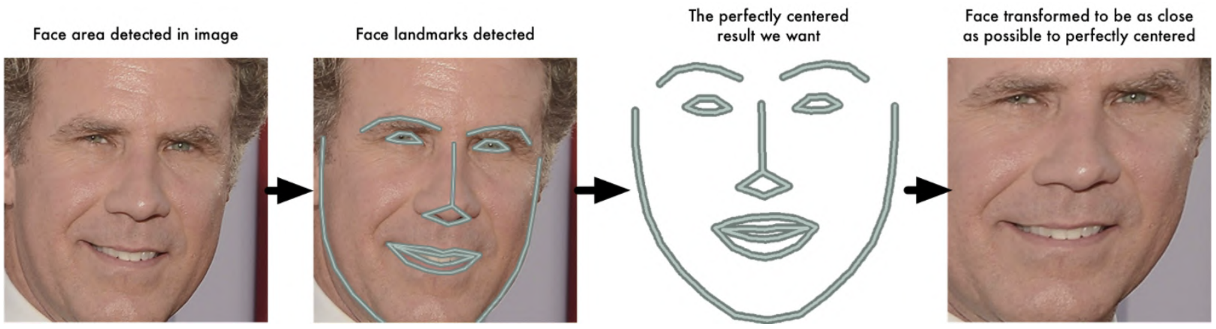


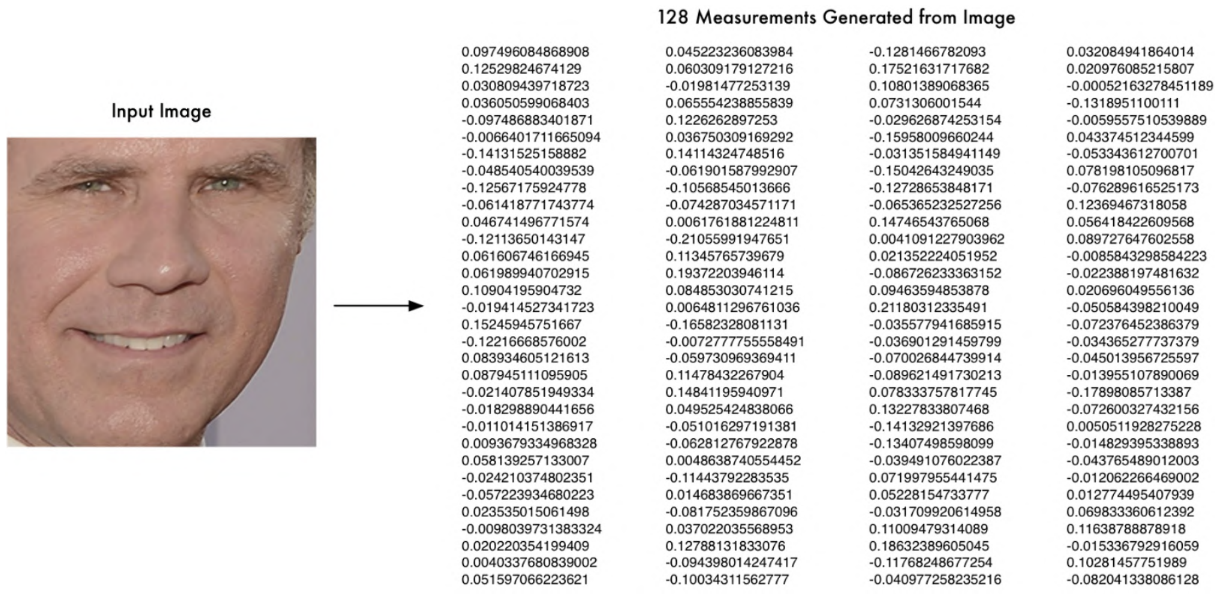
Figure 2.9.: Locating 68 face landmarks on an image [20].

3. **Finding face encodings:** The basic plan is to extract certain measurements from the face which could differentiate two different faces uniquely. So, 128 measurements are generated for every face which stand distinctive to each unique face (Figure 2.11). For different pictures of the same individual, these 128 measurements should be


Face area detected in image
Face landmarks detected
The perfectly centered result we want
Face transformed to be as close as possible to perfectly centered
Figure 2.10.: Image warping to center the facial features [20].

approximately the same. This is achieved by training a deep learning CNN model.

4. **Finding the correct prediction from the database:** The last step is to compare these encodings to the ones already stored in the database. The one which gives the highest confidence during the match is taken as the predicted outcome for the particular test case.



0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014
0.12529824674129	0.060309179127216	0.17521631717682	0.020976085215807
0.030809439718723	-0.01981477253139	0.10801389068365	-0.00052163278451189
0.036050599068403	0.065554238855839	0.0731306001544	-0.1318951100111
-0.097486883401871	0.1226262897253	-0.029626874253154	-0.0059557510539889
-0.0066401711665094	0.036750309169292	-0.15958009660244	0.043374512344599
-0.14131525158882	0.14114324748516	-0.031351584941149	-0.053343612700701
-0.048540540039539	-0.061901587992907	-0.15042643249035	0.078198105096817
-0.12567175924778	-0.10568545013666	-0.12728653848171	-0.076289616525173
-0.061418771743774	-0.074287034571171	-0.065365232527256	0.12369467318058
0.046741496771574	0.0061761881224811	0.14746543765068	0.056418422609568
-0.12113650143147	-0.21055991947651	0.0041091227903962	0.089727647602558
0.061606746166945	0.11345765739679	0.021352224051952	-0.0085843298584223
0.061989940702915	0.19372203946114	-0.086726233363152	-0.022388197481632
0.10904195904732	0.084853030741215	0.09463594853878	0.020696049556136
-0.019414527341723	0.0064811296761036	0.21180312335491	-0.050584398210049
0.15245945751667	-0.16582328081131	-0.035577941685915	-0.072376452386379
-0.12216668576002	-0.0072777755558491	-0.036901291459799	-0.034365277737379
0.083934605121613	-0.059730969369411	-0.070026844739914	-0.045013956725597
0.087945111095905	0.11478432267904	-0.089621491730213	-0.013955107890069
-0.021407851949334	0.14841195940971	0.078333757817745	-0.17898085713387
-0.018298890441656	0.049525424838066	0.13227833807468	-0.072600327432156
-0.011014151386917	-0.051016297191381	-0.14132921397686	0.0050511928275228
0.0093679334968328	-0.062812767922878	-0.13407498598099	-0.014829395338893
0.058139257133007	0.0048638740554452	-0.039491076022387	-0.043765489012003
-0.024210374802351	-0.11443792283535	0.071997955441475	-0.012062266469002
-0.057223934680223	0.014683869667351	0.05228154733777	0.012774495407939
0.023535015061498	-0.081752359867096	-0.031709920614958	0.069833360612392
-0.0098039731383324	0.037022035568953	0.11009479314089	0.11638788878918
0.020220354199409	0.12788131833076	0.18632389605045	-0.015336792916059
0.0040337680839002	-0.094398014247417	-0.11768248677254	0.10281457751989
0.051597066223621	-0.10034311562777	-0.040977258235216	-0.082041338086128

Figure 2.11.: 128 unique encodings [20].

2.3. Hand Detection and Tracking

[29, 30, 31, 32, 33] Hand detection and tracking can be utilized in various fields such as: augmented reality, hand gesture control or sign language, to convey information to people suffering from vocal and hearing impairment. MediaPipe, an open source framework capable of running on cross platforms provides an adaptable and customisable Machine Learning (ML) solution to predict and track a human hand skeleton through MediaPipe Hands by using a single RGB input. It is primarily used in mobile GPU devices to achieve real time performance with high quality by deploying an ML pipeline comprising of multiple models working together. The overall model specifications are as follows:

Model Type: Convolutional Neural Network(CNN).
Model Architecture: 2 stage neural network pipeline with an SSD and regression model working upon the cropped region.
Model Input: An image or video of arbitrary size.
Model Output: A list of detected hands(predicts multiple hands) each comprising of the following:

- 21 3D hand knuckle landmarks.
- A floating scalar value determining the probability of hand presence in the input frame.
- A floating scalar value representing the handedness of the predicted hand: left or right hand.

The pipeline as shown in Figure 2.12 consists of a 2 stage process using 2 models as follows:

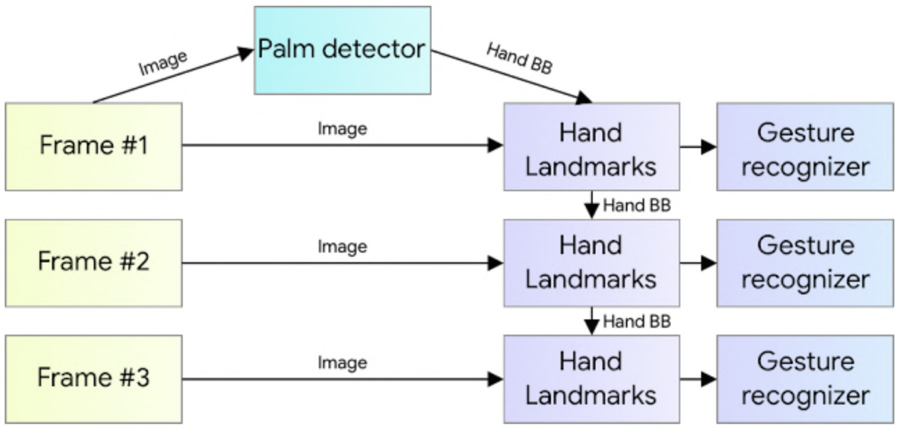


Figure 2.12.: Overview of hand detection and tracking pipeline [32].

2.3.1. Palm Detector

Palm detection uses a model known as ***BlazePalm*** which accepts an RGB image as an input and outputs a bounding box around the hand. BlazePalm detects a palm location using single shot detector(SSD) concept. The model works for a variety of hand sizes and is also able to identify occluded and self occluded hands. With the above model, an average precision of approx 95.7% has been observed. The model architecture of BlazePalm is shown in Figure 2.13.

Model Type: Convolutional Neural Network(CNN).
Model Architecture: Single Shot Detector model.
Model Input: A image/video frame of size represented as 192x193x3 pixels of channel order: RGB
Model Output: Tensor of type float comprising of predicted embeddings(the hand bounding box).

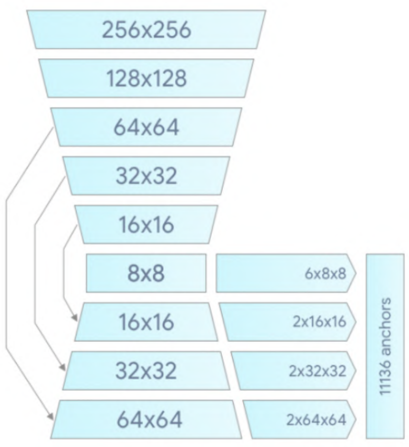


Figure 2.13.: BlazePalm model architecture [32].

2.3.2. Hand Landmark Detector

After executing the palm detection model over the entire image, a cropped part of the image consisting of just the detected palm is returned which becomes the input to the next model, ***BlazeHand***, which detects the 21 hand landmarks via regression, essentially a direct prediction of the coordinates. It is robust to partially occluded hands and acquires a consistent internal hand posture. This process of passing just the detected hand instead of complete image, reduces the overall computation, thus making the entire process faster and efficient. The 21 handlandmarks are shown in Figure 2.14.

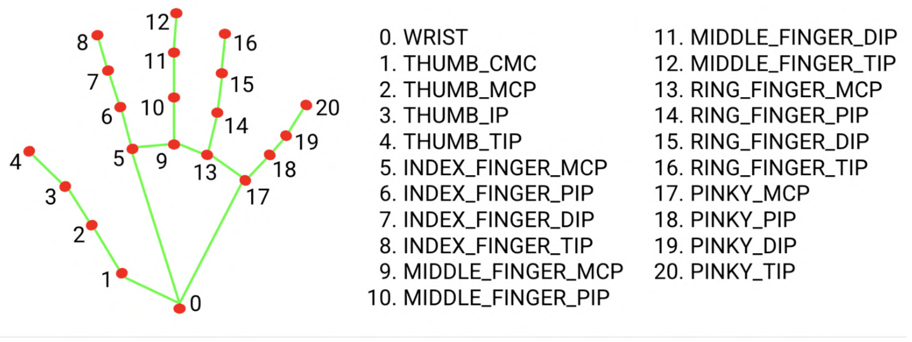


Figure 2.14.: BlazeHand output: 21 handlandmarks [32].

Model Type: Convolutional Neural Network(CNN).
Model Architecture: Regression model.
Model Input: Cropped part of the original image/video frame which consists of the detected palm represented as 224x224x3 pixels of channel order: RGB.
Model Output: The model output is as follows:

- 21 3D hand knuckle coordinates.
- A floating flag value determining the probability of the hand present in the given input image frame.
- A floating flag value indicating the presence of left or the right hand also known as handedness.

2.3.3. MediaPipe Implementation

The Figure 2.15 displays the implementation of the ML pipeline for hand detection and tracking. MediaPipe optimises the whole process even a step further by running BlazePalm only when required which saves a lot more of computational time. This is established by deducing the hand location in the successive video frames from the precisely localized hand keypoints, thus obliterating the requirement to run the palm detector model in each frame. Only if the confidence threshold falls below a certain limit, if it is the first frame or if the hand is missing from the frame, then the palm detector is put into action.

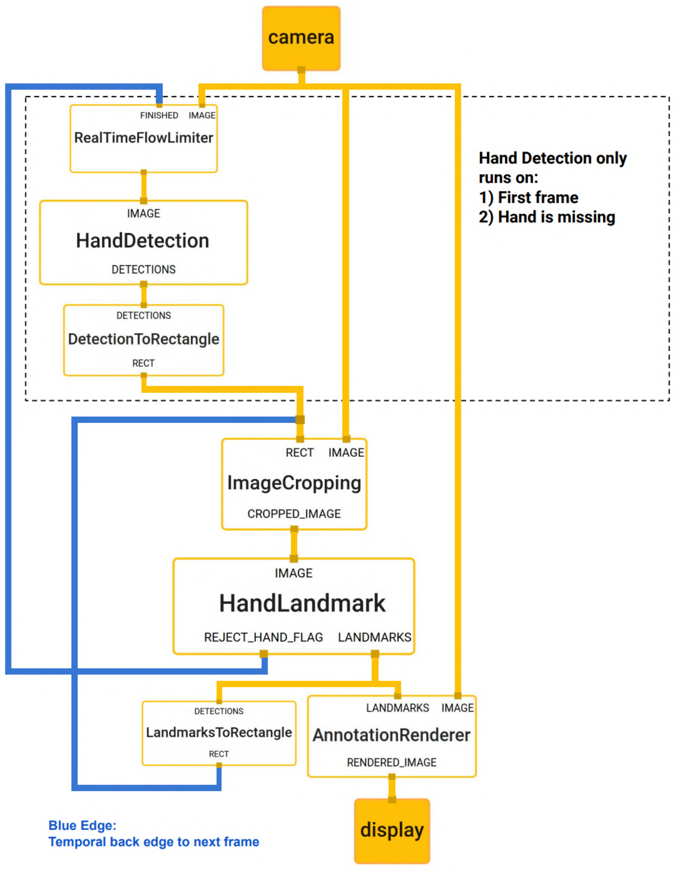


Figure 2.15.: MediaPipe Implementation of Hand detection and tracking [32].

3. Methodology

This section of the thesis covers the initial plan of action and methods chosen to approach the problem statement to achieve the required objectives. The elaboration of these methods and their implementation are covered in the next section, Chapter 4: Implementation. The general flow of the thesis from start till end has been summarized visually and intuitively in the Figure 3.1 (also mentioned in the section 1.4: Proposed Solution).

3.1. Software Requirement Specifications

Since this project is focused on a complete software development, the standard principles of software development has been followed as accurately as possible to pen down the software requirement specifications[34]. The specifications were drafted after obtaining the thesis objective and conducting the background research.

User Requirements

1. The attendance tracker system software shall be able to log students in/out of the laboratory by recognising face and a gesture.
2. The username, matriculation number, project name and the time of individual login/logout should be saved.
3. The software shall have an interactive user interface with minimum physical contact.
4. The software shall have system preferences controlled only by the admin.
5. The software shall allow the admin to create and delete a user through the user interface.
6. The attendance logger shall also have additional features to change the minimum time duration between one login and logout cycle, change the lab closing time and select a login mode.
7. The complete software application shall be able to run on an small, low cost computer.

System Requirements

Functional System Requirements

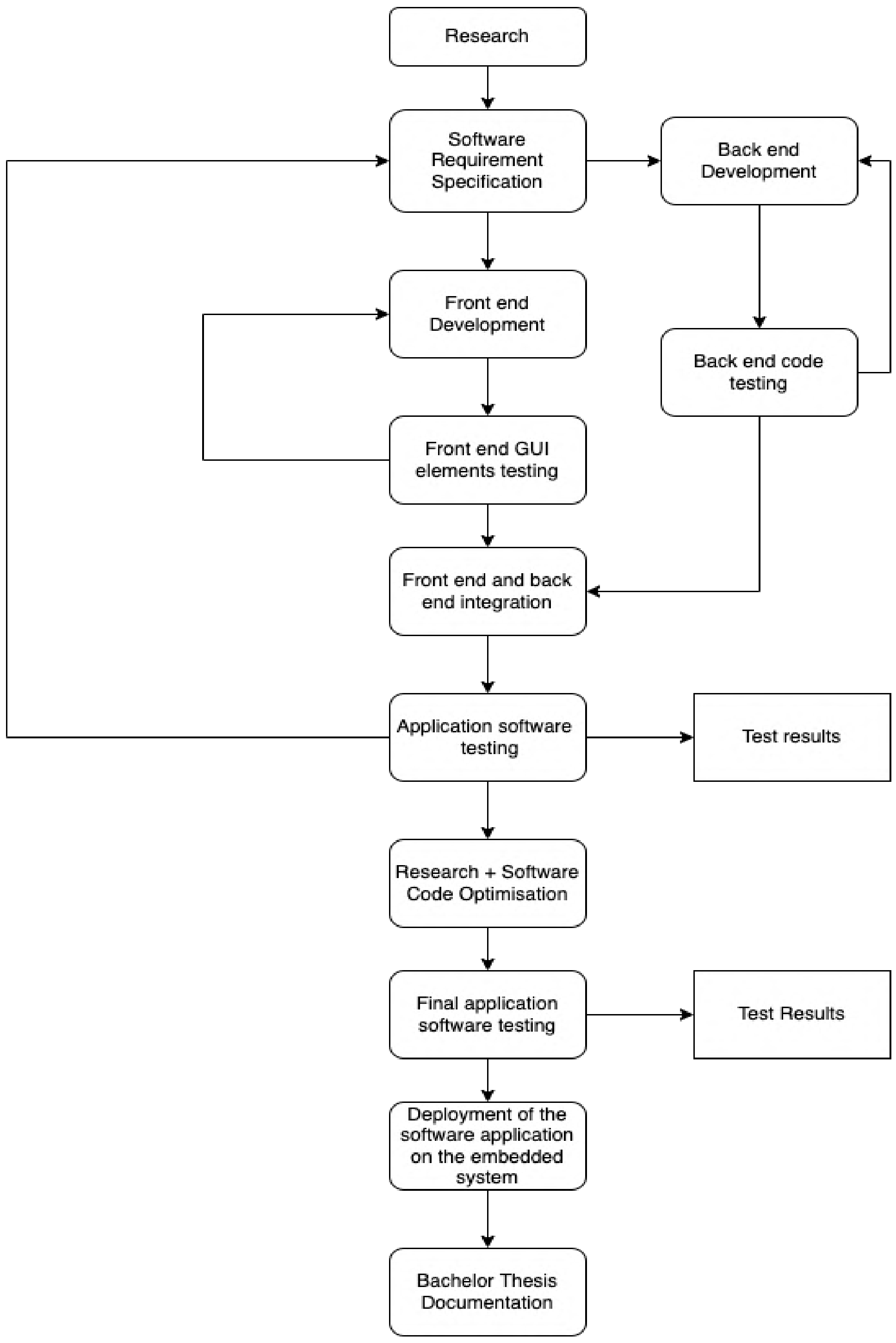

Research
Software Requirement Specification
Back end Development
Front end Development
Back end code testing
Front end GUI elements testing
Front end and back end integration
Application software testing
Test results
Research + Software Code Optimisation
Final application software testing
Test Results
Deployment of the software application on the embedded system
Bachelor Thesis Documentation

Figure 3.1.: General Thesis Workflow.

1. The attendance tracker software shall have an interactive GUI and backend to control the logic running on an embedded system for ex: raspberry pi.
2. The user shall interact with the software GUI through a touchscreen and a camera connected to the raspberry pi.
3. A login window shall handle the login/logout of the users utilising computer vision and save the name, matriculation number, project name and the timestamps of login/logout in a file.
4. The system shall allow authorised users to log in/out of the lab only after the face has been recognised and a correct gesture indicated.
5. The processing for unauthorised users shall be discarded.
6. The system shall ignore subsequent login/logout gestures.
7. The software shall not allow a person to logout prior to the expiry of the minimum time duration between login and logout.
8. The program shall not allow a user to log out before logging in.
9. The system shall not allow a user to log in after the lab closing hours.
10. The gesture recognition shall not work for unauthorised users.
11. The system preferences option shall be admin controlled through a password.
12. Under system preferences, the system shall provide options to register a new user, delete user and change the parameters.
13. The create new user window shall allow admin to register and save the details of a new user such as: name, matriculation number, project name and the face encodings in a file.
14. The delete user window shall allow the admin to delete a particular registered user from the database.
15. An autologger parameters window would encapsulate the three settings for the software: login/logout duration, lab closing time, login mode.
16. The login/logout details along with the name and timestamps shall be stored in a file which essentially becomes the digital attendance sheet.
17. If a user forgets to logout, the system logs the person out automatically after the lab closing hours.

Non Functional System Requirements

1. The system is supposed to run indoors on the embedded system, installed at the entrance of the robotics laboratory.
2. The software saves only the face encodings of the person while registering as a new user which is not in a human readable format, thus protecting the privacy of the individual registered user.
3. The system runs locally and not on any external commercial server/cloud.
4. Desirable features: "Active users window to show the active logins currently in the lab and "About software" to give a short description about the developer and how to use the software.

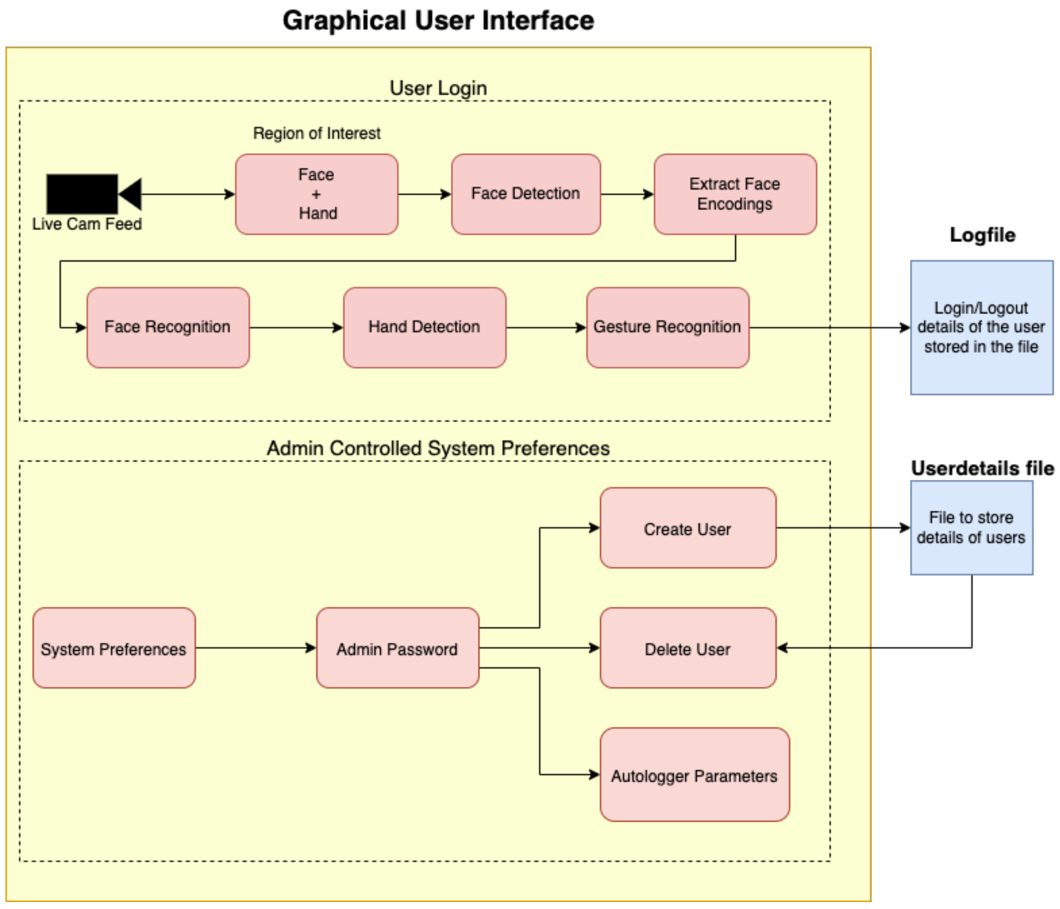


Figure 3.2.: Software workflow diagram as proposed solution.

The above shown Figure 3.2 illustrates the software workflow diagram as proposed solution comprising of 2 blocks of logic being executed through the user interface namely: user login and admin controlled system preferences.

3.2. Hardware Setup

Hardware setup has been mentioned in the methodology as it is required to fulfill the non functional requirement of the attendance tracker system of running the software on an embedded system. The device consists of 3 essential hardware:

- **Raspberry Pi model 4B:** An embedded device needs a processor/controller to process the instructions set to fulfill a given task. In this thesis Raspberry Pi (RPi) 4B as shown in Figure 3.3 is most desirable as an embedded systems computer because it's compact, lightweight, cheap and has enough capabilities to run the AI models and make the whole software run smoothly. The model specifications are shown in the Table 3.2 [35].

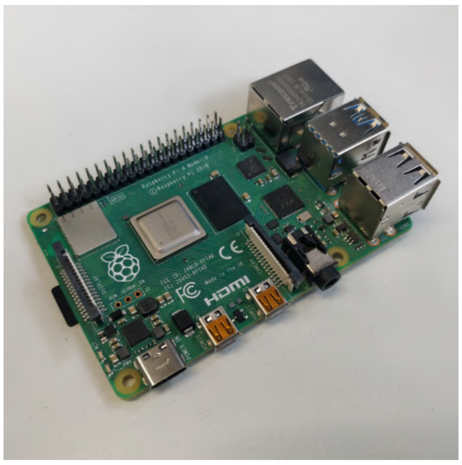


Figure 3.3.: Raspberry Pi model 4B.

Table 3.1.: Raspberry Pi Touch Display Specifications[36].

Model	Raspberry Pi Touch Display
Display Size(diagonal)	7-inches
Display Format	800(RGB) x 480 pixels
Active Area	154.08mm x 85.92mm
Touch Panel	Multi-touch capacitive touch panel
Assembly Module Size	192.96mm x 110.76mm
Power,Connections	5V through RPi GPIO, Ribbon Cable Connection to DSI port in RPi

- **Raspberry pi camera v1.3:** A camera is very essential in this project to run face detection, face recognition as well as gesture recognition. This particular camera as shown in

Figure 3.4 is chosen as it is cheap and comes with a decent resolution to fulfill the purpose of the attendance tracker system. The model specifications are given in the Table 3.3 [37].

- **Raspberry pi touchscreen:** The touchscreen is to assist user interaction with the GUI essentially running the complete application. This particular screen as shown in Figure 3.5 is chosen as it supports touch functionality and is quite inexpensive. The model specifications are shown in Table 3.1 [36].

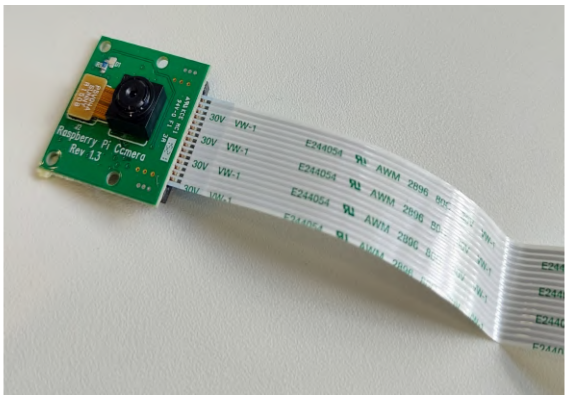


Figure 3.4.: Raspberry Pi Camera.

Figure 3.5.: Raspberry Pi Touchscreen.

Other hardware requirements are as follows:

- Secure Digital Memory Card (SD Card) - 8 Giga Bytes (GB)
- Power Adapter
- Power Cable
- A uniquely designed 3D printed case to incorporate all the electronics and make it "all in one portable product ready to be placed anywhere".

These hardware combined make it a lightweight, portable standalone embedded system. The circuit schematic for the embedded system is shown in Figure 3.6.

3.3. Software Setup

As the project aims to deliver a software package, nevertheless still on an Research and Development (R&D) stage, so Python suits best for developing the back-end as it is very popular for rapid prototyping purposes and in developmental stages. Python Tkinter library has been particularly used for the GUI as it is a very popular, robust enough to fulfill the

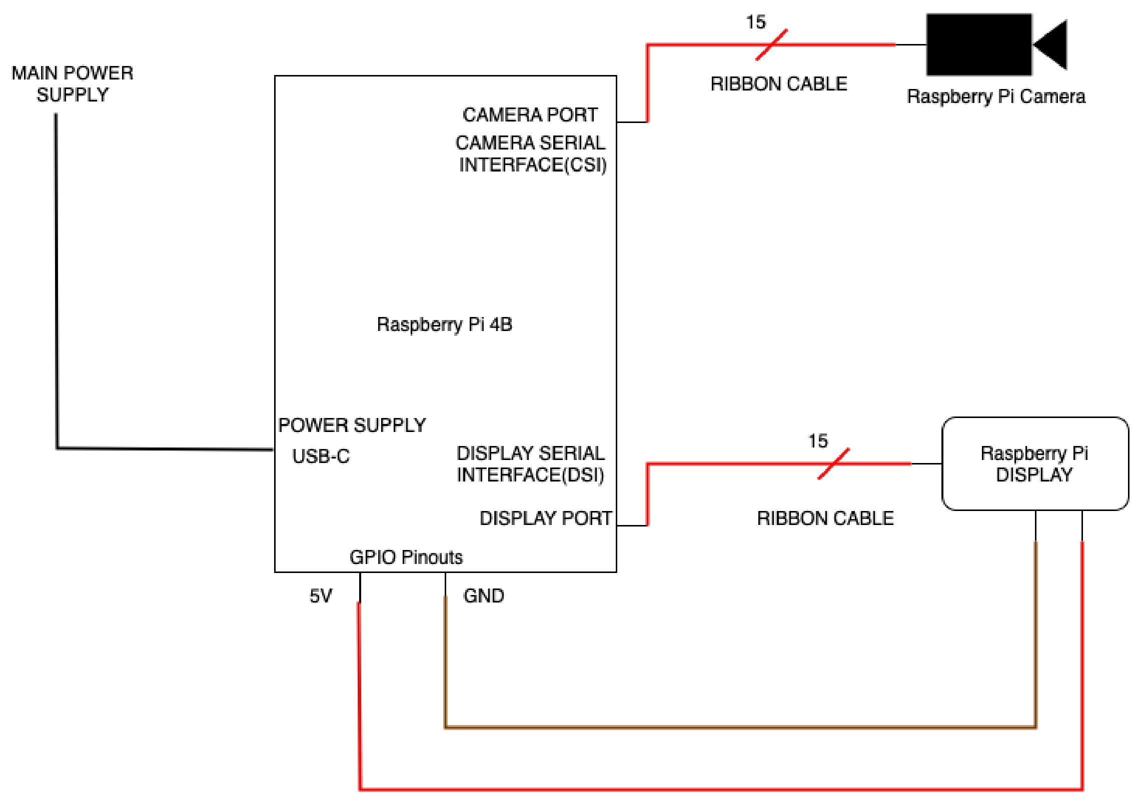


Figure 3.6.: Circuit Schematic Diagram. Drawn using draw.io[18].

requirements for this project as well as makes the front-end and back-end integration simple.

