[Analysis 2](#_Toc98686463)

[Problem Definition 2](#_Toc98686464)

[My clients 2](#_Toc98686465)

[Requirements 2](#_Toc98686466)

[User requirements 2](#_Toc98686467)

[System Requirements 3](#_Toc98686468)

[Hardware Requirements 3](#_Toc98686469)

[Requirement Justification 3](#_Toc98686470)

[Interview 3](#_Toc98686471)

[Research 3](#_Toc98686472)

[Existing Solutions 3](#_Toc98686473)

[Success Criteria 4](#_Toc98686474)

[Design 4](#_Toc98686475)

[Decomposition 5](#_Toc98686476)

[User Interface 5](#_Toc98686477)

[Sign Up Screen 5](#_Toc98686478)

[Welcome Screen 6](#_Toc98686479)

[Webcam Screen 7](#_Toc98686480)

[Algorithms 7](#_Toc98686481)

[Local Binary Pattern Histograms 7](#_Toc98686482)

[Data Structures 8](#_Toc98686483)

[Test Data 8](#_Toc98686484)

[Sub Routines 9](#_Toc98686485)

[Package Management 9](#_Toc98686486)

[Uploading to cloud 9](#_Toc98686487)

[Taking Images 10](#_Toc98686488)

[Updating GUI 11](#_Toc98686489)

[Development and Testing 11](#_Toc98686490)

[