The initial plan of action for software design is conveyed neatly through the UML diagrams. Unified Modeling Language diagram uses the power of graphical representation to describe the function of the complete system, here in this case: the attendance tracker software in a very comprehensive manner to understand the overall functionalities in detail with the purpose to visualize its main actions in a flow chart format by making use of various graphical symbols, each conveying a special meaning thus adding to the elaborated understanding of system capabilities. The design of such diagrams is mainly used in software engineering as it provides an extensive structural layout plan for the software engineers to carry out their tasks in a well organised format.

For this thesis, to explain the functionalities of the attendance tracker system, **"UML Activity diagram"** has been chosen which explains the workflows and the step-wise actions to be taken at the desired time.

The following graphical symbols convey a special meaning used during the construction of the diagram:

- **Black Circle:** Starting node of the UML activity diagram.
- **Encircled Black Circle:** End state of the UML activity diagram.
- **Diamonds:** Decision node.
- **Arrows:** The control flow direction of the UML activity diagram.
- **Rounded rectangle:** Action state.

The software development (back-end) has been divided into 3 stages: User Authentication, Hand Detection and Attendance system which has been discussed below. The intention to choose face detection, face recognition and hand detection as the final approach to solve this problem has been mentioned in section 1.4: proposed solution of Chapter 1: Introduction. The GUI layout has been covered in Chapter 4: Implementation.

- **User Authentication:** The user authentication UML activity diagram is shown in Figure 3.7, basically verifying the authentication of the personnel working in the laboratory by running a face detection and face recognition model. The python face-recognition library handles the face detection (based on HOG) and face recognition (based on training a deep convolutional neural network).
- **Hand Detection:** The hand detection UML activity diagram is shown in Figure 3.8, which helps in identifying the gesture to login (thumbs-up gesture) or logout (thumbs-down gesture) of the lab. This has been done by using a Palm Detector(BlazePalm model) and hand landmark detector(BlazeHand model).
- **Laboratory Attendance:** The lab attendance UML activity diagram[1] is shown in Figure 3.9, summarizing the main logic for handling the back-end part of the attendance tracker software. It starts with identifying the person and its gesture, obeying a certain set of conditional statements and finally writing the attendance details into a file.

[1]Please refer to these acronyms in the UML diagram: diff: difference, min: minimum, b/w: between

Table 3.2.: Raspberry Pi Specifications[35].

Model	Raspberry Pi 4 Model B
RAM	4GB
Processor	Quad Core 64-bit ARM-Cortex A72 @ 1.5GHz
Interfaces	– Bluetooth 5.0 – 1x SD Card – 2x Micro HDMI Ports – 2x USB2 Ports – 2x USB3 Ports – 1x RPi Camera Port(CSI) – 1x RPi Display Port(DSI) – 802.11 WLAN – 1x Gigabit Ethernet Port
GPIO Pins	Total Pins: 40 – Ground Pins: 8x – Power Pins: 2x 5V DC, 2x 3.3V DC – GPIO Pins: 28x with various interface options: upto 6x UART, 6x I2C, 5x SPI, 1x SDIO Interface, 1x DPI(Parallel RGB Display), 1x PCM, upto 2x PWM channels, upto 3x GPCLK Outputs
Power	USB-C power supply, delivering 5V at 3A
Operating Temperature	0 - 50 degree Celcius

Table 3.3.: Raspberry Pi Camera Specifications[37].

Model	Raspberry Pi Camera v1.3
Still Resolution	5MP
Weight	3 grams
Size	25x24x9 mm
Sensor, Sensor Resolution	OmniVision OV5647, 2592 × 1944 pixels
Connections	Ribbon Cable Connection to CSI Port in RPi

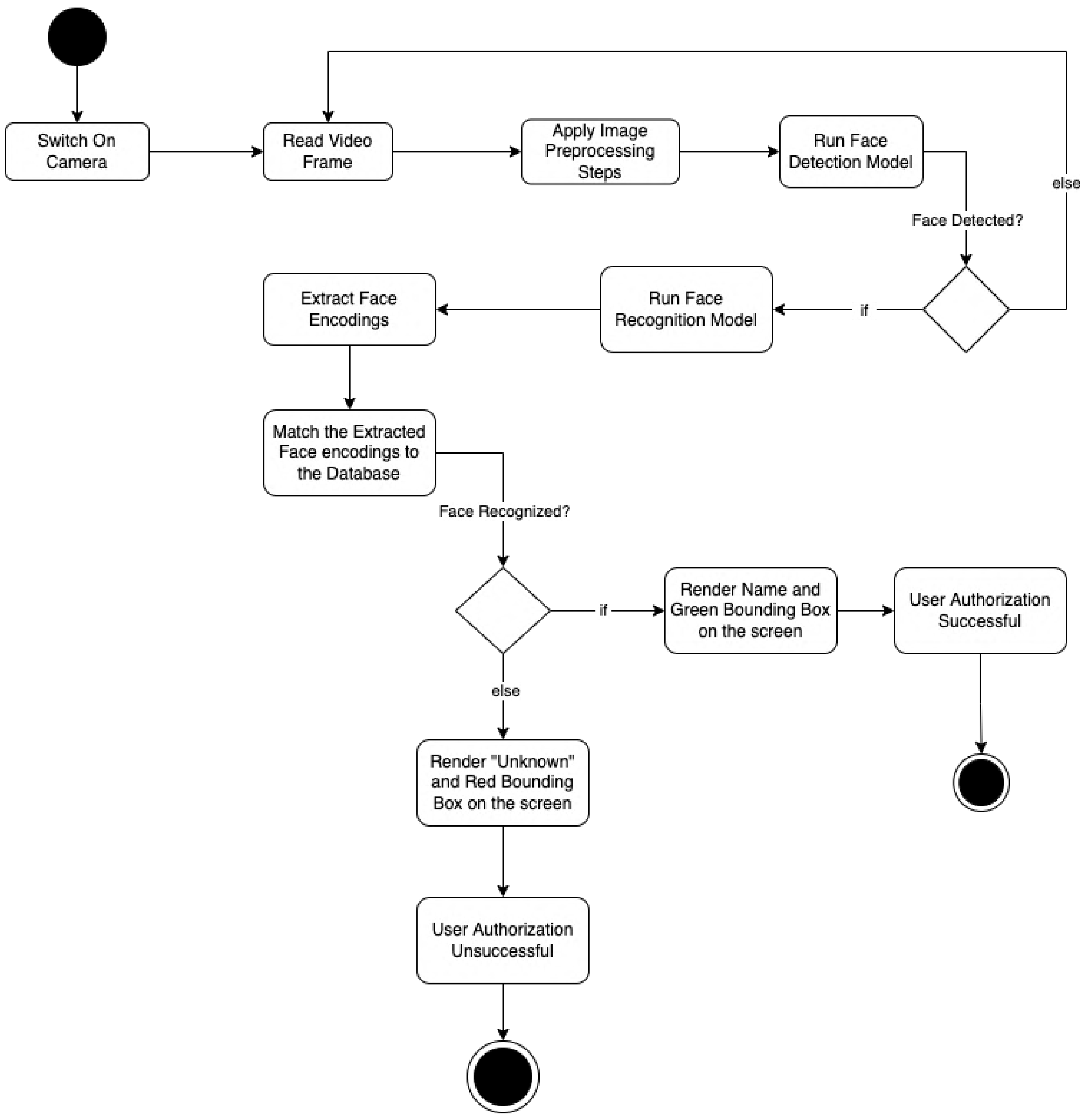

Switch On Camera
Read Video Frame
Apply Image Preprocessing Steps
Run Face Detection Model
else
Face Detected?
if
Run Face Recognition Model
Extract Face Encodings
Match the Extracted Face encodings to the Database
Face Recognized?
if
Render Name and Green Bounding Box on the screen
User Authorization Successful
else
Render "Unknown" and Red Bounding Box on the screen
User Authorization Unsuccessful

Figure 3.7.: User Authentication UML Activity Diagram.

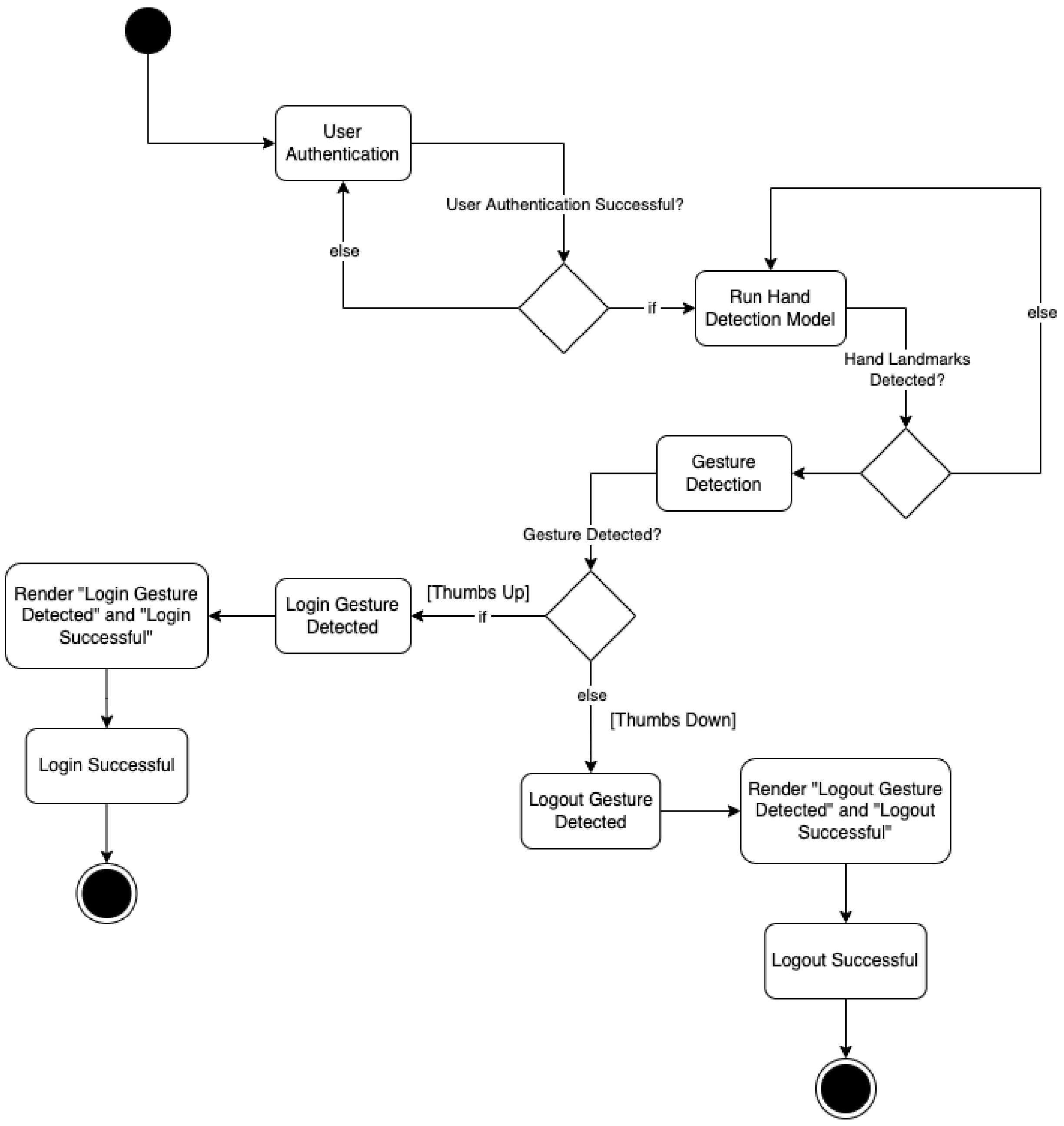


Figure 3.8.: Hand Detection UML Activity Diagram.

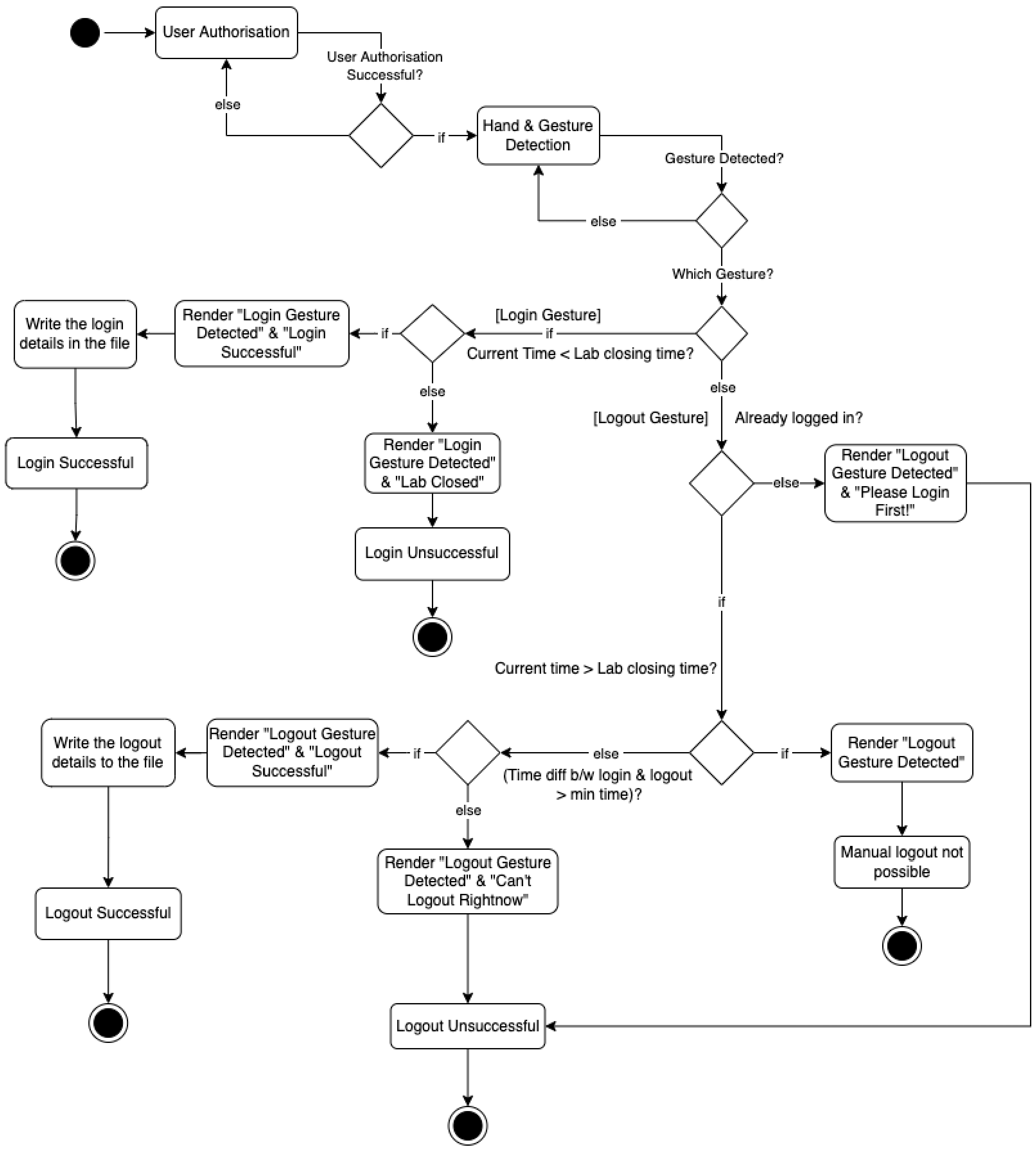

User Authorisation
User Authorisation Successful?
else
if
Hand & Gesture Detection
Gesture Detected?
else
Which Gesture?
[Login Gesture]
if
Current Time < Lab closing time?
Render "Login Gesture Detected" & "Login Successful"
if
Write the login details in the file
Login Successful
else
Render "Login Gesture Detected" & "Lab Closed"
Login Unsuccessful
else
[Logout Gesture]
Already logged in?
else
Render "Logout Gesture Detected" & "Please Login First!"
if
Current time > Lab closing time?
if
Render "Logout Gesture Detected"
Manual logout not possible
else
(Time diff b/w login & logout > min time)?
if
Render "Logout Gesture Detected" & "Logout Successful"
Write the logout details to the file
Logout Successful
else
Render "Logout Gesture Detected" & "Can't Logout Rightnow"
Logout Unsuccessful

Figure 3.9.: Lab Attendance UML Activity Diagram.

4. Implementation

This section covers in-depth detailed stages of implementation of the methodology used to build this software. It has been further bifurcated into 2 segments: front-end implementation and back-end implementation with GUI integration.

4.1. Front-end Implementation

The front-end of this attendance tracker software has been built using Python Tkinter library containing a total of 12 windows (refer to Figure 4.1). The basic idea while coding the GUI is to keep the design minimal rather focusing on the visual part, user friendly, fulfill the required functionality of the system as well as keep it as contact free as possible. To keep the design minimal for example: same background has been used throughout, large buttons have been used without much complexity making the system intuitive and fulfill the required functionality, substitutes have been made to prevent too much of typing by introducing spinboxes and dropdown menus thus making it a lot more contact free rather than typing in data using a virtual keyboard in a very small screen making it less user friendly as well.

The library constructs the GUI using the fundamental building blocks called "Widgets" within its ***main window a.k.a root window*** having same size as the raspberry pi touch display. The widgets used in this software are: Buttons, Label, Text, Entry, Spinbox, Scrollbar, Listbox and OptionMenu. Pressing a button triggers the execution of an event as per the definition in the widget itself. Each window or an event is defined as a function which gets called upon firing of an event on a button press. The GUI works within one main window rather creating a new one for each event by destroying the widgets present in the previous window first and then creating the widgets for the new window using `.destroy()` method attached to any widget. For this reason, all the widgets have been defined as `global` variables so that they are accessible in the entire scope. The back, home and next button aids a user to toggle between various windows effortlessly. Back button switches to the immediate previous window, next button toggles to the immediate ahead window whereas the home button toggles a user to the window "HOME" (Figure 4.1) irrespective of its position in the interface.

Figure 4.2 is the window "HOME" of the user interface. It contains 3 buttons (back,home and next) and 3 labels incorporating the robotics laboratory logo and the text respectively. It is essentially the starting point of the system a.k.a home page which is toggled upon pressing the home button. The syntax of a simple Tk (Tkinter) window is shown below:

```
from tkinter import *
```

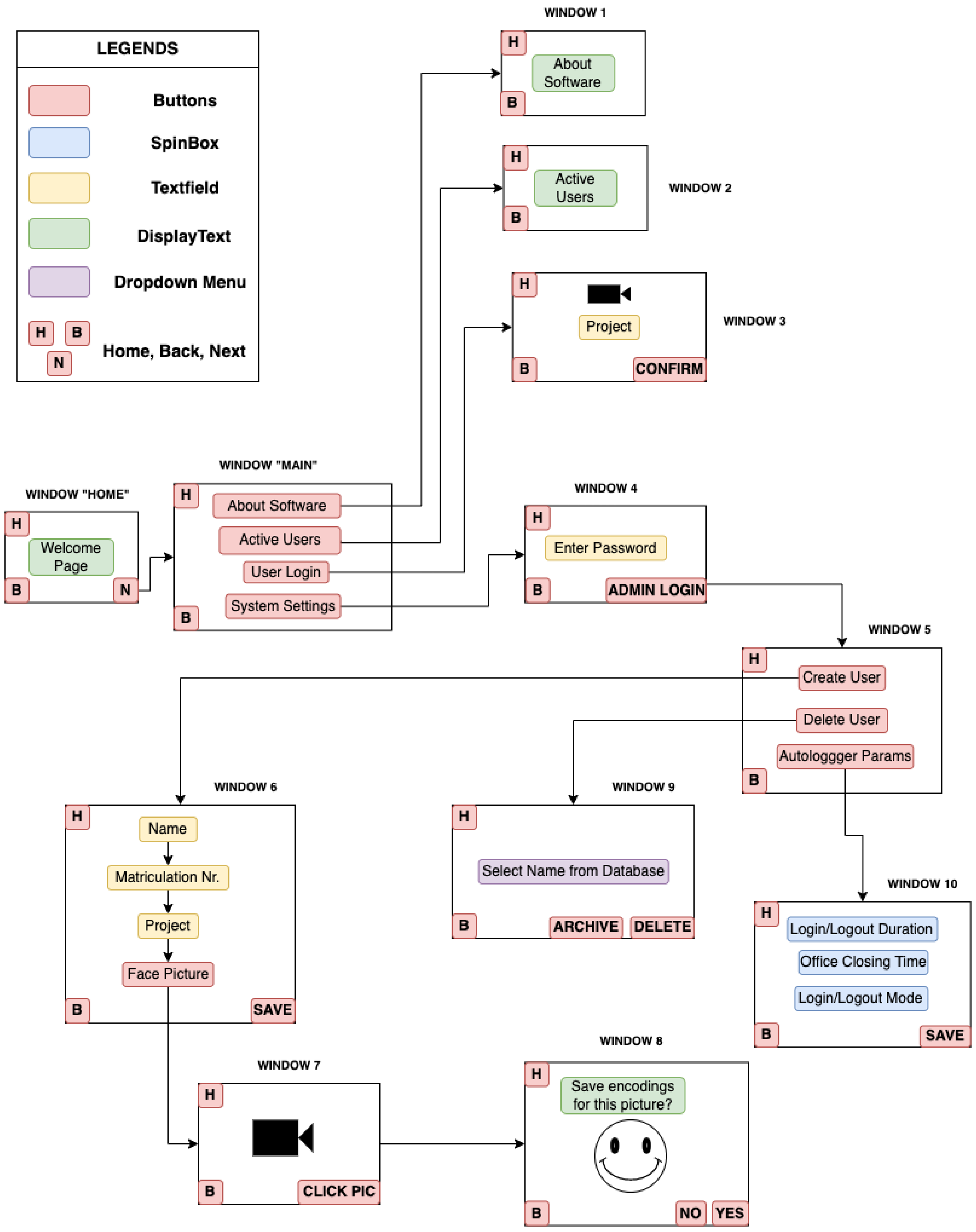
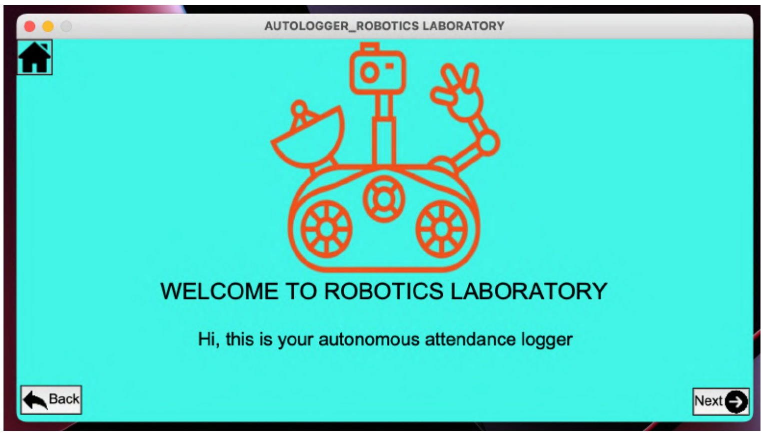


Figure 4.1.: Detailed GUI Layout for Attendance Tracker Software.

```
root = Tk()
button1 = Button(root, option = value,....)
label1 = Label(root, option = value, ...)
label1.pack()
button1.place(x = 734,y = 378)
```

Listing 4.1: Tkinter button and label widget syntax

`root` refers to the parent window which assimilates all the widgets. `root = Tk()` basically creates an instance of Tkinter frame which displays the window root and controls all the widgets defined in the root window. A widget is defined by creating its instance and attaching it to the `root`. `Options` allow the developer to set different parameters for the associated widget such as the font size & style, text, background color, border width, command, etc.



WELCOME TO ROBOTICS LABORATORY

Hi, this is your autonomous attendance logger

Figure 4.2.: Window "HOME" of the attendance tracker software.

The widgets are placed in the window by using methods called: `.pack()` and `.place()`. The `.pack()` method places the widgets in the center of the window while the `.place()` method on the contrary places the widgets according to the 2D coordinates specified by the developer in the arguments.

Firing the Next button on the window Home toggles GUI to window "MAIN" as shown in Figure 4.3 which consists of 6 buttons: home, back, Active Users, User Login/Logout, System Preferences, About Autonomous Attendance Logger. This window is of prime essence as it paves the way to the key functionalities of this software.

Pressing the Active Users button display the status of the currently logged in users in the laboratory as shown in Figure 4.4. It consists of 2 buttons, label, scrollbar and a listbox. The syntax of a scrollbar and a listbox is given below:

```
from tkinter import *
root = Tk()
myscrollbar = Scrollbar(root, orient = VERTICAL,option = value,....)
myscrollbar.place(x = 770,y = 40)
mylistbox = Listbox(root, yscrollcommand = myscrollbar.set,option = value, ...)
```

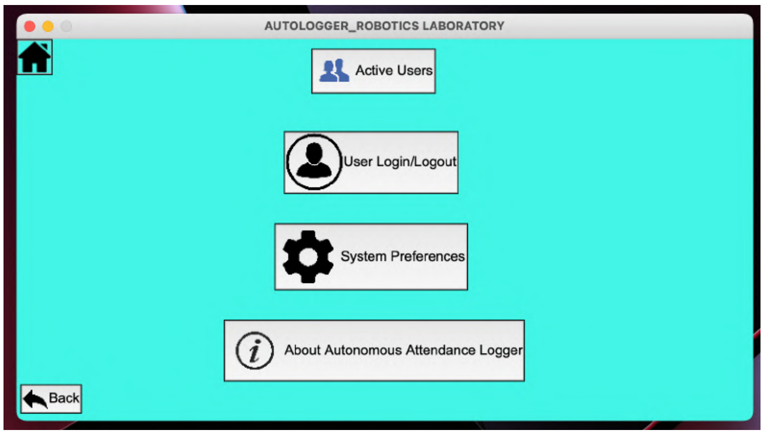

AUTOLOGGER_ROBOTICS LABORATORY
Active Users
User Login/Logout
System Preferences
About Autonomous Attendance Logger
Back

Figure 4.3.: Window "MAIN" of the software with four options.

```
mylistbox.place(x = 50, y = 40)
```

Listing 4.2: Scrollbar and listbox syntax

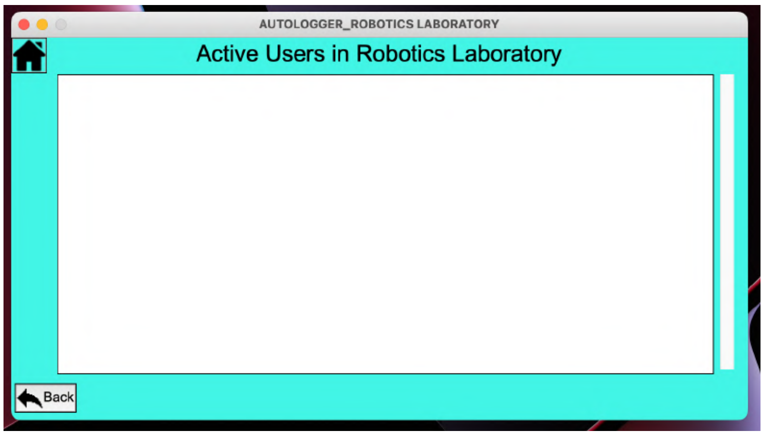

AUTOLOGGER_ROBOTICS LABORATORY
Active Users in Robotics Laboratory
Back

Figure 4.4.: Empty listbox and a scroll bar with no active users (Window 2).

The parameter `orient = VERTICAL` in scrollbar widget sets the scrollbar in the vertical position while the parameter `yscrollcommand = myscrollbar.set` attaches the scrollbar the the listbox. As discussed above, there are many other options which can be used by the programmer accordingly as key-value pairs just seperated by commas.

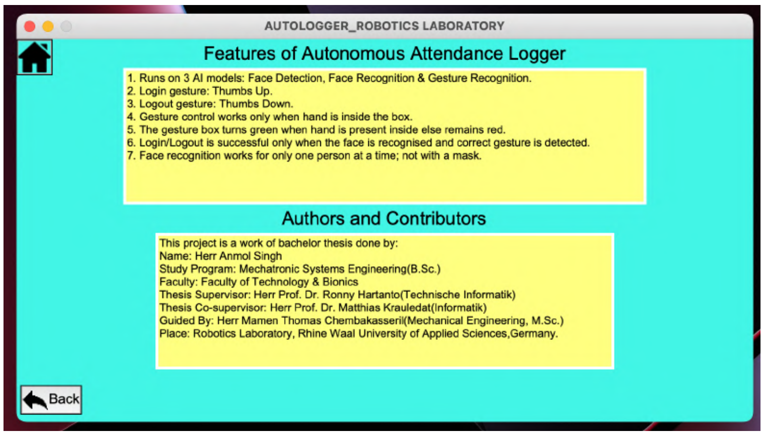
The About Autonomous Attendance Logger in Figure 4.5 shows the features of the software to be conveyed to the end user and the details of the author. This has been accomplished using `Text` and `Label` widget. The syntax and usage of `Text` widget is shown as:

```
from tkinter import *
root = Tk()
feature1 = '2. Login gesture: Thumbs Up.\n'
```

```python
feature2 = '3. Logout gesture: Thumbs Down.\n'
text1 = Text(root, font = ('Arial',12), height = 10,bg='#ffff80')
text1.insert(INSERT,feature1)
text1.insert(INSERT,feature2)
text1.pack()
```

Listing 4.3: Text widget syntax

`feature1` and `feature2` are strings which have been inserted by the Text widget using the `.insert()` method. Moreover, the above snippet shows an example of a few parameters such as font style, font size, height and background color set by the developer for the corresponding widget.



Features of Autonomous Attendance Logger

1. Runs on 3 AI models: Face Detection, Face Recognition & Gesture Recognition.
2. Login gesture: Thumbs Up.
3. Logout gesture: Thumbs Down.
4. Gesture control works only when hand is inside the box.
5. The gesture box turns green when hand is present inside else remains red.
6. Login/Logout is successful only when the face is recognised and correct gesture is detected.
7. Face recognition works for only one person at a time; not with a mask.

Authors and Contributors

This project is a work of bachelor thesis done by:
Name: Herr Anmol Singh
Study Program: Mechatronic Systems Engineering(B.Sc.)
Faculty: Faculty of Technology & Bionics
Thesis Supervisor: Herr Prof. Dr. Ronny Hartanto(Technische Informatik)
Thesis Co-supervisor: Herr Prof. Dr. Matthias Krauledat(Informatik)
Guided By: Herr Mamen Thomas Chembakasseril(Mechanical Engineering, M.Sc.)
Place: Robotics Laboratory, Rhine Waal University of Applied Sciences,Germany.

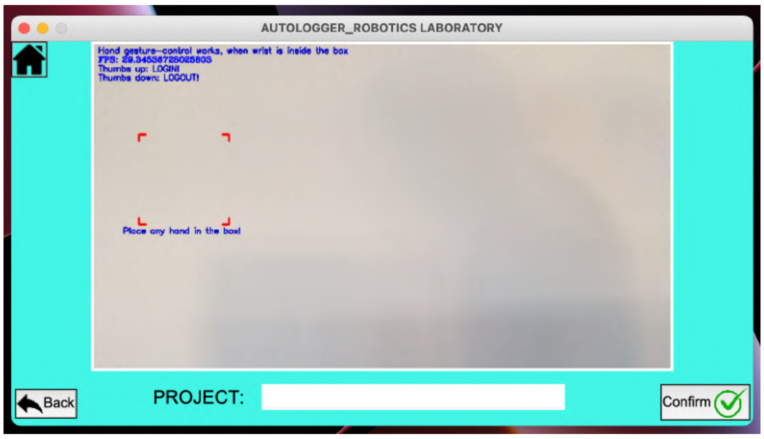
Figure 4.5.: About the software (Window 1).

The User login/logout which manages the attendance system of the students working in the laboratory opens up window 3 as shown in Figure 4.1 running a live camera feed accompanied by 3 buttons: confirm, home and back, a Label widget and an Entry widget. The corresponding GUI is displayed in Figure 4.6 . Entry widget allows user to input data via keyboard. An example snippet of Entry widget is shown below:

```python
from tkinter import *
root = Tk()
entrywidget = Entry(root, width=35, borderwidth = 2, bg="white", fg = "black")
entrywidget.place(x = 320,y = 370)
```

Listing 4.4: Entry widget syntax

```python
from tkinter import *
from PIL import ImageTk, Image
import cv2
root = Tk()
video = Label(root)
video.pack()

global cap
```


AUTOLOGGER_ROBOTICS LABORATORY

Hand gesture-control works, when wrist is inside the box
FPS: 29.34536728025803
Thumbs up: LOGIN!
Thumbs down: LOGOUT!

Place any hand in the box!

PROJECT:

Figure 4.6.: User login/logout (Window 3).

```python
cap = cv2.VideoCapture(0)

while True:
    ret,frame = cap.read()
    cvimage = cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA)
    cvimage = cv2.resize(cvimage, (int(aspectratio*350),350))
    img = Image.fromarray(cvimage)
    imgtk = ImageTk.PhotoImage(image=img)
    video.config(image = imgtk)
    root.update()
```

Listing 4.5: Importing webcam in Tkinter syntax

The above snippet shows how to import and use a webcam in a Tkinter window. It is achieved by reading an input frame from the camera by creating an object of `VideoCapture` first, then set up an infinite while loop in which the input frame is read by `.read()` from the above created object. Subsequently necessary preprocessing steps of color channel conversion and resizing are applied using `.cvtColor()` and `.resize()` methods respectively, then the input camera frame is converted into an `Image` using `Image` class from PIL library which is then converted into a `PhotoImage` by `ImageTk` class. Finally the PhotoImage is displayed on Tkinter window through `video` label. This label is updated continously which results in uninterrupted video frames. Additionally, some text and lines have been rendered on the webcam screen using the `.putText()` and `.line()` methods of OpenCV respectively. An example code snippet is shown below.

```python
import cv2
cv2.putText(frame,'Hello',(10,80),fontFace= cv2.FONT_HERSHEY_SIMPLEX,fontScale=0.6,color
    =(200,0,0),thickness = 2)
cv2.line(imageframe,(100,200),(115,200),(0,0,255),3)
```

Listing 4.6: Syntax of rendering text and lines using OpenCV

System Preferences is password controlled by the admin. So pressing the button opens up an authorisation page (window 4 of Figure 4.1) which prompts the admin to enter the password

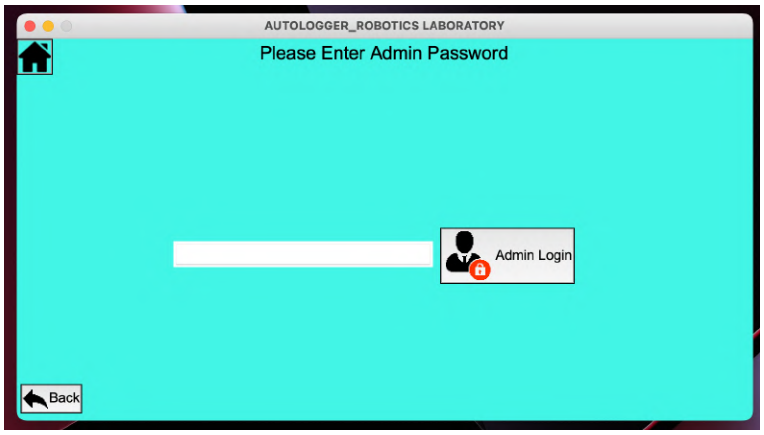


Figure 4.7.: Admin controlled system preferences (Window 4).

as presented in Figure 4.7. This window contains a label, 3 buttons: Home, Back and Admin Login along with an entry widget to enter the password. The Admin Login button leads to the System Preferences(window 5 in Figure 4.1) once the authorisation is successful. This window incorporates a label and 5 buttons: Home, Back, Create User, Delete User and Autologger Parameters. The icons associated with these buttons are defined in the button definition itself.

```python
from tkinter import *
from PIL import ImageTk, Image
root = Tk()

createphoto = PhotoImage(file = "createuser.png")
createphotoimage = createphoto.subsample(6,6)

create_user_button = Button(root, text='Create User', image = createphotoimage, compound =
    LEFT)
create_user_button.place(x = 315, y = 60)
```

Listing 4.7: Syntax of defining icons in buttons (Tkinter)

The sample code for this is shown above. The image stored in the directory is first converted to a `PhotoImage`, then the `PhotoImage` is resized using the `.subsample()` method. Once the resizing is complete, the `PhotoImage` is attached to the button using the option `image = createphotoimage`. Figure 4.8 unveils the GUI for the System Preferences.

Next window on the list is the Delete User which displayed in the Figure 4.9. accommodates an OptionMenu, 2 labels and 4 buttons: Home, Back, Delete, Archive. The purpose of this window is to select a registered user from the database and delete it or archive the details in another file. The syntax of an OptionMenu is as follows:

```python
from tkinter import *
root = Tk()
dropmenu = OptionMenu(root, variable, default, *values)
dropmenu.place(x = 379,y = 228,width = 350)
```

Listing 4.8: OptionMenu Syntax

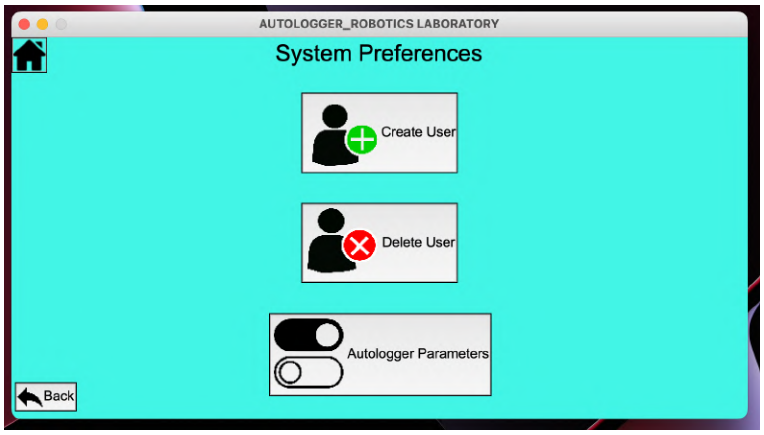


Figure 4.8.: System preferences page with three options (Window 5).

`OptionMenu` contains many options to be considered by the developer during the definition. It is contained in the parent window named `root`, the `variable` holds the currently selected option from the `OptionMenu`, `default` option is the initial option shown by the `OptionMenu` whereas the `*values` are the list of values that appear in the drop-down menu.

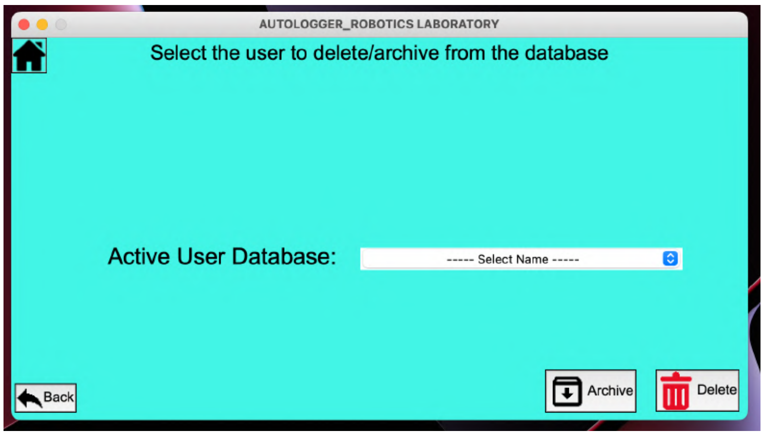
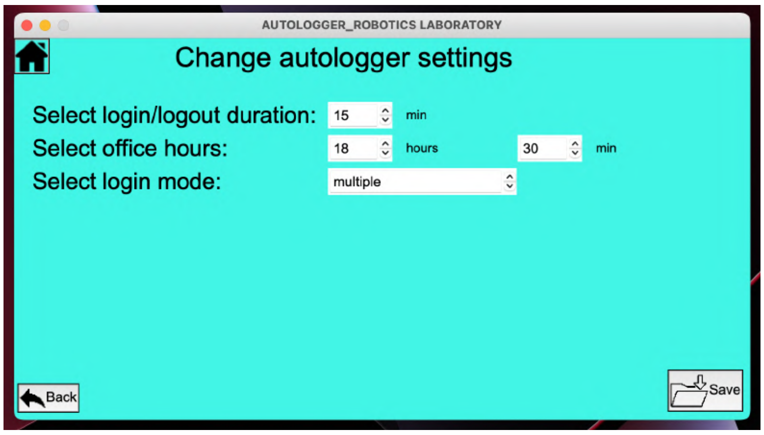

AUTOLOGGER_ROBOTICS LABORATORY
Select the user to delete/archive from the database
Active User Database:
----- Select Name -----
Back
Archive
Delete

Figure 4.9.: Delete/Archive users from the database (Window 9).

Autologger Settings (window 10 of Figure 4.1) allow the admin to change certain settings of the software such as: duration between one login and logout cycle, laboratory closing time and the login mode. These values can be changed with the help of spinboxes. There a total of 4 spinboxes, 3 buttons: Home, Back and Save and 7 labels in this GUI window (shown in Figure 4.10).

The syntax of a spinbox is as follows:

```python
from tkinter import *
root = Tk()
myspinbox = Spinbox(root, from_, to, textvariable, options,....)
```


AUTOLOGGER_ROBOTICS LABORATORY

Change autologger settings

Select login/logout duration: 15 min

Select office hours: 18 hours 30 min

Select login mode: multiple

Figure 4.10.: Attendance tracker parameters page (Window 10).

```
myspinbox.grid(row = 2, column = 3)
root.mainloop()
```

Listing 4.9: SpinBox syntax

Similar to other Tkinter widgets, `Spinbox` is also defined after defining the instance of a Tkinter frame: `root = Tk()`. It is contained in the `root` window, among other available `options` are: `from_`: which contains a minimum value, `to`: containing the maximum value and `textvariable`: which specifies a Tkinter `StringVar` object which holds the current value of the spinbox. The developer can also set other parameters such as fontstyle, fontsize, width of the spinbox etc. In this part of my code, the spinbox clings to the root window using `.grid()` method. `.grid()` basically divides the whole window into rows and columns and places the widget in the corresponding place as mentioned in the method parameters/arguments. `root.mainloop()`, a method belonging to the root window helps in executing the application which runs as a forever loop until the developer quits the window or triggers another event.

Last functionality of this software is the Create User window (window 6 of Figure 4.1) as shown in Figure 4.11. This assimilates 4 button widgets: Home, Back, Snap Picture and Save, 3 entry widgets and 5 label widgets. The placement of these labels and entry widgets have been established via `.grid()` method. The snap picture button is grayed out which means it is in disabled state. Once the above textfields are filled up, the snap picture button is enabled and upon clicking it opens window as shown in Figure 4.12. This window contains live camera feed and 3 buttons: Home, Back and Click Pic. This window (window 7 of Figure 4.1) is meant to click the picture of the new user attempting to register in the software database.

Once the button: Click Pic has been pressed, a picture of the user is snapped and a new window opens up. This window (window 8 of Figure 4.1) contains 2 labels: text label and an image label displaying the snapped picture of the new user, 4 buttons: Home, Back, Right and Wrong as shown in Figure 4.13.

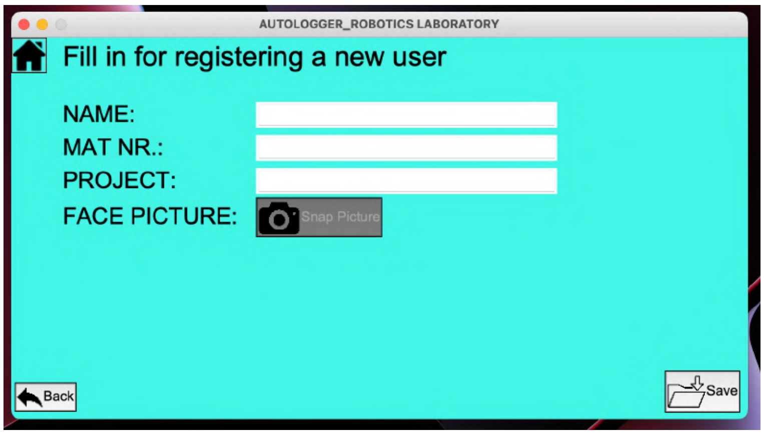

AUTOLOGGER_ROBOTICS LABORATORY
Fill in for registering a new user
NAME:
MAT NR.:
PROJECT:
FACE PICTURE:
Snap Picture
Back
Save

Figure 4.11.: Create user page with empty text fields (Window 6).

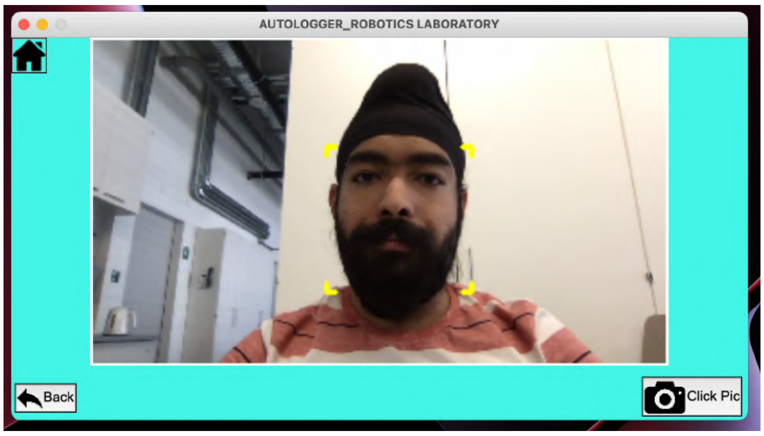


Figure 4.12.: Snap a picture along with a face tracker (Window 7).

4.2. Back-end Implementation with GUI Integration

This portion of the chapter addresses the implementation of the logic and its amalgamation with the user interface to accomplish a fully functional software.

The code starts off by importing certain python libraries such as: `tkinter`, `time`, `cv2`, `face-recognition`, `numpy`, `mediapipe`, `pickle`, `PIL`, `os` and `datetime`. Subsequently, the tkinter window has been defined with their corresponding configurations available as methods of the instance created such as window `title`, `geometry`, `background color`, explicit `resizing` and `protocol`. The corresponding code snippet is as follows:

```
def exit():
    #"dummy function"
    pass

root = Tk()
```

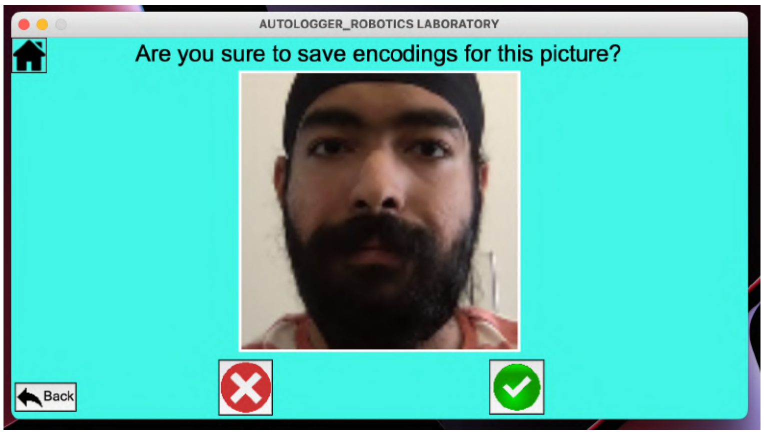

AUTOLOGGER_ROBOTICS LABORATORY
Are you sure to save encodings for this picture?
Back

Figure 4.13.: Confirmation to save the face encodings in the database: Yes/No (Window 8).

```
root.title('AUTOLOGGER_ROBOTICS LABORATORY')
root.geometry('800x415')
root.resizable(0,0)
root.configure(background = '#42f5e6')
root.protocol('WM_DELETE_WINDOW', exit)
```

Listing 4.10: Tkinter window configuration

The `.resizable()` method, doesn't allow explicit resizing of the GUI by the user as it has been declared to `False`. The `.configure()` method allows the developer to set the background color of his choice whereas, the `.protocol()` method prevents the termination of the software. This is a safety mechanism to ward off accidental termination of the software application by a user on the press of the exit button. Furthermore, the code imports and resizes the icon images to be used with the button widgets as discussed in the section 4.1: Front-end Implementation of chapter 4: Implementation. This is followed by importing the mediapipe frameworks which can run the model to detect hands, face and render the geometry with the following code:

```
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
mp_face_detection = mp.solutions.face_detection
```

Listing 4.11: MediaPipe Frameworks

Then the code checks for the existence of 3 files namely: autologger_tempfile.txt, autologger_logfile.txt and autologger_timedetails.txt. If these files are not present in the working directory, then they are automatically created by the program itself. This is the case when the code is executed for the very first time. Shown below is the code for the same:

```
timedetails_fileexists = os.path.exists('autologger_timedetails.txt')
tempfile_fileexists = os.path.exists('autologger_tempfile.txt')
logfile_fileexists = os.path.exists('autologger_logfile.txt')

if timedetails_fileexists == False:
    with open('autologger_timedetails.txt', 'w') as file:
            file.write(f'multiple,15,18,30')

if tempfile_fileexists == False:
    with open('autologger_tempfile.txt', 'w') as file:
        pass

if logfile_fileexists == False:
    with open('autologger_logfile.txt', 'w') as file:
        pass
```

Listing 4.12: Checking the existence of the files

The first 3 lines of code contain a boolean value: `True` or `False`. The method `os.path.exists()` checks the existence of the particular specified path, here in this case: the existence of the the file. If the value is `False`, then the file is created using the write mode: `'w'`. The file object `file` allows the programmer to access, use and manipulate the available files such as reading, writing and appending. The file autologger_timedetails.txt stores in information using the `.write()` method intended to be used as default values by the code. The details of these files are to be discussed later in this section.

About Autonomous Attendance Logger
This concept was not a part of the initial functional software requirements but was added later during the software designing phase to give a brief overview of the features of the software as well as the details of the author displayed through text as discussed in section 4.1: Front-end Implementation, integrated in window 1 of the Figure 4.1 which is referred to the function name: `about_software()` in the code. This function is called by `aboutus_button` widget existing in the function: `window2()`.

User Login/Logout
This is the core functionality of this attendance tracker software system which manages the login/logout of a user by creating a digital attendance sheet. It refers to Window 3 of Figure 4.1, invoked by `login_button` incorporated in the function `window2()` of the code. The button executes the function `window3()` which sets up and displays the corresponding GUI. The code snippet of `window3()` is shown as follows:

```
def window3():
    with open('autologger_timedetails.txt', 'r') as timefile:
        loclist = timefile.read()
        ....
        ....
            video.config(image = imgtk)
            root.update()
```

Listing 4.13: User login function

The system of logging in/logging out a user utilises 3 AI models working together: face detection, face recognition and a 2 stage pipeline (considered as a single unit) running palm detection and hand landmark detection. The face detection and face recognition is to verify the authenticity of the user attempting to login/logout of the laboratory whereas, the hand gesture recognition is used to identify two unique gestures: thumb-up for login and thumbs-down for logout. Once the user and the gesture is identified, the system writes the user details along with the time in the file.

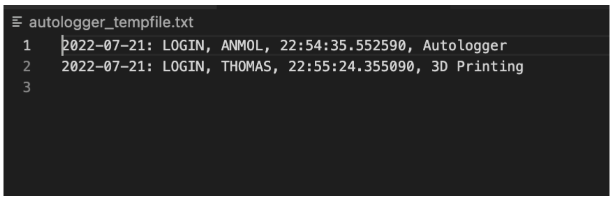
The system implements the creation of attendance sheet with the use of two files: autologger_tempfile.txt and autologger_logfile.txt. The tempfile is a temporary file storing the entries of signing in/out for just one day whereas the logfile is the permanent attendance sheet recording all the data. The reason to deploy two files instead of having just one is to reduce the computation cycles performed by the processor to read a file, which gets bigger in size due to storage of more entries as each day passes by. In other words, the system just writes the data in the logfile whereas reads, writes and makes changes to the data stored in the tempfile. The format in which the entries are stored in the above mentioned files are:

DATE: LOGIN STATUS, NAME, TIME STAMP, PROJECT

Other files being used in this function are: autologger_userdetails.txt, autologger _faceencodings.txt and autologger_timedetails.txt. The first two files are the database for the system to verify the authenticity whereas the time details file is used to extract and use the information such as: login mode type, minimum duration between a login and logout (1 cycle) and the lab closing time. The login mode consists of two options: Single login mode means that a user can only login and logout of the lab once in a day whereas the multiple login mode allows a user to login and logout multiple times in a single day.

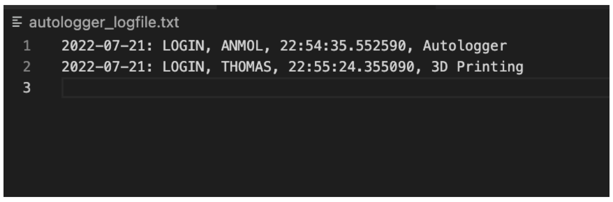
The function in the above shown code excerpt starts off by extracting the information from the autologger_timedetails.txt file as mentioned above and storing them in unique variables. The lab closing time is represented by creating a `time()` object using the `time` module from the library `datetime`. Then the system signs out the active users for the previous day who forgot to log out and also clears the tempfile for old dates. This is accomplished by reading the tempfile entries for the date and status "LOGIN". For any entry, if the date is not same as the current date and the status is "LOGIN", the program writes those entries in the logfile with "LOGOUT" status and time stamp same as the lab closing time. Moreover, it also clears the tempfile simultaenously for the old dated entries. This is showcased in Figure 4.14, 4.15, 4.16 and 4.17. This is termed as automatic logout.

The above piece of code makes the system fail-proof as in case of a sudden unprecedented power failure, the code would always check for any old date entries with active logins on booting up and if it finds one, it would log the users out, clear the tempfile and then proceed forward for the next day. Subsequently, it loads the face encodings database from the file: autologger_faceencodings.txt and creates the User login/logout window containing the defined widgets. With this, the camera turns on, face detection, face recognition and hand gesture



```
autologger_tempfile.txt
2022-07-21: LOGIN, ANMOL, 22:54:35.552590, Autologger
2022-07-21: LOGIN, THOMAS, 22:55:24.355090, 3D Printing
```

Figure 4.14.: Login tempfile.



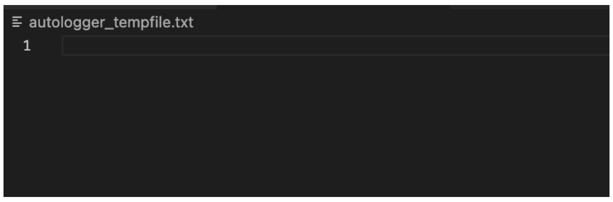
```
autologger_logfile.txt
2022-07-21: LOGIN, ANMOL, 22:54:35.552590, Autologger
2022-07-21: LOGIN, THOMAS, 22:55:24.355090, 3D Printing
```

Figure 4.15.: Login logfile.

recognition models up and running, thus making the system ready for attendance tracking.

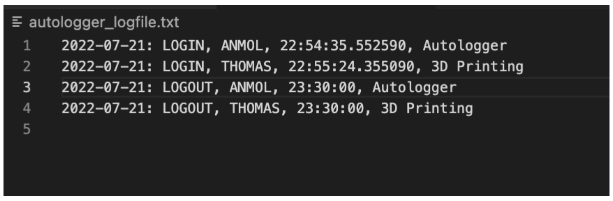
In the `while True` loop, the input frames from the camera are read (please refer to Figure 3.7, 3.8, and 3.9 for enhanced comprehension), image pre-processing steps are applied: resizing and conversion of the color channel from BGR to RGB which is then input to the face detection model to output the coordinates of the faces detected in the frame. As the face detection model is capable of multi face detection irrespective of the position or the depth of the face in the frame, there is no control over detecting a desired face as the model has been trained just to detect faces and not to detect face of a significant focus. So, the code is embedded with an additional custom algorithm which controls the input to the face recognition model to evade the possibility of a false face recognition within the frame. Moreover, the amount of future computations is also reduced. The code for the custom algorithm is shown below:

```python
def facerecognitionfun(results,wid,hei):
    previousarea = 0
    y1, x2, y2, x1 = None, None, None, None

    if results.detections:
        for idx, detection in enumerate(results.detections):
```



```
autologger_tempfile.txt
```

Figure 4.16.: Automatic Logout tempfile.



```
autologger_logfile.txt
2022-07-21: LOGIN, ANMOL, 22:54:35.552590, Autologger
2022-07-21: LOGIN, THOMAS, 22:55:24.355090, 3D Printing
2022-07-21: LOGOUT, ANMOL, 23:30:00, Autologger
2022-07-21: LOGOUT, THOMAS, 23:30:00, 3D Printing
```

Figure 4.17.: Automatic Logout logfile.

```
            xmin,ymin,width,height = detection.location_data.relative_bounding_box.xmin,
    detection.location_data.relative_bounding_box.ymin,detection.location_data.
    relative_bounding_box.width,detection.location_data.relative_bounding_box.height

            facearea = abs(width*height)

            if facearea>previousarea:
                previousarea = facearea
                y1, x2, y2, x1 = int(ymin * hei/4), (int(xmin * wid/4) + int(width * wid/4)),
    (int(ymin * hei/4) + int(height * hei/4)), int(xmin * wid/4)
    return y1, x2, y2, x1
```

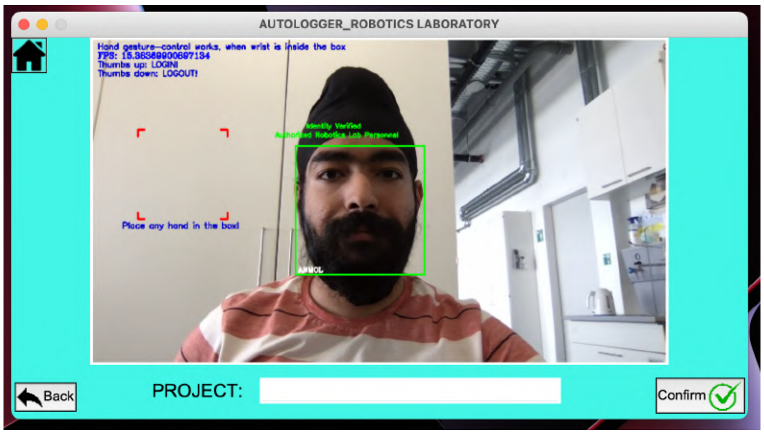
Listing 4.14: Face-recognition function

The above `facerecognitionfun` function inputs the location coordinates of all the faces along with the width and height of the frame and outputs the location of the face closest to the camera. This is achieved by calculating the area bounded by each face. The one with highest area is returned and rest are ignored. Thus the original function `window3()` calls the above function and proceeds forward with just one face detection. Besides this, the function also renders some information for the user on the screen as well as a box to be used for hand detection and gesture recognition. Once the face is detected and the location coordinates are returned, the code runs the face-recognition model on that face only to extract the face encodings. These encodings are then matched against each encoding saved in the database which returns the index of element of the database which is true for this picture: `matchindex`. Once the `matchindex` is obtained, the corresponding name of the user stored with the encodings file is extracted. This part is known as user authorisation.

If the user is identified, the bounding box turns green with the username being displayed (as shown in Figure 4.18) else the bounding box turns red stating that the user hasn't been recognized and the name is displayed as: Unknown (as shown in Figure 4.19).

Once the user authorisation is successful, the image frame is passed to another function: `handgesturecontrol()`. The snippet of the following function is shown below:

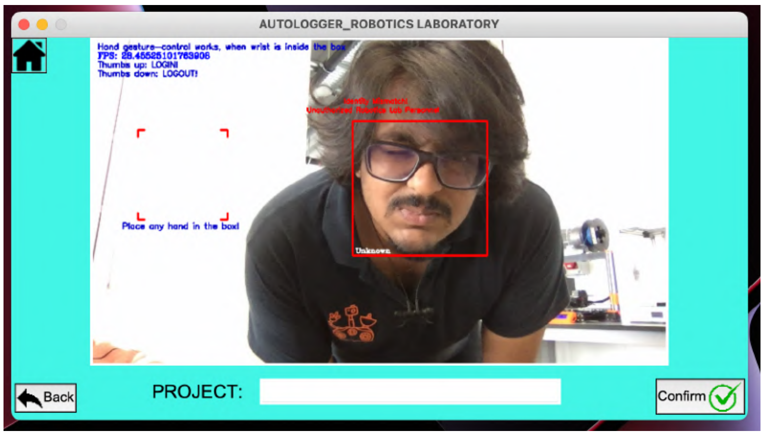
```
def handgesturecontrol(imageframe):
        imgframe = cv2.cvtColor(imageframe, cv2.COLOR_BGR2RGB)
        results = hands.process(imgframe)

```



```
Hand gesture—control works, when wrist is inside the box
FPS: 15.383699006897134
Thumbs up: LOGIN!
Thumbs down: LOGOUT!
Identity Verified
Authorized Robotics Lab Personnel
ANMOL
Place any hand in the box!
```

Figure 4.18.: Authorised user of the robotics laboratory(with name).

```
        if results.multi_hand_landmarks:

            hand_landmarks = results.multi_hand_landmarks[0]

            mp_drawing.draw_landmarks(imageframe,hand_landmarks, mp_hands.HAND_CONNECTIONS,
    mp_drawing.DrawingSpec(color = (255,0,0), thickness =  2, circle_radius = 4),
            mp_drawing.DrawingSpec(color = (200,0,0), thickness =  2, circle_radius = 2))

            x0 = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].x * width
            y0 = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].y * height
            y4 = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].y * height

            y_arr = []
            for item in hand_landmarks.landmark:
                y_arr.append(item.y * height)
```



```
Hand gesture—control works, when wrist is inside the box
FPS: 28.45525101763908
Thumbs up: LOGIN!
Thumbs down: LOGOUT!
Identity Mismatch!
Unauthorized Robotics Lab Personnel
Unknown
Place any hand in the box!
```

Figure 4.19.: Unauthorised user of the robotics laboratory(unknown). Picture Credits: Mr. Mamen Thomas Chembakasseril.

```
            y_arr.pop(4)
            y_numbers = np.array(y_arr)

            return y_numbers ,x0, y0, y4
        return None, None, None, None
```

Listing 4.15: Hand-gesture function

The above function inputs the camera frame and runs the palm detection and hand landmark detection model to detect if any hands are available. If hands are detected, it draws the landmarks for only one hand. The reason behind is that only one hand is needed at a time to fulfill the functionality of logging in/out by showing a hand gesture. This further reduces extra computations as well. This function finally returns the y-coordinates of all the detected 21 hand landmarks which would be used later for gesture recognition. An important point to be noted is that the hand detection and recognition works only after a user has been identified as evident from the Figure 4.20 and Figure 4.21.

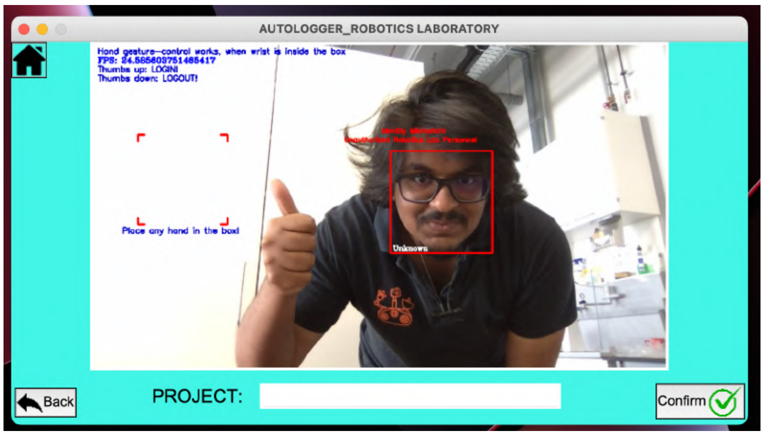

AUTOLOGGER_ROBOTICS LABORATORY
Hand gesture-control works, when wrist is inside the box
FPS: 24.585603751485417
Thumbs up: LOGIN!
Thumbs down: LOGOUT!
Place any hand in the box!
Identity Mismatch!
Unauthorized Robotics Lab Personnel
Unknown
Back
PROJECT:
Confirm

Figure 4.20.: Gesture control doesn't not work for unauthorised users. Picture Credits: Mr. Mamen Thomas Chembakasseril.

Once the hand is detected, the gesture recognition comes into role. For the gesture recognition to work, the hand must be placed within the box rendered on the camera screen. This is executed by comparing the y-coordinate value of the wrist hand landmark(shown in Figure 2.14) with the x and y coordinate values of the box. If the coordinate lies within the x and the y range, the hand is detected inside the box else the hand is detected outside the box. Once the hand is placed within the box, the color of the box changes from red to green indicating that the hand is placed in the desired area waiting to give out a gesture. As mentioned previously, that thumb-up gesture indicates login whereas the thumbs-down gesture indicates logout. The logic created to realize this is very simple: just compare the y-values of all the hand landmarks with the y-coordinate of the thumb tip. If the y-coordinate value of the thumb tip is is smaller than all the others: indicates a login else a logout.

Login Gesture Detected: Once a login gesture is perceived and rendered on the screen as "login gesture detected", the project entry widget is enabled for the user to type in the name of the current project he/she is working upon and a couple of flag variables are set to `True` or `False` according to the validity of conditional statements being checked. At first, the tempfile is read to check if the particular person has logged in before for that day. This logic is to prevent subsequent logins before logging out. If not the case, then the time of logging in is checked if it is within the lab opening time. If the time is within the working hours, the log in is successful (as shown in Figure 4.21) else a text is rendered on the screen stating that the user can't login as the lab is closed (shown in Figure 4.22).

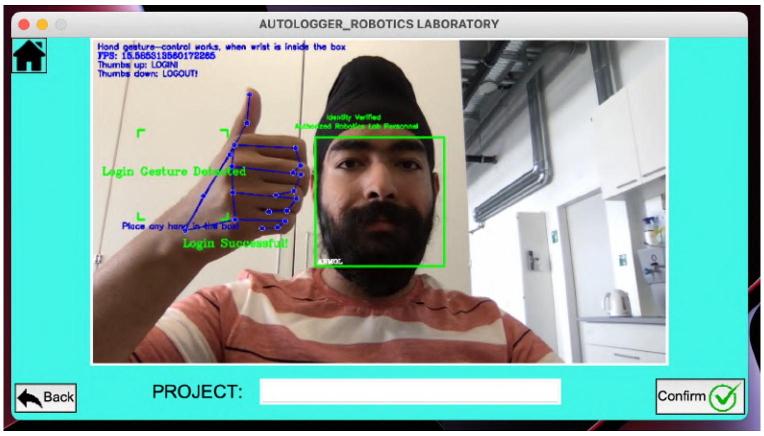

AUTOLOGGER_ROBOTICS LABORATORY
Hand gesture-control works, when wrist is inside the box
FPS: 15.585313580172265
Thumbs up: LOGIN!
Thumbs down: LOGOUT!
Identity Verified
Authorized Robotics Lab Personnel
Login Gesture Detected
Place any hand in the box!
Login Successful!
ANMOL
Back
PROJECT:
Confirm

Figure 4.21.: Authorised user successful login.

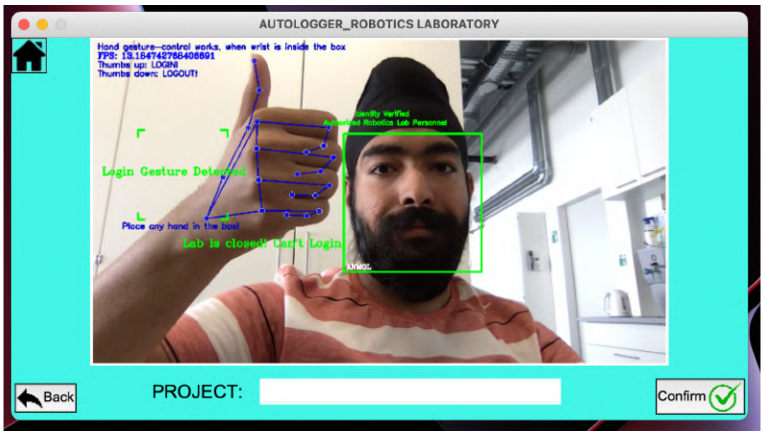

AUTOLOGGER_ROBOTICS LABORATORY
Hand gesture-control works, when wrist is inside the box
FPS: 13.184742768408891
Thumbs up: LOGIN!
Thumbs down: LOGOUT!
Identity Verified
Authorized Robotics Lab Personnel
Login Gesture Detected
Place any hand in the box!
Lab is closed! Can't Login
ANMOL
Back
PROJECT:
Confirm

Figure 4.22.: User cannot log in once the lab is closed.

Logout Gesture Detected: Once a logout gesture is perceived and rendered on the screen as "logout gesture detected", the story is a bit different. Firstly, the project entry widget is not enabled for the user as the system extracts the name of the project from the login entry itself so there is no need to type in the project name again and again. Secondly, a couple of flag variables are set to `True` or `False` according to the validity of conditional statements being checked. After detecting the logout gesture from the screen, the autologger_tempfile.txt is read to check if the person has logged in before or not. If the person has not logged in before, a message is rendered on the camera screen using OpenCV `.putText()` method stating to login into the system first (shown in Figure 4.23).

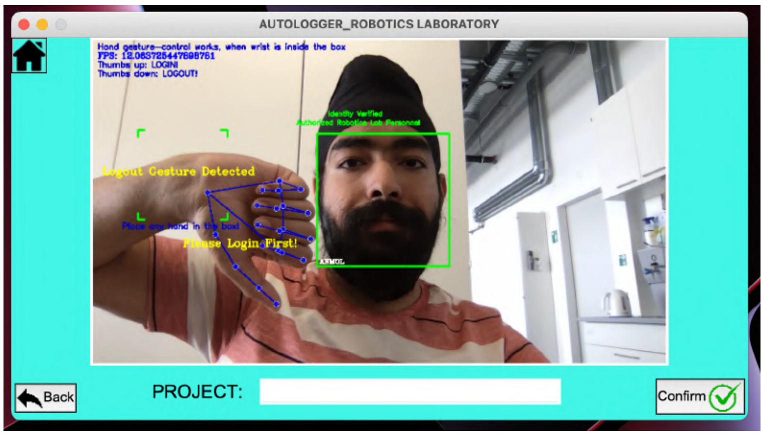
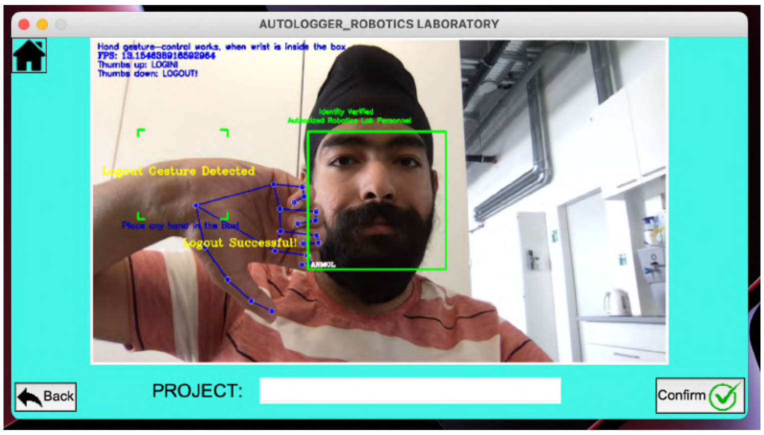

AUTOLOGGER_ROBOTICS LABORATORY
Hand gesture—control works, when wrist is inside the box
FPS: 12.063725447698761
Thumbs up: LOGIN!
Thumbs down: LOGOUT!
Identity Verified
Authorized Robotics Lab Personnel
Logout Gesture Detected
Place any hand in the box!
Please Login First!
ANMOL
Back
PROJECT:
Confirm

Figure 4.23.: Attendance tracker doesn't let the user log out before logging in for the day.

If the person has logged in before, then the current logout time is checked if it within the lab closing time. If the current time is greater than the lab closing time then manual logout is not possible. In such a case, the system will log out the active user automatically the next day. But if the current time is less than the lab closing time, another condition for minimum duration between login and logout is checked for that particular person. If the minimum time duration has been elapsed, a message is rendered for both single login and multiple login mode stating that the logout has been successful (as shown in Figure 4.24) otherwise a message is rendered on the screen affirming that the user can't logout at the moment (as shown in Figure 4.25).

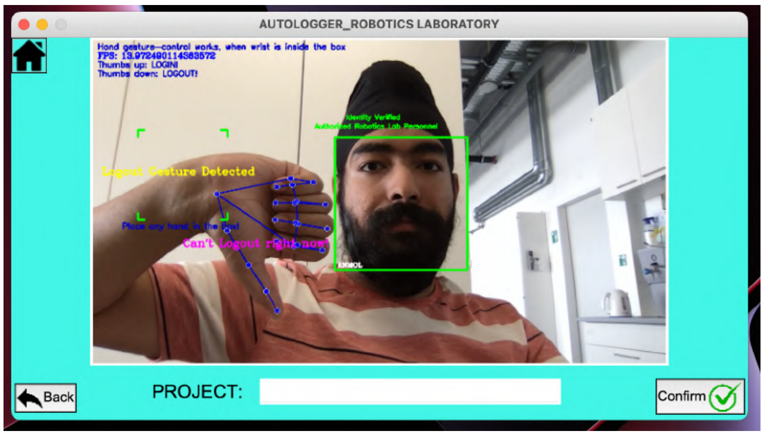
The story of user login/logout doesn't end here as this function just records the outputs of the conditional statements in various flag variables but does not write the entries of the attendance to the tempfile and logfile. This is done when the confirm button is pressed. This button is named as `confirmbutton` lying in `window3()` of the code which calls a function `confirm_login()`. The purpose of this function is to write the entries in the tempfile and logfile according to the recorded values in various flags. For the login and logout entries, only tempfile is manipulated by reading, writing and appending the data whereas the system only writes login and logout description in the logfile. This is elaborated in the following text.

First of all, after pressing the confirm button the system extracts the name of the project typed in by the user in the entry widget while logging in. If the field is left empty, then the system automatically extracts the default project name from the autologger_userdetails.txt file and



```
Hand gesture-control works, when wrist is inside the box
FPS: 13.154638916592964
Thumbs up: LOGIN!
Thumbs down: LOGOUT!
Identity Verified
Authorized Robotics Lab Personnel
Logout Gesture Detected
Place any hand in the Box!
Logout Successful!
ANMOL
```

Figure 4.24.: Authorised user successful logout.



```
Hand gesture-control works, when wrist is inside the box
FPS: 13.972490114383572
Thumbs up: LOGIN!
Thumbs down: LOGOUT!
Identity Verified
Authorized Robotics Lab Personnel
Logout Gesture Detected
Place any hand in the Box!
Can't Logout right now!
ANMOL
```

Figure 4.25.: User cannot logout before the minimum time duration between login/logout has elapsed.

then added in the login entry. As far as logout is concerned, the same project name which exists in the login entry of that user is extracted and inserted in the logout attendance entry. The concept of having an additional entry widget for project name apart from the camera in Window 3 is that the user can work on various projects in the laboratory on different days which is bound to be different from the project with which he/she registered at the beginning. Moreover, the tempfile is written in a different fashion in case of a single and a multiple login/logout mode.

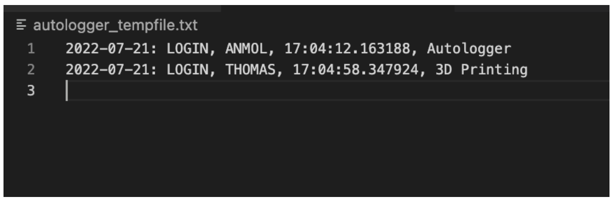
If the mode selected is: single login/logout mode then during login, the system writes the same description in both tempfile and logfile which is of format:

DATE: LOGIN STATUS, NAME, TIME STAMP, PROJECT

Whereas during logging out of the system in a single login/logout mode, the logfile is written in the same format as shown above but the tempfile is written in the following format:

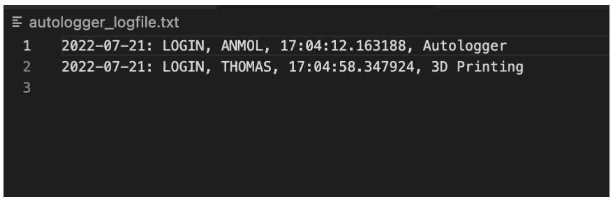
DATE: NAME

The reason behind this logic is that if the person tries to login again the same day, then the code checks the tempfile if the person had already logged out for that particular day. If that's the case as shown by the above format, the system won't allow the person to login again. This is shown by the Figure 4.26, 4.27, 4.28 and 4.29. Moreover, the system also ignores subsequent login gestures if the user has already logged in the laboratory(as shown in Figure 4.30). Subsequent logout gestures are even ignored once the person has logged out of the system(as shown in Figure 4.23). This also shows that the user can't logout before logging in.



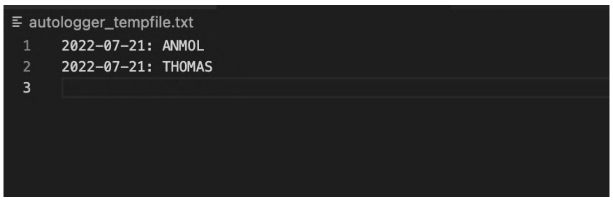
```
autologger_tempfile.txt
2022-07-21: LOGIN, ANMOL, 17:04:12.163188, Autologger
2022-07-21: LOGIN, THOMAS, 17:04:58.347924, 3D Printing
```

Figure 4.26.: Single Login mode tempfile.



```
autologger_logfile.txt
2022-07-21: LOGIN, ANMOL, 17:04:12.163188, Autologger
2022-07-21: LOGIN, THOMAS, 17:04:58.347924, 3D Printing
```

Figure 4.27.: Single Login mode logfile.

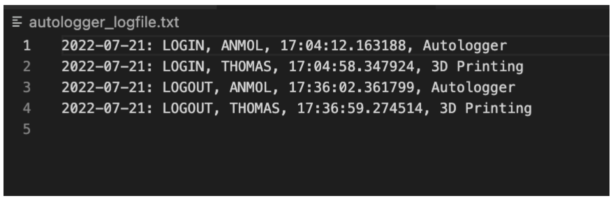


```
autologger_tempfile.txt
2022-07-21: ANMOL
2022-07-21: THOMAS
```

Figure 4.28.: Single Logout mode tempfile.

In case of a multiple login/logout mode, the tempfile and logfile are written in the following manner for login:

DATE: LOGIN STATUS, NAME, TIME STAMP, PROJECT



```
autologger_logfile.txt
2022-07-21: LOGIN, ANMOL, 17:04:12.163188, Autologger
2022-07-21: LOGIN, THOMAS, 17:04:58.347924, 3D Printing
2022-07-21: LOGOUT, ANMOL, 17:36:02.361799, Autologger
2022-07-21: LOGOUT, THOMAS, 17:36:59.274514, 3D Printing
```

Figure 4.29.: Single Logout mode logfile.

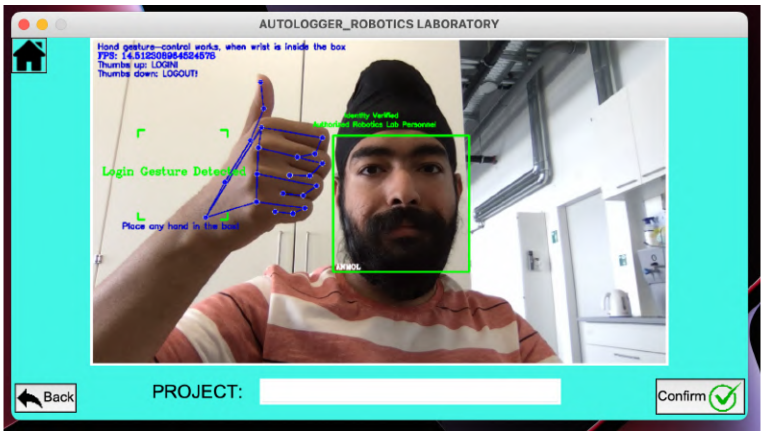
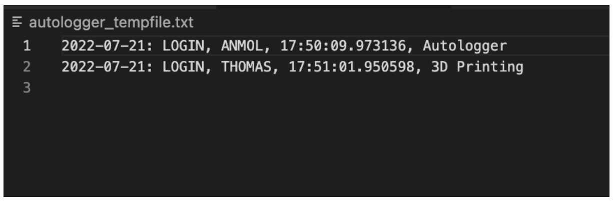

AUTOLOGGER_ROBOTICS LABORATORY
Hand gesture-control works, when wrist is inside the box
FPS: 14.512308964524576
Thumbs up: LOGIN!
Thumbs down: LOGOUT!
Identity Verified
Authorised Robotics Lab Personnel
Login Gesture Detected
Place any hand in the box!
ANMOL
Back
PROJECT:
Confirm

Figure 4.30.: Ignoring subsequent login gestures.

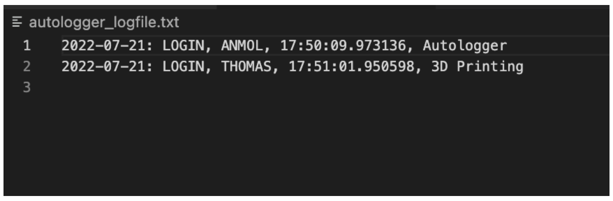
But in case of logout in a multiple login/logout mode, the entry in the logfile is written as shown above but in case of tempfile, the complete login entry is erased. Erasing the entry would allow the user to login into the system again as there is no description of his/her attendance existing in the tempfile. This is demonstrated using the Figure 4.31, 4.32, 4.33 and 4.34.



```
autologger_tempfile.txt
2022-07-21: LOGIN, ANMOL, 17:50:09.973136, Autologger
2022-07-21: LOGIN, THOMAS, 17:51:01.950598, 3D Printing
```

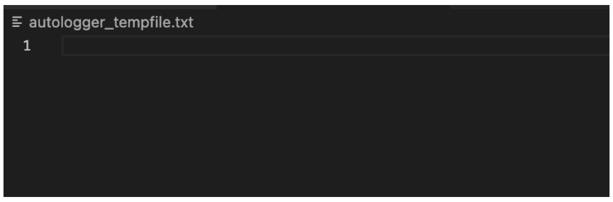
Figure 4.31.: Multiple Login mode tempfile.

The last case of file handling to be discussed is that what happens if the admin changes the autologger parameters in the middle of the day? How does the system respond in this situation? Well an algorithm has been put to action for this case as well. Figure 4.35 and Figure 4.36 depict login status for 2 users in the tempfile and logfile. Figure 4.37 and Figure 4.38 showcase one user logged out of the system on a single login/logout mode with its corresponding logout entry added to the logfile and the tempfile appended according to



```
autologger_logfile.txt
2022-07-21: LOGIN, ANMOL, 17:50:09.973136, Autologger
2022-07-21: LOGIN, THOMAS, 17:51:01.950598, 3D Printing
```

Figure 4.32.: Multiple Login mode logfile.



```
autologger_tempfile.txt
```

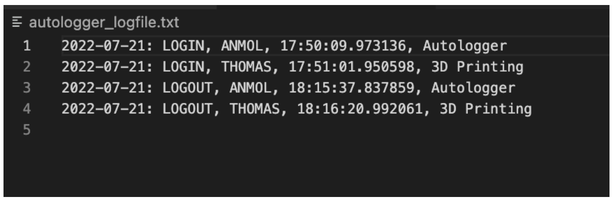
Figure 4.33.: Multiple Logout mode tempfile.

logic explained above in the section. Now the admin changes the autologger parameter settings from single login/logout mode to multiple login/logout mode. Once the changes have been saved, the Figure 4.39 shows an appended tempfile in which the logout entry has been erased. This is accomplished using the following code snippet existing in the function `save_autologger_parameters()` of the code.

```python
with open('autologger_tempfile.txt', 'r') as f:
       lineslist = f.readlines()

   with open('autologger_tempfile.txt','w') as ff:
       for line in lineslist:
           if f'LOGIN' in line.strip('\n'):
               ff.write(line)
           else:
               ff.write('')
```

Listing 4.16: Appending tempfile after saving changes from parameters window

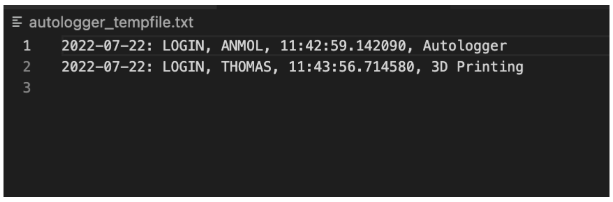
Figure 4.40 and Figure 4.41 shows that the same user who logged out of the system that day can login again. When the login mode is changed from multiple mode to single mode, no algorithm is needed as the login and logout entries are erased off from the tempfile during



```
autologger_logfile.txt
2022-07-21: LOGIN, ANMOL, 17:50:09.973136, Autologger
2022-07-21: LOGIN, THOMAS, 17:51:01.950598, 3D Printing
2022-07-21: LOGOUT, ANMOL, 18:15:37.837859, Autologger
2022-07-21: LOGOUT, THOMAS, 18:16:20.992061, 3D Printing
```

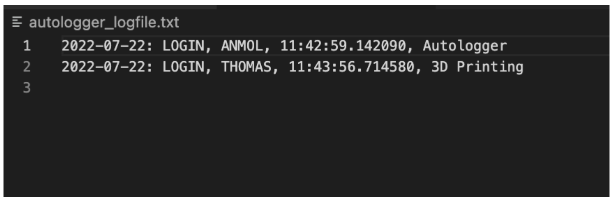
Figure 4.34.: Multiple Logout mode logfile.

user logout in multiple login/logout mode. Last but not least, once the confirm button is pressed and the entries are made to the logfile and tempfile, the widgets of Window 3 are destroyed and the GUI return to the previous window: Window "MAIN".



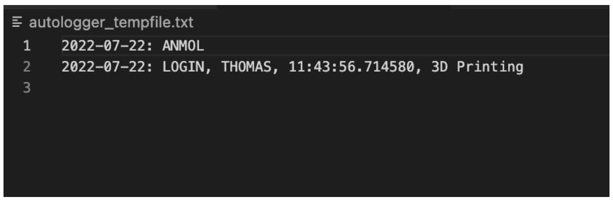
```
autologger_tempfile.txt
2022-07-22: LOGIN, ANMOL, 11:42:59.142090, Autologger
2022-07-22: LOGIN, THOMAS, 11:43:56.714580, 3D Printing
```

Figure 4.35.: Login tempfile.



```
autologger_logfile.txt
2022-07-22: LOGIN, ANMOL, 11:42:59.142090, Autologger
2022-07-22: LOGIN, THOMAS, 11:43:56.714580, 3D Printing
```

Figure 4.36.: Login Logfile.



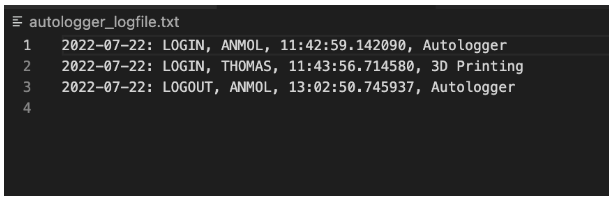
```
autologger_tempfile.txt
2022-07-22: ANMOL
2022-07-22: LOGIN, THOMAS, 11:43:56.714580, 3D Printing
```

Figure 4.37.: Tempfile showing a user logged out on a single login mode.

Active Users

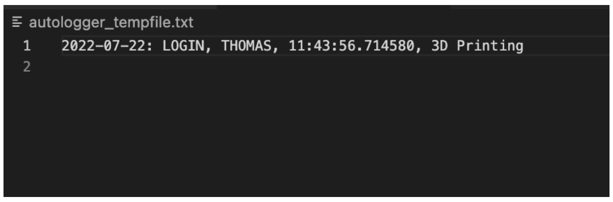
Active Users (or window 2 in Figure 4.1) displays the currently logged in students in the lab. This concept was not a part of the initial functional requirements of the system but was later added by the developer as an additional feature which would help the admin to see the list of working students and their corresponding projects by a simple click on the GUI rather opening up the logfile. The logic behind this is very simple as it reads the tempfile and displays only those entries in the listbox which have the current date and "LOGIN" status in their corresponding attendance description. The code snippet and its corresponding GUI (Figure 4.42) for active users is shown below:

```python
with open('autologger_tempfile.txt','r') as f:
    datum = str(datetime.now().date())
    stat = "LOGIN"
    for line in f:
        if datum in line and stat in line:
```



```
autologger_logfile.txt
2022-07-22: LOGIN, ANMOL, 11:42:59.142090, Autologger
2022-07-22: LOGIN, THOMAS, 11:43:56.714580, 3D Printing
2022-07-22: LOGOUT, ANMOL, 13:02:50.745937, Autologger
```

Figure 4.38.: Logfile showing a user logged out on a single login mode.



```
autologger_tempfile.txt
2022-07-22: LOGIN, THOMAS, 11:43:56.714580, 3D Printing
```

Figure 4.39.: System preferences changed and saved to multiple login mode, tempfile gets updated.

```
            active_users_listbox.insert(END, str(line))
```

Listing 4.17: Active Users

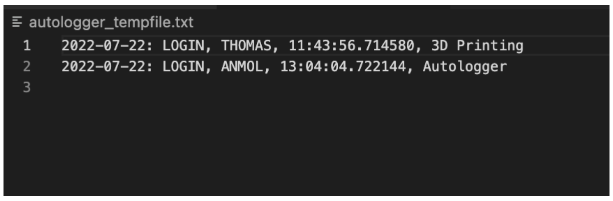
The above snippet is a part of the function `active_users_window()` where the other GUI components for above window are defined and this function is called by `active_login_button` widget existing in the function: `window2()`.

System Preferences

System Preferences section is under the control of the admin, thus it's password protected. The password prompt window opens up as soon as system preferences button is pressed which exists as `system_preferences_button` in the function `window2()` of the code (also shown as window "MAIN" in Figure 4.1). The password is accepted by the GUI from the admin through an entry widget which is defined in the function called `admin_password_window()`. Once the Admin Login button is pressed, also defined in the above function as `admin_login_button`, it calls a function `admin_verification()` to verify the password. If the password is correct it deletes the entered password in the entry widget and then calls the function `window 4()` or basically window 5 of Figure 4.1. The function for verifying the password is shown below:

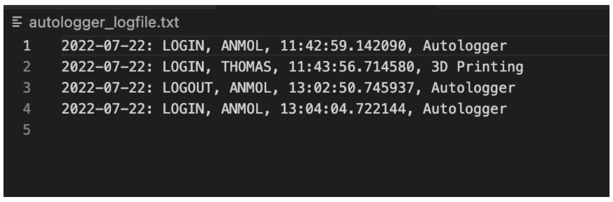
```
def admin_verification():
    password = '05eg025'
    user_pass = str(password_textfield.get())
    if user_pass == password:
        password_textfield.delete(0, END)
        window4()
    else:
        password_textfield.delete(0, END)
```

Listing 4.18: Password Verification



```
autologger_tempfile.txt
2022-07-22: LOGIN, THOMAS, 11:43:56.714580, 3D Printing
2022-07-22: LOGIN, ANMOL, 13:04:04.722144, Autologger
```

Figure 4.40.: Tempfile showing the same user can login again the same day.



```
autologger_logfile.txt
2022-07-22: LOGIN, ANMOL, 11:42:59.142090, Autologger
2022-07-22: LOGIN, THOMAS, 11:43:56.714580, 3D Printing
2022-07-22: LOGOUT, ANMOL, 13:02:50.745937, Autologger
2022-07-22: LOGIN, ANMOL, 13:04:04.722144, Autologger
```

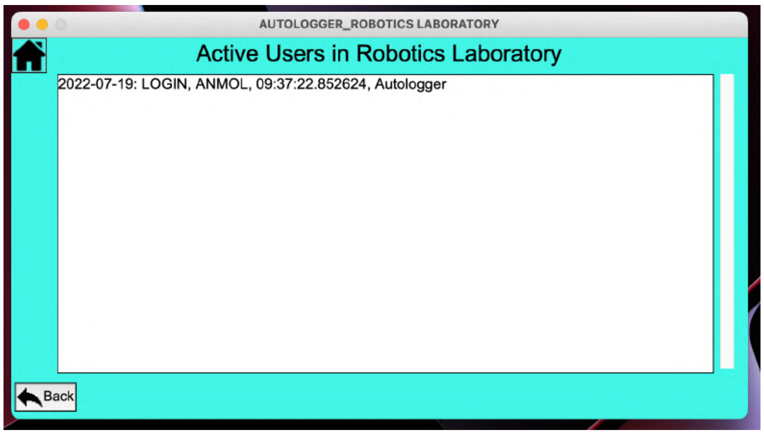
Figure 4.41.: Logfile showing the complete history during the change of system preferences from single to multiple login mode or vice versa.

Each character of the password is entered as an "*" to mask the characters and hide the password, making the system secure. The password masking is shown by the following Figure 4.43.

Create User

As the caption suggests, it is required to register a new user in the database through the GUI and is labelled as window 6 in Figure 4.1. The create user function exists in code as: `window5(encodeface=None)` which is called by `create_user_button` in `window4()`.

```python
my_str1 = StringVar()
my_str2 = StringVar()
my_str3 = StringVar()

nametextfield = Entry(root, width=35, borderwidth = 2, bg="white", fg = "black", textvariable
    = my_str1)
nametextfield.grid(row = 2, column = 2, sticky = W)

matriculationtextfield = Entry(root, width=35, borderwidth = 2, bg="white", fg = "black",
    textvariable = my_str2)
matriculationtextfield.grid(row = 3, column = 2, sticky = W)

projecttextfield = Entry(root, width=35, borderwidth = 2, bg="white", fg = "black",
    textvariable = my_str3)
projecttextfield.grid(row = 4, column = 2, sticky = W)

def enablebutton(*kwargs):
    global username
    global userid
    global userdefaultproject

    if my_str1.get() != '' and my_str2.get() != ''  and my_str3.get() != '':
```


AUTOLOGGER_ROBOTICS LABORATORY
Active Users in Robotics Laboratory
Back

2022-07-19: LOGIN, ANMOL, 09:37:22.852624, Autologger

Figure 4.42.: List of active logged in users in the robotics laboratory.

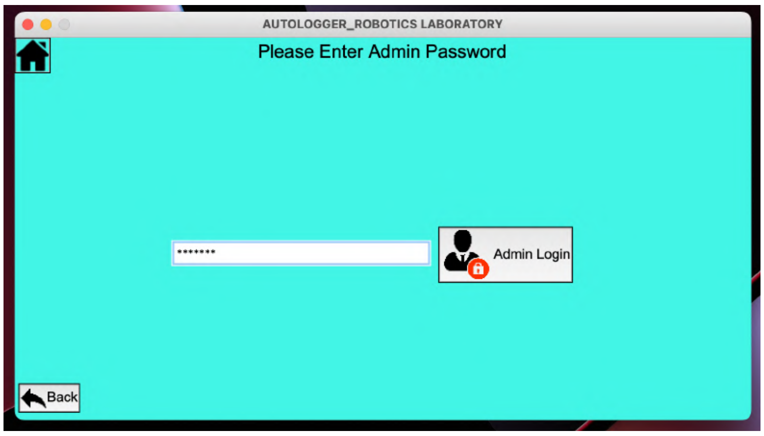


Figure 4.43.: Admin controlled window when password is entered.

```
        username = my_str1.get()
        userid = my_str2.get()
        userdefaultproject = my_str3.get()
        facepicture.config(state = 'normal')
    else:
        facepicture.config(state = 'disabled')

my_str3.trace('w', enablebutton)
```

Listing 4.19: Create user code

The above code snippet belongs to the function `window5(encodeface=None)` where the user is prompted to fill out the registration form first, then take a picture of his/her face in order to register him/her in the database. The Snap Picture button as shown in Figure 4.11 is enabled only when the above information filled out in order: name, matriculation number and project respectively. This has been done using a class `StringVar()` belonging to tkinter library. The `StringVar()` object allows a coder to manage, use and manipulate the value of an entry widget. Therefore, all the above entry widgets have a `StringVar()` object defined in their widget definition which is able to access the value stored inside those entry widgets. The `.trace()` method of the `StringVar()` object notifies when its value changes and `'w'` mode automatically invokes `enablebutton` function when the value of the associated `StringVar()` object changed. This feature turns out to be useful when other widgets needs to be updated using the current value of the `StringVar()` object. The same principle applies in this case too. As soon as the last entry widget is filled out, the `enablebutton` function is invoked which checks if all the entries are filled out and if that's the case then the Snap Picture button is enabled and the entry widget values are stored in corresponding variables. The GUI for a completely filled up registration form is shown in Figure 4.44.

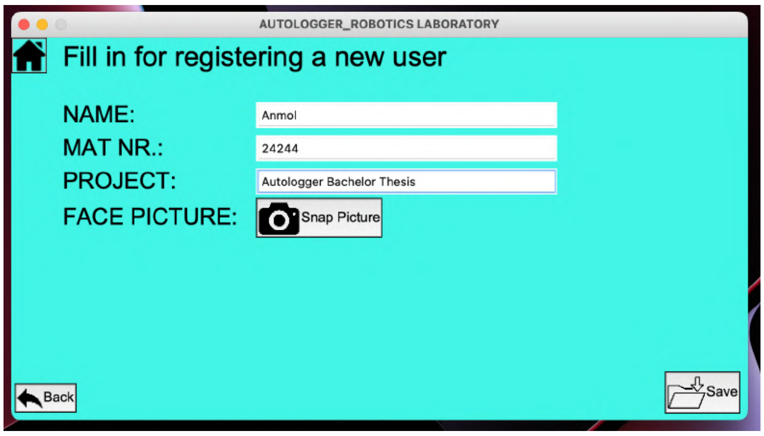

AUTOLOGGER_ROBOTICS LABORATORY
Fill in for registering a new user
Snap Picture
Back
Save
NAME: Anmol
MAT NR.: 24244
PROJECT: Autologger Bachelor Thesis
FACE PICTURE:

Figure 4.44.: Snap picture button enabled once the user fills in all the details step-by-step.

The Snap Picture button named as `facepicture` in the code opens up Window 7 as exhibited in Figure 4.1. The purpose of this window is to take the picture of the user attempting to register in the database. The code snippet for the following window is shown as follows:

```
with mp_face_detection.FaceDetection(min_detection_confidence=0.5) as face_detection:
        while cap2.isOpened():
            ret,frame2 = cap2.read()
            ....
            ....
                videolabel2.config(image = imgtk2)
                root.update()
```

Listing 4.20: Snap picture code

The above code snippet is a part of the function `window8()` where a camera object is created first, which reads the frames received from the camera. To make the system even more user friendly, a face tracker has been introduced which detects and tracks a face and creates a bounding box around it. Having a bounding box already rendered, the person can stand anywhere in the frame as long as the user's face is detected to click a picture. To accomplish this task, a face detection model is used and the 4 coordinate points received per face detection as the output from the model are fed as an input to `facerecognitionfun()` function which outputs only the coordinates of the face closest to the camera. This leads to only one detection at a time as the output is developer controlled and evades a possibility of a non significant face detection.

The Click Pic button as displayed in Figure 4.12 opens up Window 8 of Figure 4.1 when pressed. This window is essentially meant for confirmation if the clicked picture displayed is acceptable for saving the encodings of the face (refer to Figure 4.13). The instance of this button is termed as `clickpic`, incorporated in function `window8()` of the code whereas Window 9 of Figure 4.1 exists in function `window9(imagefrencoding)` of the code. The Click Pic button extracts the pixel values of just the face cropped out of the whole camera frame and passes it to function `window9(imagefrencoding)`. Here, the 2 buttons `yesbutton` and `nobutton` are of prime importance. If the image is not acceptable, then `nobutton` invokes function `no()` which destroys the current widgets and returns to the previous window, Window 7 of Figure 4.1 by invoking the function `window8()` and if the image is acceptable, then the `yesbutton` invokes a function `yes(encodingimage)` which saves the encodings of the image, destroys the current widgets and returns to Window 6 of Figure 4.1 by invoking the function `window5()`.

Once the above procedure is completed and the GUI returns to Window 6, the previously filled out information in the textfields and the faceencoding is retained, waiting for the save button to be pressed to save the complete information. The save button is instantiated as `save_button` in function `window5(encodeface=None)` which invokes a function `createuser_save_button(encodeface)` as shown in the following snippet:

```
user_file_exists_flag = os.path.exists('autologger_userdetails.txt')
encode_file_exists_flag = os.path.exists('autologger_faceencodings.txt')
            ....
            ....
            nametextfield.delete(0,END)
            matriculationtextfield.delete(0,END)
            projecttextfield.delete(0, END)
```

Listing 4.21: Save user details code

The purpose of the above block of code is to first check the existence of files: autologger_userdetails.txt shown in Figure 4.47 and autologger_faceencodings.txt shown in Figure 4.48. If the files do not exist then create these 2 files automatically and write the user details in autologger_userdetails.txt as: userid,username,userdefaultproject whereas write the face encodings in the file autologger_faceencodings.txt as: [userid,username,faceencoding] which is in a form of a python list. The file creation is usually the case when the code is executed for the very first time.

If the two files already exist, then both the files are read to check if the entered userid already exists in the database. The userid/matriculation number is the only method of uniquely identifying people. If the userid already exists, then the data is not saved, the variables are emptied and information in the entry widgets is removed but if the userid doesn't exist in the database then the information is written in the corresponding files, the variables are emptied, the widgets for the current window is destroyed and the GUI returns to previous window Window 5 of Figure 4.1 by invoking function `window4()`. As previously mentioned in disclaimer (page: iv), the data stored in autologger_faceencodings.txt file is not human readable as the data storage is in binary format (shown in Figure 4.48). This is achieved by reading contents from the file using `rb` (read binary mode) and writing data into the file by opening it in `wb` (write binary mode) and inserting data using `.dump()` method from `pickle` library.

Delete User

The delete user is meant to remove the entries of a particular registered user from the database through the GUI thus declaring the individual unauthorised. According to the original software requirement specifications, there is only one feature: delete user, which erases the user details from the database completely. But during the implementation, another feature was added to archive the userdetails. This feature proves to be useful when the same person wants to access the laboratory again in future, so it saves time and effort of not registering him from scratch rather adding the same details to the main database with one button click. Due to time constraints, only the archive feature has been added which deletes the user details from the main database but saves it in another file. The process of retrieving the details from the archive database to the main database has not been implemented in this attendance tracker software.

The delete user window corresponds to Window 9 of Figure 4.1 incorporated within the function `window6()` in the code. The GUI window is invoked by `delete_user_button` which lies in the function `window4()` of the code. The following snippet is the content of the function `window6()`:

```
deletefilelist = []
with open('autologger_userdetails.txt','r') as deletefile:
   deletefilelist = deletefile.read().splitlines()
   ....
   ....
global dropmenu
dropmenu = OptionMenu(root,mymenu,*deletefilelist)
dropmenu.place(x = 379,y = 228,width = 350)
```

Listing 4.22: Delete user window code

The point of interest in the above code is that the function reads the autologger_userdetails.txt file to display the list of registered users in the drop-menu. Additionally, the 2 buttons: delete and archive invoke functions: `delete_item_button(mymenu)` and `archive_item_button(mymenu)` to carry out further instructions. The GUI presenting the set of registered users in drop-menu is shown in Figure 4.45. By default, the option "Select Name" appears on the drop-menu.

```
itemtodelete = mymenu.get()
idno = itemtodelete.split(',')[0].strip('\n').strip(' ')
....
```

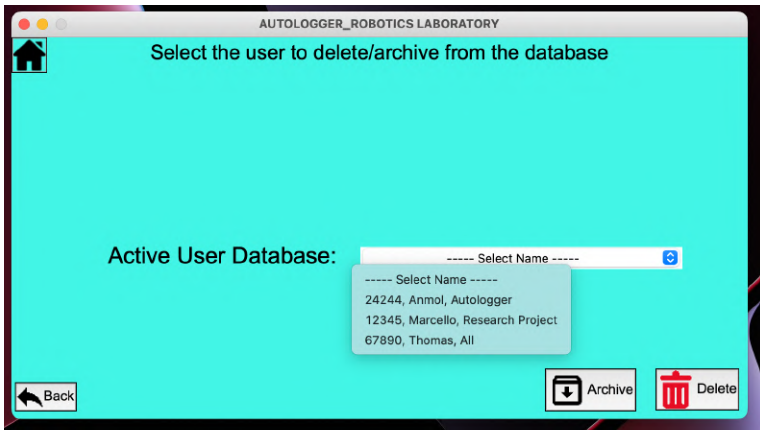


Figure 4.45.: Delete/Archive users drop-down menu.

```
....
dropmenu.destroy()
window4()
```

Listing 4.23: Code to delete a user from database

The above snippet is acquired from the function `delete_item_button(mymenu)` in the code. It obtains the current data selected in the drop-menu using a `StringVar()` object called `mymenu` and extracts the matriculation number in the variable `idno`. Then, the function checks for the existence of the 2 files: autologger_userdetails.txt and autologger_faceencodings.txt by storing a boolean `True` or boolean `False` in the variables named `user_file_exists_flag` and `encode_file_exists_flag`. If the file exists, it reads them in 2 seperate variables `users` and `enclist` respectively. Once this is done, the same 2 files are overwritten by opening them in write mode and all the data is transferred back except for that entry which contains the selected userid to be deleted. After the entry is deleted from both the files, all the current window widgets are destroyed and the GUI returns to the previous window: Window 5 of Figure 4.1 by invoking the function `window4()`.

```
itemtoarchive = mymenu.get()
idno = itemtoarchive.split(',')[0].strip('\n').strip(' ')
....
....
dropmenu.destroy()
window4()
```

Listing 4.24: Code to archive a user from database

The above snippet is obtained from the function `archive_item_button(mymenu)` in the code. The working principle of this function is quite similar to the one above. In this function, there is an additional check carried out for the existence of the file: autologger_archive.txt apart from the above mentioned files: autologger_userdetails.txt and autologger_faceencodings.txt. It obtains the data to be archived from the drop-menu using `StringVar()` object associated with it, reads and stores all the data from the two files in two different variables `users` and `enclist` respectively. The two files are overwritten and the data is transferred back except for the one which has to be deleted from the main database (both files) but to be archived in the separate file. Those entries to be archived are appended in another variable called `tobestored` which is nothing but a list and the data from both the files are appended to this variable. If the archive file doesn't exist, it creates a new archive file automatically and dumps the data from this variable in that file, else it reads the contents of the archive file in a variable called `locallist` which is a list, appends the data from the variable `tobestored` to this `locallist`, overwrites the autologger_archive.txt file by opening it in write mode and then dumps all the contents of the variable `locallist` to it. The case of creating a new file usually occurs when there is no backup taken for the main database in the past.

Autologger Parameters

The autologger parameters window corresponds to Window 10 of Figure 4.1, invoked by `autologger_parameters_button` in `window4()`. The implementation of the logic for this window is contained in the functions `window7()` and `save_autologger_parameters()`.

```
with open('autologger_timedetails.txt','r') as afile:
      listlocal = afile.read()
....
....
global save_button2
save_button2 = Button(root, text = 'Save', command = lambda: save_autologger_parameters(
    logintypespin.get(),logindurspin.get(),officehrspin.get(),officeminspin.get()), font = ('
    Arial',15),borderwidth=3,highlightbackground='black', image = saveuserphotoimage, compound
     = LEFT,background='white')
   save_button2.place(x = 710,y = 360)
```

Listing 4.25: Autologger parameters window code

The above given snippet is from the function `window7()` which contains the definition of the all the widgets(spinbox, labels, buttons) used in designing of this window. Besides the widget creation, this function also reads the file named: autologger_timedetails.txt as shown in Figure 4.46, automatically built at the start by the code with some default values. The purpose of this window is to set the value of certain parameters which is in the hands of the admin for ex: login mode type, minimum duration between one login/logout cycle and laboratory closing time which is implemented by saving these values received from the corresponding spinbox into the file. The ideology behind using a file system is that storing the values in a file is deemed safe in case of an unprecedented power failure, the device can retrieve the latest values from the file again which won't be the case if values aren't stored permanently in a file. Besides this, the advantage of using file is that if the admin doesn't change the values on this window the code can still function normally using the default values stored in this particular file.

Once the file autologger_timedetails.txt is read, the 4 values (stored in order): login mode type, minimum duration between one login/logout cycle, and laboratory closing time (value of hour and minute) is extracted. Inbuilt python features such as `StringVar()` and `IntVar()` are used to set the latest values in the corresponding spinboxes thus displaying the current set values to the user on the GUI and also to extract the values from the spinbox if changed by the admin.

```
logintypeval = logtype
logindurval = dura
officehrval1 = office1
officehrval2 = office2

with open('autologger_timedetails.txt', 'w') as file:
    file.write(f'{logintypeval},{logindurval},{officehrval1},{officehrval2}')
....
....
officeminutetext.destroy()
window4()
```

Listing 4.26: Saving autologger parameters

Once the save button corresponding to `save_button2` in function `window7()` is pressed, it receives the four values of the spinboxes using the `.get()` method and passes them into a function called: `save_autologger_parameters(logtype,dura,office1,office2)`. The primary purpose of this function is to overwrite the autologger_timedetails.txt file with the new received values from the GUI and the secondary function is to remove all the entries from the autologger_tempfile.txt which don't have the "LOGIN" status in their description. This reason behind this to allow the users to log in again who have logged out of the laboratory for that particular day.

```
def home():
    if backhomereturnbutton == 'window2':
        login_button.destroy()
        system_preferences_button.destroy()
        aboutus_button.destroy()
        active_login_button.destroy()
        window1(image1)
    ....
    ....
```

Listing 4.27: Home button function

The above excerpt is the `home()` function which basically destroys all the widgets of the current window being displayed and displays Window HOME of fig: 4.1 by invoking the function `window1(image1)`. This is performed by if-elif-else statements where the value of a variable is matched to a string, unique to each window. The block of code for the corresponding match is then executed.

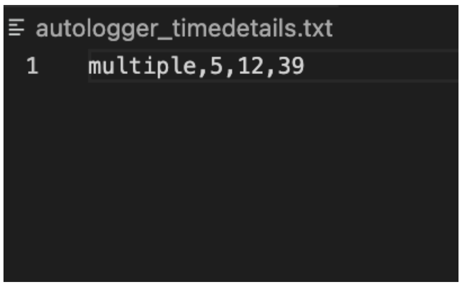
```
def back():
    if backhomereturnbutton == 'window2':
        login_button.destroy()
        system_preferences_button.destroy()
        aboutus_button.destroy()
        active_login_button.destroy()
        window1(image1)
    ....
    ....
```

Listing 4.28: Back button function

Similar to the `home()` function, the above given `back()` function works by destroying the widgets of the current window in view and then displaying the previous window by invoking the corresponding function.

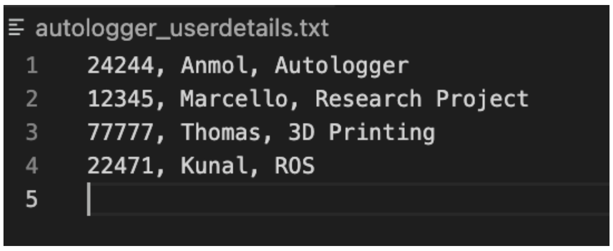
Last but not least, the point of entry of the code is defined under `if __name__ == '__main__':`

Note: To read the full code, please refer to Appendix A: Autologger Code.



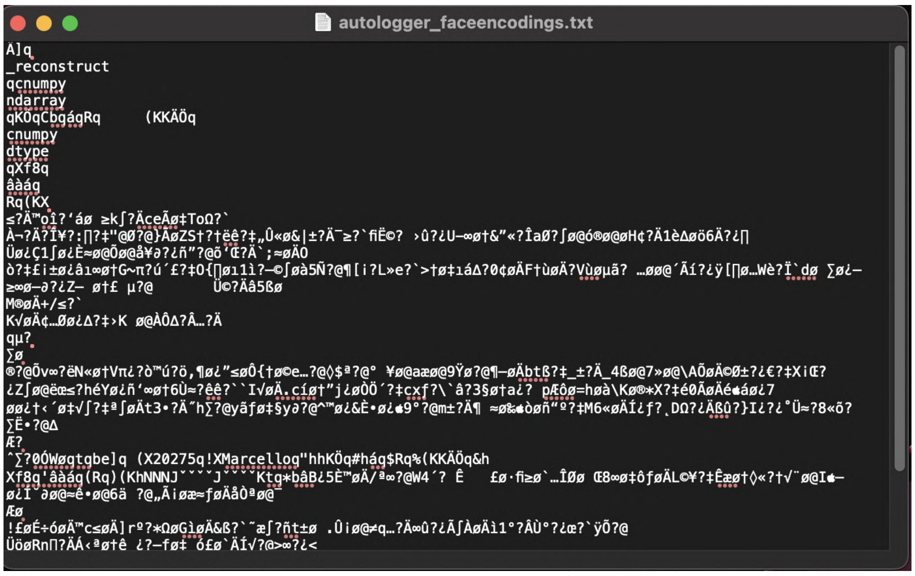
```
autologger_timedetails.txt
multiple,5,12,39
```

Figure 4.46.: autologger_timedetails.txt file.



```
autologger_userdetails.txt
24244, Anmol, Autologger
12345, Marcello, Research Project
77777, Thomas, 3D Printing
22471, Kunal, ROS
```

Figure 4.47.: autologger_userdetails.txt file.



autologger_faceencodings.txt

```
Ä]q
_reconstruct
qcnumpy
ndarray
qKÖqCbgágRq      (KKÄÖq
cnumpy
dtype
qXf8q
âàág
Rq(KX
≤?Ä™oî?‘áø ≥k∫?ÄceÃø‡ToΩ?`
À¬?Ä?Î¥?:∏?‡"@Ø?@}ÃøZS†?†ëê?‡„Û«ø&|±?Ä¯≥?`ﬁË©? ›û?¿U–∞ø†&”«?ÎaØ?∫ø@ó®ø@øH¢?Ä1èΔøö6Ä?¿∏
Üø¿Ç1∫ø¿È≈ø@Õø@å¥∂?¿ñ”?@õ‘Œ?Ä`;≈øÄO
ò?‡£i±ø¿âı∞ø†G~π?ú´£?‡O{∏øı1ì?—©∫øà5Ñ?@¶[i?L»e?`>†ø‡ıáΔ?0¢øÄF†ùøÄ?Vùøµã? …øø@´Ãí?¿ÿ[∏ø…Wè?Ï`dø ∑ø¿–
≥∞ø–∂?¿Z– ø†£ µ?@       Ü©?Äâ5ßø
M®øÄ+/≤?`
K√øÄ¢…Øø¿Δ?‡›K ø@ÀÔΔ?Â…?Ä
qµ?
∑ø
®?@Õv∞?ëN«ø†Vπ¿?ò™ú?ö,¶ø¿”≤øÔ{†ø©e…?@◊$ª?@° ¥ø@aæø@9Ÿø?@¶–øÄbtß?‡_±?Ä_4ßø@7»ø@\AÕøÄ©Ø±?¿€?‡XiŒ?
¿Z∫ø@ëœ≤?héYø¿ñ‘∞ø†6Ù≈?êê?``I√øÄ.cíø†”j¿øÒÖ´?‡cxƒ?\`â?3§ø†a¿? pÆôø=høà\Kø®*X?‡é0ÃøÄé￫áø¿7
øø¿†‹´ø‡√∫?‡ª∫øÄt3•?Ä˝h∑?@yãƒø‡§y∂?@^™ø¿&È•ø¿￫9°?@m±?Ä¶ ≈ø‰￫òøñ“º?‡M6«øÄÍ¿ƒ?¸DΩ?¿ÄßÛ?}I¿?¿˚Ü≈?8«õ?
∑Ë•?@Δ
Æ?
ˆ∑?0ÓWøqtqbe]q (X20275q!XMarcelloq"hhKÖq#háq$Rq%(KKÄÖq&h
Xf8q'âàág(Rq)(KhNNNJˇˇˇˇJˇˇˇˇKtq*bâB¿5È™øÄ/ª∞?@W4´? Ê   £ø·ﬁ≥ø`…ÎØø Œ8∞ø‡ôƒøÄL©¥?‡Êæø†◊«?†√¨ø@I￫–
ø¿Ϊ∂ø@≈ê•ø@6ä ?@„Ãiøæ≈ƒøÄåÒªø@¯
Æø
!£øÉ÷óøÄ™c≤øÄ]rº?*ΩøGìøÄ&ß?`˜æ∫?ñt±ø .Ûiø@≠q…?Ä∞û?¿Ã∫ÀøÄì1°?ÂÙ°?¿œ?`ÿÕ?@
ÜöøRn∏?ÄÁ‹ªø†ê ¿?–ƒø‡ óƒø`ÄÍ√?@>∞?¿<
```

Figure 4.48.: autologger_faceencodings.txt file.

5. Optimisation

The term optimisation refers to the best and most efficient use of the available resources at any given point of time. Optimisation is even utilized in software engineering by modifying the code in a way to improve it's quality and efficiency such that it becomes smaller in size, consumes lesser memory, executes faster and uses less computational power.

The chapter 4: Implementation as discussed above elaborates on the final optimised execution of the attendance tracker software system. The initial version of the code was developed using models namely:

- **Face Detection and Face Recognition:** carried out using the face-recognition python library developed by Adam Geitgey [26] built using the original dlib c++ library. In the attendance tracker software, this pythonic library runs the face detection using HOG-based algorithm whereas the face-recognition part using a deep convolutional neural network built on original dlib's state of the art face recognition. The reason to choose this python library over other existing face detection and face recognition models as discovered during the research [38, 39, 40, 41] is based on the number of model parameters and accuracy. Below shown in the Table 5.1 are the different face recognition models discovered during the research, their corresponding parameters and accuracies:

Table 5.1.: Face-recognition models along with their model parameters and accuracies.

Model Name	Model Parameters	Accuracy
Face-recognition python (a version of ResNet-34 Network)	in between (0.46M - 0.66M)	99.38%
Sphereface	22.671M	98.8321%
face-recognition-mobilefacenet-arcface	0.993M	99.50%
face-recognition-resnet100-arcface	65.131M	99.77%
face-recognition-resnet34-arcface	34.129M	99.65%
face-recognition-resnet50-arcface	43.576M	99.80%
facenet-20180408-102900	23.469M	99.14%

As the final target of this thesis is to deploy the model on an embedded system, the factor of computational power has to be considered, as these embedded system computers are incorporated with processors of low power and the extent of computations they can handle at any given point of time is far limited than a normal Personal Computer (PC). This is where the model parameters come into the picture. From the Table 5.1 above, it is evident that the face-recognition python library has lowest number of parameters, where lower number of parameters means lesser computation it needs to perform. This way, the face-recognition library was defined as a benchmark to start off the development of the project under the assumption that it would perform the best on the chosen embedded system hardware. Apart from this, the other reason to include it was that the alpha version of this project being developed back in 2019 by the former master students were also using the same models to solve the problem, the package is very easy to use and implement it in code, it is very widely accepted and used python library all around the world (Github repository: 45.6k stars), the python library packages the face detection and face recognition models in a very neat and organised manner as well as the documentation for the related library over the internet is very well organised and structured.

- **Hand detection and Gesture Recognition:** carried out using the Mediapipe Hands library. Mediapipe has its own pipeline of implementing hand detection and hand landmarks tracking as discussed in chapter 2: Theoretical Background. Palm detection is done by model: BlazePalm, whereas the hand landmark detection is carried out by the model: BlazeHand. Based on the findings from the literature review conducted, there was no other model found except the Mediapipe python library to detect and track hand gestures which gives the user an edge to use custom gestures without training a model, thus making it one of the primary reason to use it in the development of attendance tracker software system. Other factors that drew me to use this library in the code include its simplicity of usage and implementation, its widespread acceptance and use throughout, and the organized, structured documentation that is available online.

After obtaining the desired software requirement specifications and selecting the models, the complete software was developed from scratch on a normal PC: MacOS. Once the software development was complete, it was tested first on the PC itself (MacBook Pro) which delivered satisfactory result of **1.921 FPS rate** from the camera running all the models together. Subsequently, it was tested on the target hardware: Raspberry pi 4 where there was a drastic drop in the **FPS rate** from **1.921** to **0.781** with a very predominant visual lag in the camera frame. The lag occurring in the camera frames was principally due to the AI models running in the background of the camera frame which took a significant amount of time to process the data and deliver the output. Upon further investigation and testing, the fact that the face recognition module causing all the lag was crystal clear. So now, the code was supposed to be optimised so as to obtain an acceptable FPS rate from the camera while the models were running in the background.

Upon further research, alternatives to face recognition library was found. At this point, a face detection model named: BlazeFace offered by Mediapipe was chosen on trial basis. Besides this, upon reviewing the code it was discovered that face encodings were being calculated in every frame which was an overkill. So the algorithm was redesigned to optimise the code by implementing the new face detection model. Once the face has been detected, the encodings are calculated once until the same person is there in the frame and checked against the database. After the person goes out of the frame, the detections and encodings are reset and waits for a new face detection so that the encodings can be calculated again. To have a new face recognition, the existing person in the frame has to move out of the frame first and then the new person has to enter the frame for detection and recognition. The following code snippets (face detection and face recognition respectively) which belongs to Window 3 of Figure 4.1 or function: `window3()` in the code is the optimisation process for the code.

```python
blazeface_location_results = face_detection.process(recoloredimg)
y1, x2, y2, x1 = facerecognitionfun(blazeface_location_results,wid,hei)
livefeedlocation = [(y1,x2,y2,x1)]
```

Listing 5.1: Face detection optimisation code

```python
if x1 and counterframe == 0 and (not person_confirmed):
    livefeedencoding = fr.face_encodings(recoloredimg,livefeedlocation)[0]
    ....
    ....
    ....
elif x1 and counterframe > 0 and counterframe < 3:
    matches = fr.compare_faces(new_encodeface_list, livefeedencoding, tolerance = 0.5)
    ....
    ....
    ....
    counterframe += 1
elif x1 and (counterframe == 3 or person_confirmed):
    counterframe = 0
    ....
    ....
    ....
```

Listing 5.2: Face recognition optimisation code

Once the code was optimised with the new implementation methodology, it was tested out again on the normal PC (MacBook Pro) where the results were drastically improved with an **FPS rate** of **15.717**. Whereas, testing the optimised code on the target hardware also yielded sufficiently reasonable results of **5.543**, an increase in **FPS rate** by **609.731%** thus achieving the goal of running the software application on an embedded system. The detailed calculations and values are explained in the next chapter 6: Experiments and Results.

6. Experiments & Results

The experiments, results, software execution steps, and initial impressions of the assembled embedded system constitute the four sections of this chapter. The experiment section quantifies the extensive tests carried out both before and after the optimisation process for performance and time analysis. The results part calculates the improvement observed after optimisation, while the third portion provides the user with a step-by-step instruction on how to execute the software. Last but not least, the final section displays the picture of finished built embedded device. The sole purpose of conducting the optimisation process in this thesis is because the final implementation of the code happens on the embedded system so it needs to be as efficient as possible.

Table 6.1.: MacBook Pro (15-inch, 2017) Technical Specifications.[42]

Model Name	MacBook Pro (15-inch, 2017)
Processor Name	Quad-Core Intel Core i7
Processor Speed	2.8GHz, Turbo Boost upto 3.8GHz
Number of Processors	1
Number of Cores	4
RAM(Memory)	16GB of 2133 MHz LPDDR3
Storage	256GB SSD
Camera	720p FaceTime HD Camera
Video Graphics Card	AMD Radeon Pro 555 with 2GB of GDDR5 memory and automatic graphics switching Intel HD Graphics 630
Display Resolution	15,4-inch diagonal (2880 × 1800)

6.1. Experiments

This portion has been divided into 2 subsections: performance analysis and time analysis. Performance analysis quantifies the FPS rate of the camera during user login before and after the optimisation, while the latter part measures the time taken by major code sub modules before and after the optimisation procedure. The figures listed in the following tables are for 2 hardware: Macbook Pro (specification shown in Table 6.1) and Raspberry Pi 4 (target embedded system). Since the code was primarily implemented on a Mac before being tested on the Raspberry Pi, the figures for the Mac PC device are also included.

Inference speed analysis and time analysis were specifically taken into account because the first test of the software on the embedded system before optimisation detected significant visual lag and notable time consumption to process user login/logout, thus making the functionality of the very system almost unfeasible. The discovery of this visual lag or drop in FPS values prompted a thorough experiment on key code sub-modules to determine which segment took up the most time because it was significantly reducing the FPS values, which requires to be improved. Therefore, a time analysis was done to find the defect, and then an inference speed study was performed to determine how significantly faster the performance was following the optimisation step.

6.1.1. Performance Analysis

The following Table 6.2, displays the average FPS values for 100 frames of major code sub modules before optimisation in mac. The performance analysis was conducted at each step of integration of various modules in the back-end to observe the drop of FPS rate. At the end it was discovered that the FPS values plummeted from 29.630 (for importing the camera using OpenCV) to 4.379 (for complete back-end logic without integration of the GUI). Although the drop was quite significant (approx 85.221%, it was satisfactory enough to carry out user login/logout without any serious trouble with visual lag. At the time of completion of the back-end part with a final FPS of 4.379, optimisation didn't seem to be required.

Table 6.2.: Average FPS rates for major code sub-modules in MacBook Pro before optimisation.

Software Code Sub Modules	**Avg FPS Rate (100 Frames)**	**Hardware**
Camera	29.630 ± 4.493	MacBook Pro
Camera + Face detection + Face recognition	14.786 ± 1.297	MacBook Pro
Camera + hand landmark detection + gesture recognition	14.872 ± 1.0585	MacBook Pro
Camera + face detection + face recognition + hand landmark detection + gesture recognition	4.426 ± 0.418	MacBook Pro
Complete backend (without Tkinter)	4.379 ± 0.393	MacBook Pro
Camera on Tkinter	29.847 ± 6.0201	MacBook Pro
Camera on Tkinter	30.0439 ± 3.605	Raspberry Pi 4

The Table 6.3 shown below illustrates the information about the average FPS values (100 frames) for the complete attendance tracker software system tested out on mac and raspberry pi before and after optimisation.

Table 6.3.: Average FPS rates in MacBook Pro and Raspberry Pi 4 before and after optimisation.

Hardware	**Before Optimisation Avg FPS Rate (100 Frames)**	**After Optimisation Avg FPS Rate (100 Frames)**
MacBook Pro	1.921 ± 0.0595 FPS	15.717 ± 3.575 FPS
Raspberry Pi 4	0.781 ± 0.103 FPS	5.543 ± 2.604 FPS

6.1.2. Time Analysis

In order to identify the bottleneck and optimise that section of the code, this section examines the time required by the major code sub modules. It is separated into two categories: Peak time analysis (before and after optimisation and for both hardware: Macbook Pro and raspberry pi 4) comprises all those code segments that execute only once during initial execution and can't really be optimised because they are essential modules. The other is the average time analysis for 100 frames (before and after optimisation, for both hardware: MacBook Pro and Raspberry Pi 4), which investigates the sections of the software code linked to the camera and the visual lag seen on the target hardware.

The Table 6.4 shows peak time analysis for MacBook Pro: Before and After Optimisation.

Table 6.4.: Peak time analysis for major code sub-modules in Macbook Pro before and after optimisation.

Software Code Sub Modules	**Peak Time(sec) Before Opt.**	**Peak Time(sec) After Opt.**
Libraries (fresh reboot)	3.618	3.741
Libraries (no reboot)	2.733	2.343
Image icons	0.262	0.257
Mediapipe Frameworks	2.146E-06	2.14E-06

The Table 6.5 illustrates the average time analysis for MacBook Pro: Before and After Optimisation.

The Table 6.6 displays the peak time analysis for Raspberry Pi 4: Before and After Optimisation.

Table 6.5.: Average time analysis for major code sub-modules in MacBook Pro before and after optimisation.

Software Code Sub Modules	**Avg Time(sec) Before Opt.**	**Avg Time(sec) After Opt.**
Face-recognition function	7.830E-06 ± 3.602E-06	2.0514E-05 ± 2.224E-06
Hand gesture function	0.0288 ± 0.00371	0.0292 ± 0.00243
Face detection	0.446 ± 0.0256	0.00329 ± 3.72E-04
Face recognition	0.0214 ± 0.00316	0.0071 ± 0.0097
Hand detection	0.0297 ± 0.00458	
Face encodings	0.0202 ± 0.00266	
Camera only on Tkinter	0.0242 ± 0.00383	

Table 6.6.: Peak time analysis for major code sub-modules in Raspberry Pi 4 before and after optimisation.

Software Code Sub Modules	**Peak Time(sec) Before Opt.**	**Peak Time(sec) After Opt.**
Libraries (fresh reboot)	24.518	36.405
Libraries (no reboot)	6.446	6.489
Image Icons	0.846	0.825
Mediapipe Frameworks	6.676E-06	7.391E-06

Table 6.7.: Average time analysis for major code snippets in Raspberry Pi 4 before and after optimisation.

Software Code Sub Modules	**Avg Time(sec) Before Opt.**	**Avg Time(sec) After Opt.**
Face-recognition function	1.630E-05 ± 1.406E-06	6.134E-05 ± 1.461E-05
Hand gesture function	0.171 ± 0.0185	0.165 ± 0.0252
Face detection	0.548 ± 0.0139	0.0154 ± 0.00479
Face recognition	0.554 ± 0.0232	0.0954 ± 0.210
Hand detection	0.165 ± 0.0267	
Face encodings	0.5329 ± 0.0768	
Camera only on Tkinter	0.0368 ± 0.0818	

The Table 6.7 shows the average time analysis for Raspberry Pi 4: Before and After Optimisation.

6.2. Results

6.2.1. Performance Analysis Results

The following calculation is to estimate the percentage drop of FPS rate from mac to raspberry pi 4 before optimisation:

$$\%_{drop,BeforeOptimisation} = \left(\frac{FPS_{Mac} - FPS_{RPi}}{FPS_{Mac}}\right) x100\% = \left(\frac{1.921 - 0.781}{1.921}\right) x100\% = \mathbf{59.344\%}$$

OR

$$FPSdecreasefactor = \frac{FPS_{Mac,BeforeOptimisation}}{FPS_{RPi,BeforeOptimisation}} = \frac{1.921}{0.781} = \mathbf{2.46}$$

This calculation is to approximate the percentage drop of FPS rate from mac to raspberry pi 4 after the optimisation step:

$$\%_{drop,AfterOptimisation} = \left(\frac{FPS_{Mac} - FPS_{RPi}}{FPS_{Mac}}\right) x100\% = \left(\frac{15.717 - 5.543}{15.717}\right) x100\% = \mathbf{64.732\%}$$

OR

$$FPSdecreasefactor = \frac{FPS_{Mac,AfterOptimisation}}{FPS_{RPi,AfterOptimisation}} = \frac{15.717}{5.543} = \mathbf{2.83}$$

This computation is to evaluate the percentage increase of FPS rate in mac after optimisation:

$$\%_{FPSincrease,mac} = \left(\frac{FPS_{Mac,new} - FPS_{Mac,old}}{FPS_{Mac,old}}\right) x100\% = \left(\frac{15.717 - 1.921}{1.921}\right) x100\% = \mathbf{718.168\%}$$

OR

$$FPSincreasefactor = \frac{FPS_{Mac,AfterOptimisation}}{FPS_{Mac,BeforeOptimisation}} = \frac{15.717}{1.921} = \mathbf{8.18}$$

This calculation is to evaluate the percentage increase of FPS rate in raspberry Pi 4 after optimisation:

$$\%_{FPSincrease,rpi} = \left(\frac{FPS_{RPi,new} - FPS_{RPi,old}}{FPS_{RPi,old}}\right) x100\% = \left(\frac{5.543 - 0.781}{0.781}\right) x100\% = \mathbf{609.731\%}$$

OR

$$FPSincreasefactor = \frac{FPS_{RPi,AfterOptimisation}}{FPS_{RPi,BeforeOptimisation}} = \frac{5.543}{0.781} = \mathbf{7.1}$$

The evaluation of FPS rates concludes that the performance of the attendance tracker system after optimisation improved by a factor of 7.1 on raspberry pi 4 while an improvement by a factor of 8.18 was noticed on Macbook Pro. On the other hand, there is drop in the performance of the code by a factor of 2.46 when deployed from Macbook Pro to raspberry pi 4 before optimisation while a drop by a factor of 2.83 when the code is deployed from Macbook Pro to raspberry pi 4 after optimisation.

6.2.2. Time Analysis Results

General formulas used:

$$\%improvement = \left(\frac{newvalue - oldvalue}{oldvalue}\right) x100\%$$

$$Improvementfactor = \frac{newvalue}{oldvalue}$$

$$\%_{timeimprovement,mac} = \left(\frac{(Time_{FD,AO} + Time_{FR,AO}) - (Time_{FD,BO} + Time_{FR,BO})}{(Time_{FD,BO} + Time_{FR,BO})}\right) x100\%$$

where FD stands for Face Detection, FR stands for Face Recognition, BO stands for Before Optimisation and AO stands for After Optimisation.

$$\%_{timeimprovement,mac} = \left(\frac{(0.00329 + 0.0071) - (0.446 + 0.0214)}{(0.446 + 0.0214)}\right) x100\% = \textbf{-97.78\%}$$

OR

$$Timeimprovementfactor = \frac{Time_{Mac,AfterOptimisation}}{Time_{Mac,BeforeOptimisation}} = \left(\frac{(0.00329 + 0.0071)}{(0.446 + 0.0214)}\right) = \textbf{0.022}$$

$$\%_{timeimprovement,RPi} = \left(\frac{(Time_{FD,AO} + Time_{FR,AO}) - (Time_{FD,BO} + Time_{FR,BO})}{(Time_{FD,BO} + Time_{FR,BO})}\right) x100\%$$

$$\%_{timeimprovement,rpi} = \left(\frac{(0.0154 + 0.0954) - (0.548 + 0.554)}{(0.548 + 0.554)}\right) x100\% = \textbf{-89.95\%}$$

OR

$$Timeimprovementfactor = \frac{Time_{RPi,AfterOptimisation}}{Time_{RPi,BeforeOptimisation}} = \left(\frac{(0.0154 + 0.0954)}{(0.548 + 0.554)}\right) = \textbf{0.101}$$

The values for time analysis have been calculated based on highlighted values in the Table 6.7 and 6.5 only, where red color are the values before optimisation while the green colored ones are after optimisation. The reason to take only face detection and face recognition values into consideration is because they are the only parameters producing an extremely significant impact in the overall improvement of the optimisation results. After conducting the time analysis, it is quite evident from the figures that the improvement obtained on the embedded system is 89.95% faster after optimisation. The overall enhancement in optimisation is more dominant on Macbook Pro as compared to raspberry pi 4 because the mac processor is extremely powerful and faster than the raspberry pi (refer to the Table 6.1 and 3.2 for CPU specifications of the two hardware respectively). Additionally, as the models are CPU intensive and not GPU intensive also explains the fact that optimisation results are much higher on mac as compared to raspberry pi because of mac incorporating a much powerful CPU (2.8GHz) as compared to a lesser powerful CPU of the raspberry pi 4 (1.5GHz).

6.3. List of dependencies & steps for software execution

List of names of image files needed to run the software are as follows:

1. robopic.jpeg
   (Image Atrribution: Robotics Laboratory, HSRW)
2. homebutton.jpg
   (Image Atrribution: www.flaticon.com)
3. settingslogo.png
   (Image Atrribution: www.flaticon.com)
4. loginlogo.png
   (Image Atrribution: www.cleanpng.com)
5. nextlogo.png
   (Image Atrribution: www.flaticon.com)
6. backlogo.png
   (Image Atrribution: www.flaticon.com)
7. createuser.png
   (Image Atrribution: www.freesvg.org)
8. deleteuser.png
   (Image Atrribution: www.freesvg.org)
9. autologgerparameters.png
   (Image Atrribution: www.flaticon.com)
10. adminloginlogo.png
    (Image Atrribution: www.123rf.com)

11. cameraphoto.png
    (Image Atrribution: www.vectorified.com)
12. AboutUs.png
    (Image Atrribution: stock.adobe.com)
13. savefile.png
    (Image Atrribution: www.icon-library.com)
14. deletefile.png
    (Image Atrribution: www.icon-library.com)
15. activelogins.png
    (Image Atrribution: www.pngwing.com)
16. loginconfirm.png
    (Image Atrribution: www.shutterstock.com)
17. yesicon.png
    (Image Atrribution: www.elharrakfonts.com)
18. noicon.png
    (Image Atrribution: www.shutterstock.com)
19. archive.png
    (Image Atrribution: www.iconscout.com)

The master software code file: autologger.py

List of names of files automatically created by the software are as follows:

- autologger_tempfile.txt
- autologger_logfile.txt
- autologger_userdetails.txt
- autologger_faceencodings.txt
- autologger_archive.txt
- autologger_timedetails.txt

Steps to run the software code through terminal:

```
$ git clone https://robotics.hochschule-rhein-waal.de/gitlab/anmol280399/
  attendance-tracker-system.git
$ cd attendance-tracker-system
$ sudo apt-get install python3-venv
$ python3 -m venv autolog
$ source autolog/bin/activate
$ python3 -m pip install --upgrade pip
$ pip install -r requirements.txt

OR

To install the dependencies manually follows these steps:

$ pip install opencv-python
$ pip install face-recognition
$ pip install mediapipe
```

To execute the code through terminal:

```
$ python3 autologger.py
```

To execute the code through terminal in raspberry pi:

```
$ source autolog/bin/activate
$ export DISPLAY=:0.0
$ python3 autologger.py
```

To deactivate the virtual environment, type in the following command in the terminal:

```
$ deactivate
```

To create the requirements.txt file, follow these commands in the terminal:

```
$ pip install pipreqs
$ pipreqs .  --savepath requirements.txt --ignore autolog

OR

$ pip freeze >> requirements.txt
```

Automatic execution of the software on system boot up:
To achieve this on the raspberry pi, Crontab is used. It's a service which schedules certain tasks to be performed by the operating system at the scheduled time. To execute the code on system boot, the following steps are performed:

```
$ cd attendance-tracker-system
$ nano autologger_launch.sh
```

Edit the autologger_launch.sh file as follows:

```
#!/bin/bash
#!/bin/sh
set -xe
echo $DISPLAY
exec 1>log.out 2>&1

Now run the code as follows:

cd ~/attendance-tracker-system
source ./venv/bin/activate
python3 autologger.py
```

After writing the above commands, save and exit the file. Subsequently, execute the below written command it make it an executable:

```
$ chmod +x autologger_launch.sh
```

Now set up the crontab service to schedule the execution of the code on system reboot:

```
$ crontab -e
$ @reboot DISPLAY:=0 /home/autolog/attendance-tracker-system/autologger_launch.
    sh >> log_out.txt 2>&1
```

Now the system can be rebooted to see the results:

```
$ sudo reboot now
```

List of dependencies required to run the software are:

1. face-recognition
2. mediapipe
3. numpy
4. opencv-python
5. Pillow

6.4. Attendance Tracker System: The Embedded Device

Below shown Figure 6.1, is the final assembled embedded device to run the attendance tracker system. The Figure 6.2 shows the device up and running.

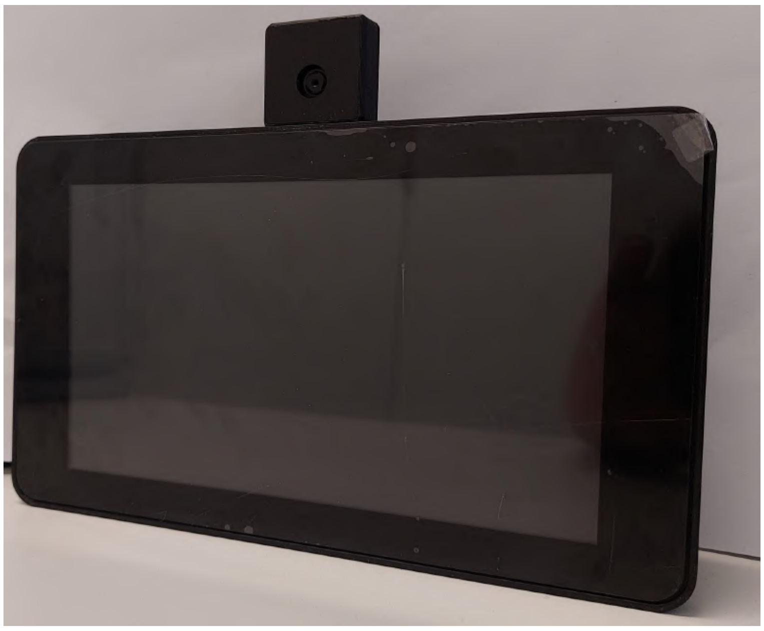


Figure 6.1.: Attendance tracker system embedded device: Assembled

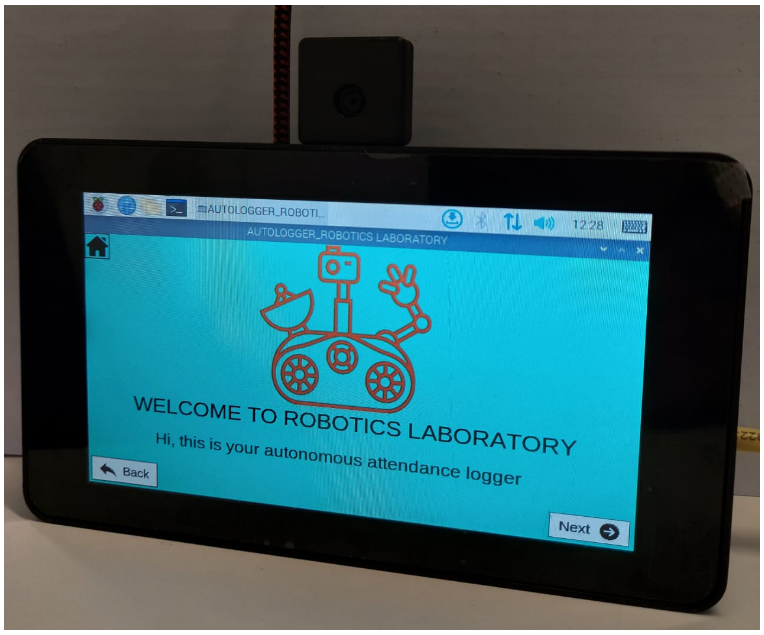

AUTOLOGGER_ROBOTI...
12:28
AUTOLOGGER_ROBOTICS LABORATORY
WELCOME TO ROBOTICS LABORATORY
Hi, this is your autonomous attendance logger
Back
Next

Figure 6.2.: Attendance tracker system embedded device: Running

7. Discussion: Limitations & Future Works

This section of the thesis is an open conversation discussing about the limitations of the attendance tracker software as a whole and recommendations to improve it further in the future. As the proposal section has not been implemented or tested out, it remains as an open suggestion to carry out the new implementation and conclude the final results.

7.1. Limitations

Regarding the attendance tracker software the limitations are as follows:

The face recognition doesn't work when a person is wearing a mask because the model calculates the encoding for the entire face. Thus, having a mask would obscure the face and the encoding cannot be calculated for the entire region of interest. Thus, a mismatch would occur while comparing it against the database. Secondly, the face recognition works perfectly for frontal face position but doesn't work for extreme candid poses because of the model limitations. Furthermore, the face detection don't work if the person is standing far from the camera (further than 1 metre).

The software fails to work under low ambient light conditions as computer vision doesn't work properly in dim light. Computer vision achieves significantly good results under normal lighting conditions but fails under low light as the image captured by the camera becomes dark and very noisy (specs of bright pixels in the dark photo resembling grainy structure, with no clarity or uniformity of color in the image) thus making it difficult for the software to process the image. Concerning facial recognition part of the software, to detect a new face the previous person has to go out of the frame first and then a new person can enter the frame to get a correct recognition. This limitation added to the robustness of the system is due to the optimisation to get a better performance from the device. Another constraint is that the system can be fooled easily by showing up a picture of another authorised person to log into the system because the face recognition model works for 2D images only and doesn't store any depth information in the encoding. By showing up a picture, it extracts the same 2D encoding of the face and compares them to the ones saved in the database thus making it very vulnerable to false authorisation. Moreover, the system doesn't verify whether the person making the hand gesture is the one who is visible in the camera frame. It just validates the face and then recognises the hand and the appropriate gesture to log in or log out the user as needed.

As far as the hardware is concerned, the software itself runs completely locally and has no dependency on the internet connection but the raspberry pi needs an internet connection to keep a track of time based on the selected time zone using global NTP servers. As soon as the raspberry pi is switched off, it loses the count of time because the hardware lacks the presence of an RTC. The absence of the RTC module on raspberry pi is solely to keep the cost of the overall hardware as low as possible.

7.2. Recommendations: New Development As Future Works

Although the final results of developing the complete software from scratch yielded very positive results, it was optimised just enough to get a good performance results on the embedded system. Nevertheless, this code can be continued further as a separate thesis topic to carry out intensive optimisations in those aspects which went untouched during my thesis due to time constraints. This could be achieved by implementing a more object oriented programming approach, or by using threads, or by conducting a memory and CPU optimisation. Focusing on these aspects could make the system perform better on the same hardware specifications.

This attendance tracker software could be developed further to make the system fool proof. As 2D information of the face is a loop hole, an additional depth information is required, which could make the system more secure and less vulnerable to shortcomings. It requires a hardware upgrade of having a stereo camera being able perceive the depth information and use this as an additional parameter to protect system's vulnerability. [43] This could be established by using 3D face recognition to recognise human faces with more accuracy, variable facial positions and expressions. The existing problem of the software system of its inability to recognise the gesture coming from the same person standing in front of the camera frame could be solved utilising a pose estimation model. With the help of this, the attempt of deceiving and logging in/out of the system by impersonating another authorised personnel could be eliminated by confirming that the pose for the hand gesture and the face viewable in the camera frame belong to the same human body. Moreover, the system could be connected to the local university network to send a monthly backup of the digital attendance sheet to the server or to access the tracker system remotely. The hardware issue of time keeping could be solved by connecting the RPi using a PoE connection which would power the system as well as provide internet access thus solving the problem. Another way of addressing the problem is by adding an RTC module to the raspberry pi.

Last but not least, testing out the software on a better hardware, with a much powerful CPU as compared to the current embedded system could also yield overall better performance results.

8. Conclusion

To sum up, the main target of this dissertation to develop a prototype of an attendance tracker system using computer vision based AI deployed on a cost effective, low power consumption embedded system has been accomplished successfully. The system developed reveals itself to be significantly superior to the traditional methods of recording attendance, such as eliminating the carbon footprints that students find tedious to complete. Moreover, the corona pandemic has made this attendance tracker system even more valuable because it is built on the idea of being "as touch free as possible," which greatly reduces the risk of viral transmission while recording attendance, as opposed to the traditional ways of signing a user list. Furthermore, by establishing a digital attendance sheet rather than physically storing hard copies of the user list, this software tracker system also automates the entire process of attendance management system. This design employs an embedded system with a cheap camera and a touchscreen to interact with the software, unlike other methods of tracking attendance like iris scanners and fingerprint scanners, which need the purchase of additional hardware peripherals and raise the cost of the final product.

The device is extremely light, portable, and simple to install because it houses all the electronics in a lovely, lightweight 3D printed enclosure. A power adaptor can be used to power the system. Furthermore, because the embedded system: the Raspberry Pi 4 is built to operate under low power specifications, the system is very energy-efficient. The attendance tracker system is elegantly packaged as a complete piece of software, with a back-end managing the logic and a user-interactive GUI serving as the front-end. The python code has implemented face recognition model to authenticate the user and a subsequent gesture detection model to detect a hand gesture for logging the user in/out of the laboratory as required. In order to improve the system's usability, more functionalities have been added to the software.

The performance of the device becomes extremely important because the thesis's ultimate goal is to deploy the entire code on an embedded device, where limited hardware resources are expected to be used as efficiently as possible. After optimisation, the system records a final roundabout FPS value of 5.543; decent enough to facilitate a smooth user experience. The system's performance with respect to time was improved by 89.95% during the optimisation phase, which is a substantial improvement.

Despite the fact that the thesis's objective was accomplished, there is still room for development in the current system in those areas that were brushed aside due to time constraints, so that it can become even better in the future. Potential recommendations have been discussed and provided in the aforementioned chapter. This creates an endless cycle of development and evolution that draws new programmers to improve the system's current iteration, advancing it to a point where it is almost flawless.

A. Appendix: Autologger Code

```python
# Importing necessary python libraries
import time as t
from tkinter import *
from PIL import ImageTk, Image
import cv2
import face_recognition as fr
import numpy as np
import pickle
import mediapipe as mp
from datetime import date, datetime, time
import os



def exit():
    pass


root = Tk()
root.title('AUTOLOGGER_ROBOTICS LABORATORY')
root.geometry('800x415')
root.resizable(0,0)
root.configure(background = '#42f5e6')
root.protocol('WM_DELETE_WINDOW', exit)

aspectratio = 1280.0/720.0

backbutton = ''
homeclick = ''
returnwindow = ''


# Image Icons
image1 = Image.open("robopic.jpeg")
image1 = ImageTk.PhotoImage(image1.resize((250,250)))

image3 = Image.open("homebutton.jpg")
image3 = ImageTk.PhotoImage(image3.resize((35,35)))

settingphoto = PhotoImage(file = "settingslogo.png")
settingphotoimage = settingphoto.subsample(30,30)

loginphoto = PhotoImage(file = "loginlogo.png")
loginphotoimage = loginphoto.subsample(8,8)
```

```

nextphoto = PhotoImage(file = "nextlogo.png")
nextphotoimage = nextphoto.subsample(15,15)

backphoto = PhotoImage(file = "backlogo.png")
backphotoimage = backphoto.subsample(18,18)

createphoto = PhotoImage(file = "createuser.png")
createphotoimage = createphoto.subsample(6,6)

deletephoto = PhotoImage(file = "deleteuser.png")
deletephotoimage = deletephoto.subsample(6,6)

otherphoto = PhotoImage(file = "autologgerparameters.png")
otherphotoimage = otherphoto.subsample(6,6)

adminphoto = PhotoImage(file = "adminloginlogo.png")
adminphotoimage = adminphoto.subsample(8,8)

clickphoto = PhotoImage(file = "cameraphoto.png")
clickphotoimage = clickphoto.subsample(12,12)

aboutphoto = PhotoImage(file = "AboutUs.png")
aboutphotoimage = aboutphoto.subsample(8,8)

saveuserphoto = PhotoImage(file = "savefile.png")
saveuserphotoimage = saveuserphoto.subsample(12,12)

deleteuserphoto = PhotoImage(file = "deletefile.png")
deleteuserphotoimage = deleteuserphoto.subsample(12,12)

activelogin = PhotoImage(file = "activelogins.png")
activeloginphotoimage = activelogin.subsample(8,8)

confirmlogin = PhotoImage(file = "loginconfirm.png")
confirmloginphotoimage = confirmlogin.subsample(8,8)

yespicsave = PhotoImage(file = "yesicon.png")
yespicphotoimage = yespicsave.subsample(9,9)

nopicsave = PhotoImage(file = "noicon.png")
nopicphotoimage = nopicsave.subsample(9,9)

archivesave = PhotoImage(file = "archive.png")
archivephotoimage = archivesave.subsample(6,6)


loginmodetup = ['single','multiple']

labeltext1 = 'Features of Autonomous Attendance Logger'
feature0 = '1. Runs on 3 AI models: Face Detection, Face Recognition & Gesture Recognition.\n'
feature1 = '2. Login gesture: Thumbs Up.\n'
```

```
feature2 = '3. Logout gesture: Thumbs Down.\n'
feature3 = '4. Gesture control works only when hand is inside the box.\n'
feature4 = '5. The gesture box turns green when hand is present inside else remains red.\n'
feature5 = '6. Login/Logout is successful only when the face is recognised and correct gesture
     is detected.\n'
feature6 = '7. Face recognition works for only one person at a time; not with a mask.\n'
labelauthor = 'Authors and Contributors'
labelauthorstr1 = 'This project is a work of bachelor thesis done by:\n'
labelname = 'Herr Anmol Singh\n'
labelstudyprogram = 'Mechatronic Systems Engineering(B.Sc.)\n'
labelfaculty = 'Faculty of Technology & Bionics\n'
labelthesissupervisor = 'Herr Prof. Dr. Ronny Hartanto(Technische Informatik)\n'
labelcosupervisor = 'Herr Prof. Dr. Matthias Krauledat(Informatik)\n'
labelguidance = 'Herr Mamen Thomas Chembakasseril(Mechanical Engineering, M.Sc.)\n'
labelplace = 'Robotics Laboratory, Rhine Waal University of Applied Sciences,Germany.\n'

global login_logout_flag
login_logout_flag = 0
global flag1
flag1 = -1
global flag2
flag2 = -1
global flag3
flag3 = -1
global flag5
flag5 = -1
global flag6
flag6 = -1
global projecttextfield2
global project_name
global username
username = ''
global userid
userid = ''
global userdefaultproject
userdefaultproject = ''
global confirm_login_logout_flag
confirm_login_logout_flag = 0
global face_login_logout_flag
face_login_logout_flag = 0
global projectstr
projectstr = ''
global default_project
default_project = ''


width = 640
height = 480

labopentime = time(hour = 6, minute = 0, second = 0, microsecond = 000000)

timedetails_fileexists = os.path.exists('autologger_timedetails.txt')
```

```
tempfile_fileexists =   os.path.exists('autologger_tempfile.txt')
logfile_fileexists =   os.path.exists('autologger_logfile.txt')


#Format: logintype,loginlogoutminduration,officehrval1(hr),officehrval2(min) - This is a
    default value.
if timedetails_fileexists == False:
    with open('autologger_timedetails.txt', 'w') as file:
            file.write(f'multiple,15,18,30')

if tempfile_fileexists == False:
    with open('autologger_tempfile.txt', 'w') as file:
        pass

if logfile_fileexists == False:
    with open('autologger_logfile.txt', 'w') as file:
        pass


# Mediapipe frameworks
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

mp_face_detection = mp.solutions.face_detection


# Face Recognition Function
def facerecognitionfun(results,wid,hei):
    previousarea = 0
    y1, x2, y2, x1 = None, None, None, None

    if results.detections:
        for idx, detection in enumerate(results.detections):
            xmin,ymin,width,height = detection.location_data.relative_bounding_box.xmin,
    detection.location_data.relative_bounding_box.ymin,detection.location_data.
    relative_bounding_box.width,detection.location_data.relative_bounding_box.height

            facearea = abs(width*height)

            if facearea>previousarea:
                previousarea = facearea
                y1, x2, y2, x1 = int(ymin * hei/4), (int(xmin * wid/4) + int(width * wid/4)),
    (int(ymin * hei/4) + int(height * hei/4)), int(xmin * wid/4)
    return y1, x2, y2, x1


# Hand gesture function
def handgesturecontrol(imageframe):
        imgframe = cv2.cvtColor(imageframe, cv2.COLOR_BGR2RGB)
        results = hands.process(imgframe)

        if results.multi_hand_landmarks:

            hand_landmarks = results.multi_hand_landmarks[0]

            mp_drawing.draw_landmarks(imageframe,hand_landmarks, mp_hands.HAND_CONNECTIONS,
                                    mp_drawing.DrawingSpec(color = (255,0,0), thickness =  2,
    circle_radius = 4),
                                    mp_drawing.DrawingSpec(color = (200,0,0), thickness =  2,
    circle_radius = 2))
            x0 = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].x * width
            y0 = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].y * height
            y4 = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].y * height

            y_arr = []
            for item in hand_landmarks.landmark:
                y_arr.append(item.y * height)
            y_arr.pop(4)
            y_numbers = np.array(y_arr)

            return y_numbers ,x0, y0, y4
        return None, None, None, None



def delete_item_button(mymenu):

    itemtodelete = mymenu.get()
    idno = itemtodelete.split(',')[0].strip('\n').strip(' ')

    user_file_exists_flag = os.path.exists('autologger_userdetails.txt')
    encode_file_exists_flag = os.path.exists('autologger_faceencodings.txt')

    if user_file_exists_flag and encode_file_exists_flag:
        with open('autologger_userdetails.txt','r') as file1:
            users = file1.readlines()
        with open('autologger_faceencodings.txt','rb') as file2:
            enclist = pickle.load(file2)


    with open('autologger_userdetails.txt','w') as file1:
        for line in users:
            if itemtodelete in line:
                file1.write('')
            else:
                file1.write(line)


    for idx,line in enumerate(enclist):
        if idno == line[0]:
            enclist.pop(idx)

    with open('autologger_faceencodings.txt','wb') as file2:
        pickle.dump(enclist,file2)


    delete_user_label.destroy()
    delete_button.destroy()
    archive_button.destroy()
    userlist.destroy()
    dropmenu.destroy()
    window4()



def archive_item_button(mymenu):

    itemtoarchive = mymenu.get()
    idno = itemtoarchive.split(',')[0].strip('\n').strip(' ')

    archive_file_exists_flag = os.path.exists('autologger_archive.txt')

    user_file_exists_flag = os.path.exists('autologger_userdetails.txt')
    encode_file_exists_flag = os.path.exists('autologger_faceencodings.txt')

    if user_file_exists_flag and encode_file_exists_flag:
        with open('autologger_userdetails.txt','r') as file1:
            users = file1.readlines()
        with open('autologger_faceencodings.txt','rb') as file2:
            enclist = pickle.load(file2)


    tobestored = []


    with open('autologger_userdetails.txt','w') as file1:
        for line in users:
            if itemtoarchive in line:
                tobestored.append(list(line))
                file1.write('')
            else:
                file1.write(line)

    for idx,line in enumerate(enclist):
        if idno == line[0]:
            tobestored[0].append(enclist.pop(idx))


    with open('autologger_faceencodings.txt','wb') as file2:
        pickle.dump(enclist,file2)

    locallist = []

    if archive_file_exists_flag:
        with open('autologger_archive.txt','rb') as filereading:
            locallist = pickle.load(filereading)

        with open('autologger_archive.txt','wb') as file3:
            locallist.append(tobestored)
            pickle.dump(locallist,file3)

    elif archive_file_exists_flag == False:
            with open('autologger_archive.txt','wb') as file3:
                pickle.dump(tobestored,file3)


    delete_user_label.destroy()
    delete_button.destroy()
    archive_button.destroy()
    userlist.destroy()
    dropmenu.destroy()
    window4()



def confirm_login():

    with open('autologger_timedetails.txt', 'r') as timefile:
        loclist = timefile.read()

    loginmode = loclist.split(',')[0].strip('\n').strip(' ')

    global confirm_login_logout_flag
    confirm_login_logout_flag = 1
    global login_logout_flag
    global face_login_logout_flag

    global projectstr
    global default_project
    projectstr = projecttextfield2.get()
    global currenttime
    global currentdate
    global flag1
    global flag2
    global flag3
    global flag5
    global flag6
    global project_name

    if login_logout_flag == 1 and face_login_logout_flag == 1:          #LOGIN

        if confirm_login_logout_flag == 1:

            if flag1 == 0 and flag2 == 1:

                if projectstr != '' and loginmode == 'single':
                    projecttextfield2.delete(0,END)
                    currentdate = datetime.now().date()
                    currenttime = datetime.now().time()
                    with open('autologger_tempfile.txt', 'a') as f:
                        f.write(f'{currentdate}: LOGIN, {name}, {currenttime}, {projectstr}\n')

                    with open('autologger_logfile.txt', 'a') as file:
                        file.write(f'{currentdate}: LOGIN, {name}, {currenttime}, {projectstr}\
    n')
                    login_logout_flag = 0
                    face_login_logout_flag = 0
                elif projectstr != '' and loginmode == 'multiple':
                    currentdate = datetime.now().date()
                    currenttime = datetime.now().time()
                    projecttextfield2.delete(0,END)
                    with open('autologger_tempfile.txt', 'a') as f:
                        f.write(f'{currentdate}: LOGIN, {name}, {currenttime}, {projectstr}\n')

                    with open('autologger_logfile.txt', 'a') as file:
                        file.write(f'{currentdate}: LOGIN, {name}, {currenttime}, {projectstr}\
    n')
                    login_logout_flag = 0
                    face_login_logout_flag = 0
                elif projectstr == '' and loginmode == 'single':
                    currentdate = datetime.now().date()
                    currenttime = datetime.now().time()
                    projecttextfield2.delete(0,END)
                    with open('autologger_tempfile.txt', 'a') as f:
                        f.write(f'{currentdate}: LOGIN, {name}, {currenttime}, {
    default_project}\n')
                    with open('autologger_logfile.txt', 'a') as file:
                        file.write(f'{currentdate}: LOGIN, {name}, {currenttime}, {
    default_project}\n')
                    login_logout_flag = 0
                    face_login_logout_flag = 0
                elif projectstr == '' and loginmode == 'multiple':
                    currentdate = datetime.now().date()
                    currenttime = datetime.now().time()
                    projecttextfield2.delete(0,END)
                    with open('autologger_tempfile.txt', 'a') as f:
                        f.write(f'{currentdate}: LOGIN, {name}, {currenttime}, {
    default_project}\n')
                    with open('autologger_logfile.txt', 'a') as file:
                        file.write(f'{currentdate}: LOGIN, {name}, {currenttime}, {
    default_project}\n')
                    login_logout_flag = 0
                    face_login_logout_flag = 0
                else:
                    pass

    elif login_logout_flag == 2 and face_login_logout_flag == 1:        #LOGOUT

        if confirm_login_logout_flag == 1:

            if flag3 == 1 and flag5 == 0 and flag6 == 1:

                if projectstr != '' and loginmode == 'single':
                    currentdate = datetime.now().date()
                    currenttime = datetime.now().time()
                    projecttextfield2.delete(0,END)
                    with open('autologger_tempfile.txt', 'r') as f:
                        lineslist = f.readlines()
                    with open('autologger_tempfile.txt','w') as ff:
                        for line in lineslist:
                            if f'{currentdate}: LOGIN, {name},' in line.strip('\n'):
                                ff.write(f'{currentdate}: {name}\n')
                            else:
                                ff.write(line)
                    with open('autologger_logfile.txt', 'a') as fff:
                        fff.write(f'{currentdate}: LOGOUT, {name}, {currenttime}, {
    project_name}\n')
                    login_logout_flag = 0
                    face_login_logout_flag = 0

                elif projectstr != '' and loginmode == 'multiple':
                    currentdate = datetime.now().date()
                    currenttime = datetime.now().time()
                    projecttextfield2.delete(0,END)
                    with open('autologger_tempfile.txt', 'r') as f:
                        lineslist = f.readlines()

                    with open('autologger_tempfile.txt','w') as ff:
                        for line in lineslist:
                            if f'{currentdate}: LOGIN, {name},' in line.strip('\n'):
                                ff.write('')
                            else:
                                ff.write(line)

                    with open('autologger_logfile.txt', 'a') as fff:
                        fff.write(f'{currentdate}: LOGOUT, {name}, {currenttime}, {
    project_name}\n')
                    login_logout_flag = 0
                    face_login_logout_flag = 0

                elif projectstr == '' and loginmode == 'single':
                    currentdate = datetime.now().date()
                    currenttime = datetime.now().time()
                    projecttextfield2.delete(0,END)
                    with open('autologger_tempfile.txt', 'r') as f:
                        lineslist = f.readlines()
                    with open('autologger_tempfile.txt','w') as ff:
                        for line in lineslist:
                            if f'{currentdate}: LOGIN, {name},' in line.strip('\n'):
                                ff.write(f'{currentdate}: {name}\n')
                            else:
                                ff.write(line)
                    with open('autologger_logfile.txt', 'a') as fff:
                        fff.write(f'{currentdate}: LOGOUT, {name}, {currenttime}, {project_name}\n')
                    login_logout_flag = 0
                    face_login_logout_flag = 0

                elif projectstr == '' and loginmode == 'multiple':
                    currentdate = datetime.now().date()
                    currenttime = datetime.now().time()
                    projecttextfield2.delete(0,END)
                    with open('autologger_tempfile.txt', 'r') as f:
                        lineslist = f.readlines()

                    with open('autologger_tempfile.txt','w') as ff:
                        for line in lineslist:
                            if f'{currentdate}: LOGIN, {name},' in line.strip('\n'):
                                ff.write('')
                            else:
                                ff.write(line)

                    with open('autologger_logfile.txt', 'a') as fff:
                        fff.write(f'{currentdate}: LOGOUT, {name}, {currenttime}, {project_name}\n')
                    login_logout_flag = 0
                    face_login_logout_flag = 0

                else:
                    pass

    projectlabel2.destroy()
    projecttextfield2.destroy()
    confirmbutton.destroy()
    video.destroy()
    cap.release()
    cv2.destroyAllWindows()
    window1(image1)


def home():

    if backhomereturnbutton == 'window2':
        login_button.destroy()
        system_preferences_button.destroy()
        aboutus_button.destroy()
        active_login_button.destroy()
        window1(image1)

    elif backhomereturnbutton == 'window3':
        projectlabel2.destroy()
        projecttextfield2.destroy()
        confirmbutton.destroy()
        video.destroy()
        cap.release()
        cv2.destroyAllWindows()
        window1(image1)

    elif backhomereturnbutton == 'window4':
        create_user_button.destroy()
        delete_user_button.destroy()
        autologger_parameters_button.destroy()
        labelsetting.destroy()
        window1(image1)

    elif backhomereturnbutton == 'window5':
        global username
        username = ''
        global userid
        userid = ''
        global userdefaultproject
        userdefaultproject = ''
        facepicture.destroy()
        facepicturelabel.destroy()
        projecttextfield.destroy()
        projectlabel.destroy()
        matriculationtextfield.destroy()
        matriculationlabel.destroy()
        nametextfield.destroy()
        namelabel.destroy()
        user_label.destroy()
        emptylabel.destroy()
        save_button.destroy()
        window1(image1)

    elif backhomereturnbutton == 'window6':
        delete_user_label.destroy()
        delete_button.destroy()
        archive_button.destroy()
        userlist.destroy()
        dropmenu.destroy()
        window1(image1)

    elif backhomereturnbutton == 'window7':
        change_setting_label.destroy()
        logindur_label.destroy()
        officehr_label.destroy()
        loginmode_label.destroy()
        save_button2.destroy()
        emptylabel2.destroy()
        logintypespin.destroy()
        logindurspin.destroy()
        officehrspin.destroy()
        officeminspin.destroy()
        logindurtext.destroy()
        officehourtext.destroy()
        officeminutetext.destroy()
        window1(image1)

    elif backhomereturnbutton == 'window8':
        if username != '' and userid != '' and userdefaultproject != '':
            username = ''
            userid = ''
            userdefaultproject = ''

        videolabel2.destroy()
        clickpic.destroy()
        cap2.release()
        cv2.destroyAllWindows()
        window1(image1)

    elif backhomereturnbutton == 'admin_password_window':
        passwordwidget.destroy()
        password_textfield.destroy()
        admin_login_button.destroy()
        window1(image1)

    elif backhomereturnbutton == 'about_software':
        heading1.destroy()
        text1.destroy()
        text2.destroy()
        heading2.destroy()
        window1(image1)

    elif backhomereturnbutton == 'active_users_window':
        active_users_label.destroy()
        active_users_listbox.destroy()
        scrollbar_listbox.destroy()
        window1(image1)

    elif backhomereturnbutton == 'window9':
        if username != '' and userid != '' and userdefaultproject != '':
            username = ''
            userid = ''
            userdefaultproject = ''
        surelabel.destroy()
        yesbutton.destroy()
        nobutton.destroy()
        labelpic.destroy()
        window1(image1)



def back():

    if backhomereturnbutton == 'window2':
        login_button.destroy()
        system_preferences_button.destroy()
        aboutus_button.destroy()
        active_login_button.destroy()
        window1(image1)

    elif backhomereturnbutton == 'window3':
        projectlabel2.destroy()
        projecttextfield2.destroy()
        confirmbutton.destroy()
        video.destroy()
        cap.release()
        cv2.destroyAllWindows()
        window2()

    elif backhomereturnbutton == 'window4':
        create_user_button.destroy()
        delete_user_button.destroy()
        autologger_parameters_button.destroy()
        labelsetting.destroy()
        admin_password_window()

    elif backhomereturnbutton == 'window5':
        global username
        username = ''
        global userid
        userid = ''
        global userdefaultproject
        userdefaultproject = ''
        facepicture.destroy()
        facepicturelabel.destroy()
        projecttextfield.destroy()
        projectlabel.destroy()
        matriculationtextfield.destroy()
        matriculationlabel.destroy()
        nametextfield.destroy()
        namelabel.destroy()
        user_label.destroy()
        emptylabel.destroy()
        save_button.destroy()
        window4()

    elif backhomereturnbutton == 'window6':
        delete_user_label.destroy()
        delete_button.destroy()
        archive_button.destroy()
        userlist.destroy()
        dropmenu.destroy()
        window4()

    elif backhomereturnbutton == 'window7':
        change_setting_label.destroy()
        logindur_label.destroy()
        officehr_label.destroy()
        loginmode_label.destroy()
        save_button2.destroy()
        emptylabel2.destroy()
        logintypespin.destroy()
        logindurspin.destroy()
        officehrspin.destroy()
        officeminspin.destroy()
        logindurtext.destroy()
        officehourtext.destroy()
        officeminutetext.destroy()
        window4()

    elif backhomereturnbutton == 'window8':
        videolabel2.destroy()
        clickpic.destroy()
        cap2.release()
        cv2.destroyAllWindows()
        window5()

    elif backhomereturnbutton == 'admin_password_window':
        passwordwidget.destroy()
        password_textfield.destroy()
        admin_login_button.destroy()
        window2()

    elif backhomereturnbutton == 'about_software':
        heading1.destroy()
        text1.destroy()
        text2.destroy()
        heading2.destroy()
        window2()

    elif backhomereturnbutton == 'active_users_window':
        active_users_label.destroy()
        active_users_listbox.destroy()
        scrollbar_listbox.destroy()
        window2()

    elif backhomereturnbutton == 'window9':
        surelabel.destroy()
        yesbutton.destroy()
        nobutton.destroy()
        labelpic.destroy()
        window8()



def yes(encodingimage):
    surelabel.destroy()
    yesbutton.destroy()
    nobutton.destroy()
    labelpic.destroy()

    global encode
    encode = fr.face_encodings(encodingimage)

    window5(encode)


def no():
    surelabel.destroy()
    yesbutton.destroy()
    nobutton.destroy()
    labelpic.destroy()
    window8()



def window9(imagefrencoding):
    videolabel2.destroy()
    clickpic.destroy()
    cap2.release()
    cv2.destroyAllWindows()

    global backhomereturnbutton
    backhomereturnbutton = 'window9'

    global surelabel
    surelabel = Label(root,text='Are you sure to save encodings for this picture?',font=('
     Arial',25),bg='#42f5e6')
    surelabel.pack()

    cvimage3 = cv2.cvtColor(imagefrencoding,cv2.COLOR_BGR2RGB)
    cvimage3 = cv2.resize(cvimage3,(300,300))
    img345 = Image.fromarray(cvimage3)

    imgtk3 = ImageTk.PhotoImage(image=img345)

    global labelpic
    labelpic = Label(root, image=imgtk3)
    labelpic.photo = imgtk3
    labelpic.pack()

    global yesbutton
    yesbutton = Button(root,command = lambda: yes(imagefrencoding),bg='#42f5e6',
     highlightbackground='black', image = yespicphotoimage, compound = LEFT,background='white')
    yesbutton.place(x = 519,y = 350)

    global nobutton
    nobutton = Button(root,command = no,bg='#42f5e6', highlightbackground='black', image =
     nopicphotoimage, compound = LEFT,background='white')
    nobutton.place(x = 225,y = 350)
    root.update()

```

```


def window8():

    global backhomereturnbutton
    backhomereturnbutton = 'window8'

    facepicture.destroy()
    facepicturelabel.destroy()
    projecttextfield.destroy()
    projectlabel.destroy()
    matriculationtextfield.destroy()
    matriculationlabel.destroy()
    nametextfield.destroy()
    namelabel.destroy()
    user_label.destroy()
    emptylabel.destroy()
    save_button.destroy()

    global clickpic
    clickpic = Button(root, text='Click Pic',command = lambda: window9(displayimg), font=('
    Arial',15),bg='#42f5e6', highlightbackground='black', image = clickphotoimage, compound =
    LEFT,background='white')
    clickpic.place(x = 620,y = 360)

    global videolabel2
    videolabel2 = Label(root)
    videolabel2.pack()

    global cap2
    cap2 = cv2.VideoCapture(0)
    width = cap2.get(3)
    height = cap2.get(4)

    t.sleep(1)

    with mp_face_detection.FaceDetection(min_detection_confidence=0.5) as face_detection:

        while cap2.isOpened():
            ret,frame2 = cap2.read()

            frame2 = cv2.resize(frame2,(0,0),None,fx = 0.25,fy = 0.25)        #320,180
            img = cv2.cvtColor(frame2, cv2.COLOR_BGR2RGB)

            blazeface_location_results = face_detection.process(img)

            y1, x2, y2, x1 = facerecognitionfun(blazeface_location_results,width,height)
            livefeedlocation = [(y1,x2,y2,x1)]


            if livefeedlocation:
                y1,x2,y2,x1 = livefeedlocation[0]

                global newy1
                global newx2
                global newy2
                global newx1


                global fixedwidth
                global fixedheight
                global midpointx
                global midpointy

                fixedwidth = 100
                fixedheight = 100

                midpointx = (x1+x2)/2
                midpointy = (y1+y2)/2

                newy1,newx2,newy2,newx1 = int(midpointy - fixedheight/2),int(midpointx + fixedwidth/2),int(midpointy + fixedheight/2),int(midpointx - fixedwidth/2)

                displayimg = frame2[newy1:newy2,newx1:newx2]

                cv2.line(img,(x1,y1),(x1+5,y1),(255,255,0),2)
                cv2.line(img,(x1,y1),(x1,y1+5),(255,255,0),2)
                cv2.line(img,(x2,y1),(x2,y1+5),(255,255,0),2)
                cv2.line(img,(x2,y1),(x2-5,y1),(255,255,0),2)
                cv2.line(img,(x2,y2),(x2,y2-5),(255,255,0),2)
                cv2.line(img,(x2,y2),(x2-5,y2),(255,255,0),2)
                cv2.line(img,(x1,y2),(x1+5,y2),(255,255,0),2)
                cv2.line(img,(x1,y2),(x1,y2-5),(255,255,0),2)

                cvimage2 = cv2.resize(img, (int(aspectratio*350),350))
                img2 = Image.fromarray(cvimage2)
                imgtk2 = ImageTk.PhotoImage(image=img2)
                videolabel2.config(image = imgtk2)
                root.update()
            else:
                cvimage2 = cv2.resize(img, (int(aspectratio*350),350))
                img2 = Image.fromarray(cvimage2)
                imgtk2 = ImageTk.PhotoImage(image=img2)
                videolabel2.config(image = imgtk2)
                root.update()



def save_autologger_parameters(logtype,dura,office1,office2):

    logintypeval = logtype
    logindurval = dura
    officehrval1 = office1
    officehrval2 = office2
```

```python


    #Format: logintype,loginduration,officehrval1(hr),officehrval2(min)
    with open('autologger_timedetails.txt', 'w') as file:
        file.write(f'{logintypeval},{logindurval},{officehrval1},{officehrval2}')

    with open('autologger_tempfile.txt', 'r') as f:
        lineslist = f.readlines()

    with open('autologger_tempfile.txt','w') as ff:
        for line in lineslist:
            if f'LOGIN' in line.strip('\n'):
                ff.write(line)
            else:
                ff.write('')



    change_setting_label.destroy()
    logindur_label.destroy()
    officehr_label.destroy()
    loginmode_label.destroy()
    save_button2.destroy()
    emptylabel2.destroy()
    logintypespin.destroy()
    logindurspin.destroy()
    officehrspin.destroy()
    officeminspin.destroy()
    logindurtext.destroy()
    officehourtext.destroy()
    officeminutetext.destroy()
    window4()



def window7():

    with open('autologger_timedetails.txt','r') as afile:
        listlocal = afile.read()

    val1 = listlocal.split(',')[0].strip('\n').strip(' ')
    val2 = int(listlocal.split(',')[1].strip('\n').strip(' '))
    val3 = int(listlocal.split(',')[2].strip('\n').strip(' '))
    val4 = int(listlocal.split(',')[3].strip('\n').strip(' '))

    my_var1 = StringVar()
    my_var2 = IntVar()
    my_var3 = IntVar()
    my_var4 = IntVar()

    global backhomereturnbutton
    backhomereturnbutton = 'window7'
    create_user_button.destroy()
    delete_user_button.destroy()
    autologger_parameters_button.destroy()
    labelsetting.destroy()

    global change_setting_label
    change_setting_label = Label(root, text='Change autologger settings',font=('Arial',30),bg='
    #42f5e6')
    change_setting_label.grid(row = 0, column = 2,sticky = E, columnspan = 2)

    global emptylabel2
    emptylabel2 = Label(root,bg='#42f5e6',width = 1)
    emptylabel2.grid(row = 1,column = 0)

    global logindur_label
    logindur_label = Label(root, text='Select login/logout duration: ',font=('Arial',25),bg='
    #42f5e6')
    logindur_label.grid(row = 2,column = 2, sticky = W)

    global officehr_label
    officehr_label = Label(root, text='Select office hours: ',font=('Arial',25),bg='#42f5e6')
    officehr_label.grid(row = 3,column = 2, sticky = W)

    global loginmode_label
    loginmode_label = Label(root, text='Select login mode: ',font=('Arial',25),bg='#42f5e6')
    loginmode_label.grid(row = 4,column = 2, sticky = W)

    global logintypespin
    logintypespin = Spinbox(root,values = loginmodetup,textvariable = my_var1,font=('Arial',15)
    )
    logintypespin.grid(row = 4, column = 3,sticky = W)
    my_var1.set(val1)

    global logindurspin
    logindurspin = Spinbox(root,from_ = 5, to = 60, increment= 5,width = 5,textvariable =
    my_var2,font=('Arial',15))
    logindurspin.grid(row = 2, column = 3, sticky = W)
    my_var2.set(val2)

    global logindurtext
    logindurtext = Label(root,text='min',font=('Arial',14),bg='#42f5e6')
    logindurtext.grid(row = 2, column = 3, sticky = W,padx = 80)

    global officehrspin
    officehrspin = Spinbox(root,from_= 6,to = 23, width = 5,textvariable = my_var3,font=('
    Arial',15))
    officehrspin.grid(row = 3, column = 3, sticky = W)
    my_var3.set(val3)

    global officehourtext
    officehourtext = Label(root,text='hours',font=('Arial',14),bg='#42f5e6')
    officehourtext.grid(row = 3, column = 3, sticky = W,padx = 80)

    global officeminspin
    officeminspin = Spinbox(root,from_= 0, to = 59, width = 5,textvariable = my_var4,font=('Arial',15))
    officeminspin.grid(row = 3, column = 4, sticky = W)
    my_var4.set(val4)

    global officeminutetext
    officeminutetext = Label(root,text='min',font=('Arial',14),bg='#42f5e6')
    officeminutetext.grid(row = 3, column = 4, sticky = W,padx = 80)

    global save_button2
    save_button2 = Button(root, text = 'Save', command = lambda: save_autologger_parameters(logintypespin.get(),logindurspin.get(),officehrspin.get(),officeminspin.get()),
                          font = ('Arial',15),borderwidth=3,highlightbackground='black', image = saveuserphotoimage, compound = LEFT,background='white')
    save_button2.place(x = 660,y = 350)



def window6():

    deletefilelist = []
    with open('autologger_userdetails.txt','r') as deletefile:
        deletefilelist = deletefile.read().splitlines()

    global backhomereturnbutton
    backhomereturnbutton = 'window6'

    global mymenu
    mymenu = StringVar()
    mymenu.set('----- Select Name -----')

    deletefilelist.insert(0,'----- Select Name -----')

    create_user_button.destroy()
    delete_user_button.destroy()
    autologger_parameters_button.destroy()
    labelsetting.destroy()

    global delete_user_label
    delete_user_label = Label(root, text='Select the user to delete/archive from the database',font=('Arial',22),bg='#42f5e6')
    delete_user_label.pack(side='top')

    global delete_button
    delete_button = Button(root, text = 'Delete',command = lambda: delete_item_button(mymenu), font = ('Arial',15),borderwidth=3,highlightbackground='black', image = deleteuserphotoimage, compound = LEFT,background='white')
    delete_button.place(x=650,y=350)

    global archive_button
    archive_button = Button(root, text = 'Archive',command = lambda: archive_item_button(
    mymenu), font = ('Arial',15),borderwidth=3,highlightbackground='black', image =
    archivephotoimage, compound = LEFT,background='white')
    archive_button.place(x=490,y=350)

    global userlist
    userlist = Label(root, text='Active User Database: ',font=('Arial',25),bg='#42f5e6')
    userlist.place(x = 100,y = 220)

    global dropmenu
    dropmenu = OptionMenu(root,mymenu,*deletefilelist)
    dropmenu.place(x = 440,y = 228,width = 350)



def createuser_save_button(encodeface):

    global username
    global userid
    global userdefaultproject


    checkflag1 = -1
    checkflag2 = -1
    readencodefacelist = []

    user_file_exists_flag = os.path.exists('autologger_userdetails.txt')
    encode_file_exists_flag = os.path.exists('autologger_faceencodings.txt')

    if user_file_exists_flag and encode_file_exists_flag:

        with open('autologger_userdetails.txt','r') as f:
            readuserlist = f.readlines()
            for line in readuserlist:
                if userid in line:
                    checkflag1 = 1
                    break
                else:
                    checkflag1 = 0

        with open('autologger_faceencodings.txt','rb') as fileread:
            readencodefacelist = pickle.load(fileread)
            for idx in range(len(readencodefacelist)):
                if userid == readencodefacelist[idx][0]:
                    checkflag2 = 1
                    break
                else:
                    checkflag2 = 0


    if (user_file_exists_flag == False and encode_file_exists_flag == False) or (
    readencodefacelist == [] and readuserlist == []):
        checkflag1 = 0
        checkflag2 = 0


    if username != '' and userid != '' and userdefaultproject != '' and encodeface != None:

        if checkflag1 == 0 and checkflag2 == 0:

            nametextfield.delete(0,END)
            matriculationtextfield.delete(0,END)
            projecttextfield.delete(0, END)

            with open('autologger_userdetails.txt','a') as file1:
                file1.write(f'{userid}, {username}, {userdefaultproject}\n')
            with open('autologger_faceencodings.txt','wb') as file2:
                readencodefacelist.append([userid, username, encodeface[0]])
                pickle.dump(readencodefacelist, file2)


            username = ''
            userid = ''
            userdefaultproject = ''

            facepicture.destroy()
            facepicturelabel.destroy()
            projecttextfield.destroy()
            projectlabel.destroy()
            matriculationtextfield.destroy()
            matriculationlabel.destroy()
            nametextfield.destroy()
            namelabel.destroy()
            user_label.destroy()
            emptylabel.destroy()
            save_button.destroy()
            window4()

        if checkflag1 == 1 and checkflag2 == 1:

            username = ''
            userid = ''
            userdefaultproject = ''

            nametextfield.delete(0,END)
            matriculationtextfield.delete(0,END)
            projecttextfield.delete(0, END)


def window5(encodeface=None):

    global username
    global userid
    global userdefaultproject


    global backhomereturnbutton
    backhomereturnbutton = 'window5'

    create_user_button.destroy()
    delete_user_button.destroy()
    autologger_parameters_button.destroy()
    labelsetting.destroy()

    my_str1 = StringVar()
    my_str2 = StringVar()
    my_str3 = StringVar()

    global user_label
    user_label = Label(root, text='Fill in for registering a new user',font=('Arial',30),bg='
    #42f5e6')
    user_label.grid(row = 0, column = 1, sticky = W, columnspan = 2)

    global emptylabel
    emptylabel = Label(root,bg='#42f5e6',width = 5)
    emptylabel.grid(row = 1, column = 0)

    global namelabel
    namelabel = Label(root, text='NAME: ',font=('Arial',25),bg='#42f5e6')
    namelabel.grid(row = 2, column = 1, sticky = W)

    global nametextfield
    nametextfield = Entry(root, width=35, borderwidth = 2, bg="white", fg = "black",
    textvariable = my_str1)
    nametextfield.grid(row = 2, column = 2, sticky = W)

    global matriculationlabel
    matriculationlabel = Label(root, text='MAT NR.: ', font=('Arial',25),bg='#42f5e6')
    matriculationlabel.grid(row = 3, column = 1, sticky = W)

    global matriculationtextfield
    matriculationtextfield = Entry(root, width=35, borderwidth = 2, bg="white", fg = "black",
    textvariable = my_str2)
    matriculationtextfield.grid(row = 3, column = 2, sticky = W)

    global projectlabel
    projectlabel = Label(root, text='PROJECT: ', font=('Arial',25),bg='#42f5e6')
    projectlabel.grid(row = 4, column = 1, sticky = W)

    global projecttextfield
    projecttextfield = Entry(root, width=35, borderwidth = 2, bg="white", fg = "black",
    textvariable = my_str3)
    projecttextfield.grid(row = 4, column = 2, sticky = W)

    global facepicturelabel
```

```
    facepicturelabel = Label(root, text='FACE PICTURE:  ', font=('Arial',25),bg='#42f5e6')
    facepicturelabel.grid(row = 5, column = 1, sticky = W)

    global facepicture
    facepicture = Button(root, text='Snap Picture',command = window8, font=('Arial',15),bg='
    #42f5e6', highlightbackground='black', image = clickphotoimage, compound = LEFT,background
    ='white',state = 'disabled')
    facepicture.grid(row = 5, column = 2, sticky = W)

    global save_button
    save_button = Button(root, text = 'Save',command = lambda: createuser_save_button(
    encodeface), font = ('Arial',15),borderwidth=3,highlightbackground='black', image =
    saveuserphotoimage, compound = LEFT,background='white')
    save_button.place(x = 660, y = 350)


    def enablebutton(*kwargs):
        global username
        global userid
        global userdefaultproject

        if my_str1.get() != '' and my_str2.get() != ''  and my_str3.get() != '':
            username = my_str1.get()
            userid = my_str2.get()
            userdefaultproject = my_str3.get()
            facepicture.config(state = 'normal')
        else:
            facepicture.config(state = 'disabled')

    my_str3.trace('w', enablebutton)

    if username != '' and userid != '' and userdefaultproject != '':
        my_str1.set(username)
        my_str2.set(userid)
        my_str3.set(userdefaultproject)



def window4():

    global backhomereturnbutton
    backhomereturnbutton = 'window4'

    global backbutton
    backbutton = 'window4'

    global homeclick
    homeclick = 'window4'

    passwordwidget.destroy()
    password_textfield.destroy()
    admin_login_button.destroy()

    global create_user_button
    create_user_button = Button(root, text='Create User', font=('Arial',15),command = window5,
    bg='#42f5e6', highlightbackground='black', image = createphotoimage, compound = LEFT,
    background='white')
    create_user_button.place(x = 315, y = 60)

    global delete_user_button
    delete_user_button = Button(root, text='Delete User', font=('Arial',15),command = window6,
    bg='#42f5e6', highlightbackground='black', image = deletephotoimage, compound = LEFT,
    background='white')
    delete_user_button.place(x = 315, y = 180)

    global autologger_parameters_button
    autologger_parameters_button = Button(root, text='Autologger Parameters', font=('Arial',15)
    ,command = window7,bg='#42f5e6', highlightbackground='black', image = otherphotoimage,
    compound = LEFT,background='white')
    autologger_parameters_button.place(x = 280, y = 300)

    global labelsetting
    labelsetting = Label(root, text='System Preferences',font=('Arial',25),bg='#42f5e6')
    labelsetting.pack()


def active_users_window():

    login_button.destroy()
    system_preferences_button.destroy()
    aboutus_button.destroy()
    active_login_button.destroy()

    global backhomereturnbutton
    backhomereturnbutton = 'active_users_window'

    global active_users_label
    active_users_label = Label(root, text= 'Active Users in Robotics Laboratory',font=('Arial'
    ,25),bg='#42f5e6')
    active_users_label.pack()

    global scrollbar_listbox
    scrollbar_listbox = Scrollbar(root, orient = VERTICAL)
    scrollbar_listbox.place(x = 770,y = 40,height = 320)        #side = RIGHT, fill = BOTH

    global active_users_listbox
    active_users_listbox = Listbox(root,width = 70,height = 15,font=('Arial',13),
    yscrollcommand=scrollbar_listbox.set)
    active_users_listbox.place(x = 50, y = 40)


    with open('autologger_tempfile.txt','r') as f:
        datum = str(datetime.now().date())
        stat = "LOGIN"
        for line in f:
            if datum in line and stat in line:
                active_users_listbox.insert(END, str(line))

    scrollbar_listbox.config(command = active_users_listbox.yview)


def window3():

    with open('autologger_timedetails.txt', 'r') as timefile:
        loclist = timefile.read()

    loginmode = loclist.split(',')[0].strip('\n').strip(' ')
    loginmindur = int(loclist.split(',')[1].strip('\n').strip(' ')) * 60
    office1 = int(loclist.split(',')[2].strip('\n').strip(' '))
    office2 = int(loclist.split(',')[3].strip('\n').strip(' '))
    officehours = time(hour = office1, minute = office2, second = 0, microsecond = 000000)

    with open('autologger_tempfile.txt','r') as f:
        readlist = f.readlines()


    with open('autologger_tempfile.txt','r+') as file:
        todays_date = datetime.now().date()


    #logging out after booting up for old dates.
        for line in file:
            if 'LOGIN' in line:

                datelog = line.split(',')[0].split(':')[0]
                logstatus = line.split(',')[0].split(':')[1].strip()
                namelog = line.split(',')[1].strip()
                projectnameauto = line.split(',')[3].strip('\n').strip()

                if datelog in line and logstatus in line:
                    if str(datelog) != str(todays_date) and str(logstatus) == 'LOGIN':
                        with open('autologger_logfile.txt','a') as secondfile:
                            secondfile.write(f'{datelog}: LOGOUT, {namelog}, {officehours}, {
    projectnameauto}\n')


    #Clearing the tempfile for old dates.
    with open('autologger_tempfile.txt','w') as ff:

        for line in readlist:
            datelog = line.split(',')[0].split(':')[0]

            if datelog != str(datetime.now().date()):
                ff.write('')
            else:
                ff.write(line)
```

```


#Load the facedatabase and encodedfacesdatabase files
with open('autologger_faceencodings.txt', 'rb') as f:
    encodelistknownfaces = pickle.load(f)


new_encodeface_list = []
for item in range(len(encodelistknownfaces)):
    new_encodeface_list.append(encodelistknownfaces[item][2])

global backhomereturnbutton
backhomereturnbutton = 'window3'

login_button.destroy()
system_preferences_button.destroy()
aboutus_button.destroy()
active_login_button.destroy()

global projectlabel2
projectlabel2 = Label(root, text='PROJECT: ', font=('Arial',20),bg='#42f5e6')
projectlabel2.place(x = 130,y = 363)

global confirmbutton
confirmbutton = Button(root, text='Confirm',command = confirm_login,font=('Arial',15), bg='
 #42f5e6',borderwidth=0,highlightbackground='black', image = confirmloginphotoimage,
 compound = RIGHT,background='white')
confirmbutton.place(x = 648,y = 364)

global projecttextfield2
projecttextfield2 = Entry(root, width=35, borderwidth = 2, bg="white", fg = "black", state
 = 'disabled')
projecttextfield2.place(x = 270,y = 370)

global video
video = Label(root)
video.pack()

global cap
cap = cv2.VideoCapture(0)
wid = cap.get(3)
hei = cap.get(4)

previoustime = t.time()     #Not zero
global hands
global projectstr
global confirm_login_logout_flag
global login_logout_flag
global face_login_logout_flag
global flag1
global flag2
global flag3
```

```
    global flag5
    global flag6
    global project_name
    y1, x2, y2, x1 = None, None, None, None
    counterframe = 0
    livefeedencoding = []
    global name
    name  = ''
    person_confirmed = False
    original_name = ''

    with mp_hands.Hands(model_complexity=0, min_detection_confidence=0.5,
    min_tracking_confidence = 0.5)  as hands, mp_face_detection.FaceDetection(
    min_detection_confidence=0.5) as face_detection:

        while True:

            success, imageframe = cap.read()
            imageframe = cv2.flip(imageframe,1)

            #FPS calculation
            currenttimefps = t.time()
            fps = 1/(currenttimefps - previoustime)
            previoustime = currenttimefps

            resizedimg = cv2.resize(imageframe,(0,0),None, fx = 0.25, fy = 0.25)
            recoloredimg = cv2.cvtColor(resizedimg, cv2.COLOR_BGR2RGB)

            recoloredimg.flags.writeable = False
            blazeface_location_results = face_detection.process(recoloredimg)

            y1, x2, y2, x1 = facerecognitionfun(blazeface_location_results,wid,hei)
            livefeedlocation = [(y1,x2,y2,x1)]


            if x1 is None:
                name = ''
                person_confirmed = False


            cv2.putText(imageframe,f'FPS: {(fps)}',(10,40),cv2.FONT_HERSHEY_COMPLEX
    ,0.6,(255,0,0),2)
            cv2.putText(imageframe,'Place any hand in the box!',(65,420),fontFace= cv2.
    FONT_HERSHEY_SIMPLEX,fontScale=0.6,color=(200,0,0),thickness = 2)
            cv2.putText(imageframe,'Hand gesture-control works, when wrist is inside the box'
    ,(10,20),fontFace= cv2.FONT_HERSHEY_SIMPLEX,fontScale=0.6,color=(200,0,0),thickness = 2)
            cv2.putText(imageframe,'Thumbs up: LOGIN!',(10,60),fontFace= cv2.
    FONT_HERSHEY_SIMPLEX,fontScale=0.6,color=(200,0,0),thickness = 2)
            cv2.putText(imageframe,'Thumbs down: LOGOUT!',(10,80),fontFace= cv2.
    FONT_HERSHEY_SIMPLEX,fontScale=0.6,color=(200,0,0),thickness = 2)

            cv2.line(imageframe,(100,200),(115,200),(0,0,255),3)
            cv2.line(imageframe,(100,200),(100,215),(0,0,255),3)
            cv2.line(imageframe,(300,200),(300,215),(0,0,255),3)
            cv2.line(imageframe,(300,200),(285,200),(0,0,255),3)
            cv2.line(imageframe,(100,400),(100,385),(0,0,255),3)
            cv2.line(imageframe,(100,400),(115,400),(0,0,255),3)
            cv2.line(imageframe,(300,400),(285,400),(0,0,255),3)
            cv2.line(imageframe,(300,400),(300,385),(0,0,255),3)

            if x1 and counterframe == 0 and (not person_confirmed):

                livefeedencoding = fr.face_encodings(recoloredimg,livefeedlocation)[0]
                matches = fr.compare_faces(new_encodeface_list, livefeedencoding, tolerance = 0.5)     #Returns a list of true and false against all the encodings; one closest is true.
                facedistance = fr.face_distance(new_encodeface_list,livefeedencoding) #Returns a list of face distances against the whole of database. Smallest one is true.
                matchindex = np.argmin(facedistance)        #Returns the index of the element of the database which is true for the picture.

                if matches[matchindex]:
                    name = encodelistknownfaces[matchindex][1].upper()
                    face_login_logout_flag = 1
                    original_name = encodelistknownfaces[matchindex][1]

                elif not matches[matchindex]:
                    name = 'Unknown'

                counterframe +=1

            elif x1 and counterframe > 0 and counterframe < 3:

                matches = fr.compare_faces(new_encodeface_list, livefeedencoding, tolerance = 0.5)
                facedistance = fr.face_distance(new_encodeface_list,livefeedencoding)
                matchindex = np.argmin(facedistance)

                if matches[matchindex]:
                    a_name = encodelistknownfaces[matchindex][1].upper()
                    original_name = encodelistknownfaces[matchindex][1]

                    if a_name == name:
                        person_confirmed = True
                    else:
                        counterframe = 0
                        name = ''

                elif not matches[matchindex]:
                    name = 'Unknown'
                    person_confirmed = True

                counterframe += 1

            elif x1 and (counterframe == 3 or person_confirmed):
```

```python

            counterframe = 0

            if name == 'Unknown':

                person_confirmed = False

                cv2.line(imageframe,(100,200),(115,200),(0,0,255),3)
                cv2.line(imageframe,(100,200),(100,215),(0,0,255),3)
                cv2.line(imageframe,(300,200),(300,215),(0,0,255),3)
                cv2.line(imageframe,(300,200),(285,200),(0,0,255),3)
                cv2.line(imageframe,(100,400),(100,385),(0,0,255),3)
                cv2.line(imageframe,(100,400),(115,400),(0,0,255),3)
                cv2.line(imageframe,(300,400),(285,400),(0,0,255),3)
                cv2.line(imageframe,(300,400),(300,385),(0,0,255),3)

                y1,x2,y2,x1 = livefeedlocation[0]

                y1,x2,y2,x1 = y1*4,x2*4,y2*4,x1*4
                cv2.rectangle(imageframe,(x1,y1),(x2,y2),(0,0,255),3)
                cv2.putText(imageframe,name,(x1+6,y2-6),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),2)
                cv2.putText(imageframe,'Identity Mismatch!',(int(x1-20),int(y1-40)),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
                cv2.putText(imageframe,'Unauthorized Robotics Lab Personnel',(int(x1-103),int(y1-20)),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)


            else:
                y1,x2,y2,x1 = livefeedlocation[0]
                y_numbers, x0, y0, y4 = handgesturecontrol(imageframe)

                y1,x2,y2,x1 = y1*4,x2*4,y2*4,x1*4

                cv2.rectangle(imageframe,(x1,y1),(x2,y2),(0,255,0),3)
                cv2.putText(imageframe,name,(x1+6,y2-6),cv2.FONT_HERSHEY_COMPLEX,0.5,(255,255,255),2)
                cv2.putText(imageframe,'Identity Verified',(int(x1+25),int(y1-40)),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,255,0),2)
                cv2.putText(imageframe,'Authorized Robotics Lab Personnel',(int(x1-45),int(y1-20)),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,255,0),2)

                with open('autologger_userdetails.txt', 'r') as readmefile:
                    global default_project
                    for item in readmefile:
                        if original_name in item:
                            default_project = item.split(',')[2].strip('\n').strip(' ')

                if y0 != None:
                    if y0 < 400 and y0 > 200 and x0 > 100 and x0 < 300:

                        cv2.line(imageframe,(100,200),(115,200),(0,255,0),3)
                        cv2.line(imageframe,(100,200),(100,215),(0,255,0),3)
                        cv2.line(imageframe,(300,200),(300,215),(0,255,0),3)
                        cv2.line(imageframe,(300,200),(285,200),(0,255,0),3)
                        cv2.line(imageframe,(100,400),(100,385),(0,255,0),3)
                        cv2.line(imageframe,(100,400),(115,400),(0,255,0),3)
                        cv2.line(imageframe,(300,400),(285,400),(0,255,0),3)
                        cv2.line(imageframe,(300,400),(300,385),(0,255,0),3)

                        # Code block to log in the lab on any day
                        if (np.all(y_numbers>y4)):

                            login_logout_flag = 1
                            cv2.putText(imageframe,"Login Gesture Detected",(20,300),cv2.FONT_HERSHEY_COMPLEX,0.8,(0,255,0),2)

                            search_name = str(name)
                            search_date = str(datetime.now().date())
                            search_log_status = 'LOGIN'

                            with open('autologger_tempfile.txt', 'r') as f:
                                flag1 = 0
                                flag2 = 0

                                if f:

                                    for line in f:
                                        word = line.split(':')[1].strip('\n').strip(' ').split(',')[0].strip('\n').strip(' ')
                                        if ((search_date in line and search_name in line and search_log_status in line) or (search_name == word)):
                                            flag1 = 1
                                            break

                                if flag1 == 0:
                                    if (datetime.now().time() > labopentime and datetime.now().time() < officehours):
                                        flag2 = 1

                                    if flag2 == 1:
                                        projecttextfield2.config(state = 'normal')
                                        cv2.putText(imageframe,"Login Successful!",(200,460),cv2.FONT_HERSHEY_COMPLEX,0.8,(0,255,0),2)
                                        currentdate = datetime.now().date()
                                        currenttime = datetime.now().time()
                                    elif flag2 == 0:
                                        cv2.putText(imageframe,"Lab is closed! Can't Login!",(200,460),cv2.FONT_HERSHEY_COMPLEX,0.8,(0,255,0),2)

                                elif flag1 == 1:
                                    pass


                        # Code block to log out of the lab on any day.
                        elif (np.all(y_numbers<y4)):
                            login_logout_flag = 2
                            cv2.putText(imageframe,"Logout Gesture Detected",(20,300),cv2.FONT_HERSHEY_COMPLEX,0.8,(0,255,255),2)
                            search_name = str(name)
                            search_date = str(datetime.now().date())
                            search_log_statusone = 'LOGIN'


                            with open('autologger_tempfile.txt', 'r') as f:
                                flag3 = 0
                                flag5 = 0
                                flag6 = 0

                                for line in f:
                                    if search_date in line and search_name in line and search_log_statusone in line:
                                        flag3 = 1
                                        user_login_time = datetime.strptime(line.split(',')[2].strip('\n').strip(' ').split('.')[0], '%H:%M:%S').time()
                                        final_time = datetime.combine(date.today(), user_login_time)
                                        project_name = line.split(',')[3].strip('\n').strip(' ')
                                        break

                                if flag3 == 0:
                                    cv2.putText(imageframe,"Please Login First!",(200,460),cv2.FONT_HERSHEY_COMPLEX,0.8,(0,255,255),2)

                                elif flag3 == 1:

                                        if (datetime.now().time() > officehours):
                                            flag5 = 1

                                        if flag5 == 0:

                                            global logouttime, timediff
                                            logouttime = datetime.now()
                                            timediff = logouttime - final_time
                                            if timediff.seconds > loginmindur:
                                                flag6 = 1

                                            if flag6 == 1:

                                                if loginmode == 'single':

                                                    cv2.putText(imageframe,"Logout Successful!",(200,460),cv2.FONT_HERSHEY_COMPLEX,0.8,(0,255,255),2)
                                                    currentdate = datetime.now().date()
                                                    currenttime = datetime.now().time()

                                                    elif loginmode == ’multiple’:
                                                        cv2.putText(imageframe,"Logout Successful!",(200,460),cv2.FONT_HERSHEY_COMPLEX,0.8,(0,255,255),2)
                                                        currentdate = datetime.now().date()
                                                        currenttime = datetime.now().time()

                                                    else:
                                                        pass

                                                elif flag6 == 0:
                                                    cv2.putText(imageframe,"Can’t Logout right now!",(200,460),cv2.FONT_HERSHEY_COMPLEX,0.8,(255,0,255),2)


            cvimage = cv2.cvtColor(imageframe,cv2.COLOR_BGR2RGBA)
            cvimage = cv2.resize(cvimage, (int(aspectratio*350),350))
            img = Image.fromarray(cvimage)
            imgtk = ImageTk.PhotoImage(image=img)
            video.config(image = imgtk)
            root.update()



def about_software():

    login_button.destroy()
    system_preferences_button.destroy()
    aboutus_button.destroy()
    active_login_button.destroy()

    global backhomereturnbutton
    backhomereturnbutton = ’about_software’

    global heading1
    heading1 = Label(root, text = labeltext1, font = (’Arial’,18),bg=’#42f5e6’)
    heading1.pack()

    global text1
    text1 = Text(root, font = (’Arial’,11), height = 8,bg=’#ffff80’)
    text1.insert(INSERT,feature0)
    text1.insert(INSERT,feature1)
    text1.insert(INSERT,feature2)
    text1.insert(INSERT,feature3)
    text1.insert(INSERT,feature4)
    text1.insert(INSERT,feature5)
    text1.insert(INSERT,feature6)
    text1.pack()

    global heading2
    heading2 = Label(root, text = labelauthor, font = (’Arial’,20),bg=’#42f5e6’)
    heading2.pack()

    global text2
    text2 = Text(root, font = ('Arial',11), height = 8, width = 70,bg='#ffff80')
    text2.insert(INSERT,labelauthorstr1)
    text2.insert(INSERT,'Name: '+labelname)
    text2.insert(INSERT,'Study Program: '+labelstudyprogram)
    text2.insert(INSERT,'Faculty: '+labelfaculty)
    text2.insert(INSERT,'Thesis Supervisor: '+labelthesissupervisor)
    text2.insert(INSERT,'Thesis Co-supervisor: '+labelcosupervisor)
    text2.insert(INSERT,'Guided By: '+labelguidance)
    text2.insert(INSERT,'Place: '+labelplace)
    text2.pack()



def admin_verification():
    password = '05eg025'
    user_pass = str(password_textfield.get())
    if user_pass == password:
        password_textfield.delete(0, END)
        window4()
    else:
        password_textfield.delete(0, END)



def admin_password_window():

    login_button.destroy()
    system_preferences_button.destroy()
    aboutus_button.destroy()
    active_login_button.destroy()

    global backhomereturnbutton
    backhomereturnbutton = 'admin_password_window'

    global passwordwidget
    passwordwidget = Label(root, text='Please Enter Admin Password',font=('Arial',20),bg='#42
     f5e6')
    passwordwidget.pack()

    global password_textfield
    password_textfield = Entry(root, width=30, borderwidth = 2, bg="white", fg = "black", show
     ='*')
    password_textfield.place(x = 170,y = 220)

    global admin_login_button
    admin_login_button = Button(root, text='Admin Login', font=('Arial',15),command=
     admin_verification, bg='#42f5e6', highlightbackground='black', image = adminphotoimage,
     compound = LEFT,background='white')
    admin_login_button.place(x = 460,y = 205)

```

```


def window2():

    with open('autologger_timedetails.txt', 'r') as timefile:
        loclist = timefile.read()

    office1 = int(loclist.split(',')[2].strip('\n').strip(' '))
    office2 = int(loclist.split(',')[3].strip('\n').strip(' '))
    officehours = time(hour = office1, minute = office2, second = 0, microsecond = 000000)

    with open('autologger_tempfile.txt','r') as f:
        readlist = f.readlines()


    with open('autologger_tempfile.txt','r+') as file:
        todays_date = datetime.now().date()
        momentstime = datetime.now().time()


    #logging out after booting up for old dates.
        for line in file:
            if 'LOGIN' in line:

                datelog = line.split(',')[0].split(':')[0]
                logstatus = line.split(',')[0].split(':')[1].strip()
                namelog = line.split(',')[1].strip()
                projectnameauto = line.split(',')[3].strip('\n').strip()

                if datelog in line and logstatus in line:
                    if str(datelog) != str(todays_date) and str(logstatus) == 'LOGIN':
                        with open('autologger_logfile.txt','a') as secondfile:
                            secondfile.write(f'{datelog}: LOGOUT, {namelog}, {officehours}, {
    projectnameauto}\n')


    #Clearing the tempfile for old dates.
    with open('autologger_tempfile.txt','w') as ff:

        for line in readlist:
            datelog = line.split(',')[0].split(':')[0]

            if datelog != str(datetime.now().date()):
                ff.write('')
            else:
                ff.write(line)


    label0.destroy()
    label1.destroy()
    button1.destroy()
    labeltext.destroy()

    global backhomereturnbutton
    backhomereturnbutton = 'window2'

    global login_button
    login_button = Button(root, text = 'User Login/Logout', font = ('Arial',15), command =
    window3,borderwidth=3,highlightbackground='black', image = loginphotoimage, compound =
    LEFT, background = 'white')
    login_button.place(x = 290,y = 100)

    global system_preferences_button
    system_preferences_button  = Button(root, text = 'System Preferences', font = ('Arial',15),
     command = admin_password_window, borderwidth = 3, highlightbackground='black', image =
    settingphotoimage, compound = LEFT, background = 'white')
    system_preferences_button.place(x = 280,y = 200)

    global aboutus_button
    aboutus_button = Button(root, text = 'About Autonomous Attendance Logger', font = ('Arial'
    ,15),command = about_software,borderwidth=3,highlightbackground='black', image =
    aboutphotoimage, compound = LEFT,background='white')
    aboutus_button.place(x = 200,y = 305)

    global active_login_button
    active_login_button = Button(root, text = 'Active Users',command = active_users_window,
    font = ('Arial',15),borderwidth=3,highlightbackground='black', image =
    activeloginphotoimage, compound = LEFT,background='white')
    active_login_button.place(x = 320,y = 10)


def window1(img):

    global backhomereturnbutton
    backhomereturnbutton = 'window1'

    global label0
    label0 = Label(root, image=img, bg = '#42f5e6')
    label0.pack()

    global label1
    label1 = Label(root, text='WELCOME TO ROBOTICS LABORATORY', font=('Arial',25),bg='#42f5e6')

    label1.pack()

    global labeltext
    labeltext = Label(root, text='Hi, this is your autonomous attendance logger', font=('Arial'
    ,20),bg='#42f5e6')
    labeltext.place(x=110,y= 310)

    global button1
    button1 = Button(root, text='Next', font=('Arial',15),command=window2, bg='#42f5e6',
    borderwidth=0,highlightbackground='black', image = nextphotoimage, compound = RIGHT,
    background='white')
    button1.place(x = 690,y = 372)

home_button = Button(root, image = image3, command = home, bg='#42f5e6',
                    highlightbackground='black',highlightthickness=1, fg = 'black', background
    ='#42f5e6')
home_button.place(x = 0,y = 0)


button2 = Button(root, text='Back', font=('Arial',15),command = back,bg='#42f5e6',
    highlightbackground='black', image = backphotoimage, compound = LEFT,background='white')
button2.place(x = 4,y = 370)

if __name__ == '__main__':
    window1(image1)
    root.mainloop()
```

List of Figures

List of Tables

Listings

Acronyms

3D 3-Dimensional. vii, 12, 14, 20, 77, 78

A Ampere. 23

a.k.a also known as. iv, 27

AI Artificial Intelligence. vi, vii, 2, 5, 6, 19, 63, 78

Avg Average. 66–68

BDSG Bundesdatenschutzgesetz. iv

BGR Blue Green Red. 40

CI/CD Continous Integration/Continous Development. 4

CNN Convolutional Neural Network. 8, 11–13

CPU Central Processing Unit. 71, 77

CSI Camera Serial Interface. 23

CV Computer Vision. vi, 2, 6

DC Direct Current. 23

DPI Display Parallel Interface. 23

DSI Display Serial Interface. 19, 23

FPS Frames Per Second. 8, 63–67, 69, 70, 78

GB Giga Bytes. 20, 23, 65

GHz Giga Hertz. 65, 71

GPCLK General Purpose Clock. 23

GPIO General Purpose Input Output. 19, 23

GPS Global Positioning System. 2

GPU Graphical Processing Unit. 8, 11, 71

GUI Graphical User Interface. ii, 2–4, 17, 20, 22, 27, 29, 31, 33, 34, 37, 38, 50–52, 54–57, 59, 66, 78

HD High Definition. 65

HDMI High Definition Multimedia Interface. 23

HOG Histogram of Oriented Gradients. 6, 7, 9, 22, 62

HSRW Hochschule Rhein Waal. vi

I2C Inter Integrated Circuit. 23

IDE Integrated Development Environment. 4

IRS Intelligent Robotic Services. vi

IVR Interactive Voice Response. 2

MHz Mega Hertz. 65

ML Machine Learning. 11, 14

MP MegaPixel. 23

NTP Network Time Protocol. 77

Opt. Optimisation. 67, 68

PC Personal Computer. 63, 64, 66

PCM Pulse Code Modulation. 23

PLA Polylactic Acid. vii

PoE Power Over Ethernet. 77

PWM Pulse Width Modulation. 23

QR Quick Response. 1

R&D Research and Development. 20

RAM Random Access Memory. 23, 65

RFID Radio Frequency Identification. 1

RGB Red Green Blue. 9, 11–13, 19, 23, 40

RPi Raspberry Pi. 19, 23, 77

RTC Real Time Clock. 77

SD Card Secure Digital Memory Card. 20, 23

SDIO Secure Digital Input Output. 23

SPI Serial Peripheral Interface. 23

SSD Single Shot Multibox Detector. 8, 11, 12, 65

UART Universal Asynchronous Receiver/Trasmitter. 23

UAV Unmanned Aerial Vehicle. vi

UID Unique Identification Number. 1

UML Unified Modeling Language. 21, 22, 24–26, 117

USB Universal Serial Bus. 2, 23

V Volts. 19, 23

WLAN Wireless Local Area Network. 23

Bibliography

[1] B. f. J. Bundesministerium der Justiz. *Bundesdatenschutzgesetz*. URL: https://www.gesetze-im-internet.de/bdsg_2018/index.html. (accessed: 21.07.2022).

[2] F. Masalha and N. Hirzallah. “A students attendance system using QR code”. In: *International Journal of Advanced Computer Science and Applications* 5.3 (2014). URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.683.6728&rep=rep1&type=pdf.

[3] T. Lim, S. Sim, and M. Mansor. “RFID based attendance system”. In: *2009 IEEE Symposium on Industrial Electronics & Applications*. Vol. 2. 2009, pp. 778–782. DOI: 10.1109/ISIEA.2009.5356360. URL: https://ieeexplore.ieee.org/document/5356360.

[4] K. Sudha, S. Shinde, T. Thomas, and A. Abdugani. “Article: Barcode based Student Attendance System”. In: *International Journal of Computer Applications* 119.2 (June 2015), pp. 1–5.

[5] G. Ononiwu and N. Okorafor. “RADIO FREQUENCY IDENTIFICATION (RFID) BASED ATTENDANCE SYSTEM WITH AUTOMATIC DOOR UNIT”. In: 2 (Apr. 2012).

[6] B. K. P. Mohamed and C. V. Raghu. “Fingerprint attendance system for classroom needs”. In: *2012 Annual IEEE India Conference (INDICON)*. 2012, pp. 433–438. DOI: 10.1109/INDCON.2012.6420657.

[7] S. Dey, S. Barman, R. K. Bhukya, R. K. Das, B. C. Haris, S. R. M. Prasanna, and R. Sinha. “Speech biometric based attendance system”. In: *2014 Twentieth National Conference on Communications (NCC)*. 2014, pp. 1–6. DOI: 10.1109/NCC.2014.6811345. URL: https://ieeexplore.ieee.org/document/6811345.

[8] M. S. Uddin, S. Allayear, N. Das, and F. Talukder. “A Location Based Time and Attendance System”. In: *International Journal of Computer Theory and Engineering* (Jan. 2014), pp. 36–38. DOI: 10.7763/IJCTE.2014.V6.832.

[9] K. O. Okokpujie, E. Noma-Osaghae, O. J. Okesola, S. N. John, and O. Robert. “Design and Implementation of a Student Attendance System Using Iris Biometric Recognition”. In: *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2017, pp. 563–567. DOI: 10.1109/CSCI.2017.96.

[10] H. Yang and X. Han. “Face Recognition Attendance System Based on Real-Time Video Processing”. In: *IEEE Access* 8 (2020), pp. 159143–159150. DOI: 10.1109/ACCESS.2020.3007205.

[11] Y. Kawaguchi and T. Shoji. “Face Recognition-based Lecture Attendance System”. In: (Jan. 2005).

[12] A. Raghuwanshi and P. D. Swami. “An automated classroom attendance system using video based face recognition”. In: *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. 2017, pp. 719–724. DOI: 10.1109/RTEICT.2017.8256691.

[13] A. Jha, I. P. G. Shenwai, A. Batra, S. Kotian, and P. Modi. *GesSure – A Robust Face-Authentication enabled Dynamic Gesture Recognition GUI Application*. 2022. DOI: 10.48550/ARXIV.2207.11033. URL: https://arxiv.org/abs/2207.11033.

[14] C. for Disease Control and Prevention. *Science Brief: SARS-CoV-2 and Surface (Fomite) Transmission for Indoor Community Environments*. URL: https://www.cdc.gov/coronavirus/2019-ncov/more/science-and-research/surface-transmission.html. (accessed: 27.07.2022).

[15] Microsoft. *Visual Studio: Code Editor*. URL: https://visualstudio.microsoft.com. (accessed: 27.07.2022).

[16] T. D. C. Tom Preston Werner(founder). *Github*. URL: https://github.com. (accessed: 27.07.2022).

[17] A. Khusid. *Miro: the leading visual collaboration platform*. URL: https://miro.com. (accessed: 27.07.2022).

[18] G. Alder. *Diagrams.net*. URL: https://app.diagrams.net. (accessed: 27.07.2022).

[19] S. Hettiarachchi. *Analysis of different face detection and recognition models for Android*. 2021.

[20] A. Geitgey. *Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning*. URL: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78. (accessed: 04.08.2022).

[21] M. Google. *Mediapipe Face Detection*. URL: https://google.github.io/mediapipe/solutions/face_detection.html. (accessed: 03.08.2022).

[22] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann. *BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs*. 2019. DOI: 10.48550/ARXIV.1907.05047. URL: https://arxiv.org/abs/1907.05047.

[23] G. R. P. .-. CV4AR/VR. *BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs*. URL: https://sites.google.com/view/perception-cv4arvr/blazeface. (accessed: 03.08.2022).

[24] M. Google. *Mediapipe ModelCard*. URL: https://google.github.io/mediapipe/solutions/models.html#face-detection. (accessed: 03.08.2022).

[25] dlib. *dlib C++ Library*. URL: http://dlib.net. (accessed: 22.08.2022).

[26] A. Geitgey. *Face Recognition*. URL: https://github.com/ageitgey/face_recognition. (accessed: 04.08.2022).

[27] PyPi. *Face Recognition*. URL: https://pypi.org/project/face-recognition/. (accessed: 04.08.2022).

[28] A. Geitgey. *Machine Learning is Fun: A Hands-On Guide to Machine Learning, Deep Learning, AI and Natural Language Processing, 2nd Edition ebook*. 2019.

[29] Mediapipe. *MediaPipe Hands*. URL: `https://google.github.io/mediapipe/solutions/hands.html`. (accessed: 05.08.2022).

[30] Mediapipe. *MediaPipe Hands Model Card(Lite/Full)*. URL: `https://drive.google.com/file/d/1-rmIgTfuCbBPW_IFHkh3f0-U_lnGrWpg/preview`. (accessed: 05.08.2022).

[31] Google. *Google AI Blog: On-Device Real Time Hand Tracking with MediaPipe*. URL: `https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html`. (accessed: 05.08.2022).

[32] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann. *MediaPipe Hands: On-device Real-time Hand Tracking*. 2020. DOI: `10.48550/ARXIV.2006.10214`. URL: `https://arxiv.org/abs/2006.10214`.

[33] D. Cochard. *BlazeHand : A Machine Learning Model for Detecting Hand Key Points*. URL: `https://medium.com/axinc-ai/blazehand-a-machine-learning-model-for-detecting-hand-key-points-c3943b82739a`. (accessed: 05.08.2022).

[34] I. Sommerville. "Software Engineering, Tenth Edition". In: Pearson, 2016. Chap. 1,2,4.

[35] R. Pi. *Datasheets*. URL: `https://datasheets.raspberrypi.com`. (accessed: 11.08.2022).

[36] R. Pi. *Datasheets, Specifications*. URL: `https://www.raspberrypi.com/products/raspberry-pi-touch-display/`. (accessed: 11.08.2022).

[37] R. Pi. *Datasheets, Specifications*. URL: `https://www.raspberrypi.com/documentation/accessories/camera.html`. (accessed: 11.08.2022).

[38] OpenVINO. *OpenVINO Documentation: Face Recognition Models*. URL: `https://docs.openvino.ai/2021.3/face_recognition_models_public.html`. (accessed: 22.08.2022).

[39] OpenVINO. *OpenVINO Documentation: Face Recognition Models*. URL: `https://docs.openvino.ai/latest/search.html?q=face%5C%20recognition`. (accessed: 22.08.2022).

[40] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. 2015. DOI: `10.48550/ARXIV.1512.03385`. URL: `https://arxiv.org/abs/1512.03385`.

[41] D. King. *High Quality Face Recognition with Deep Metric Learning*. URL: `http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html`. (accessed: 22.08.2022).

[42] Apple. *MacBook Pro (15-inch, 2017) - Technical Specifications*. URL: `https://support.apple.com/kb/SP756?locale=en_GB`. (accessed: 26.08.2022).

[43] S. Z.
bibinitperiod S. Xiao. *3D Face Recognition: A Survey*. URL: `https://hcis-journal.springeropen.com/articles/10.1186/s13673-018-0157-2`. (accessed: 07.09.2022).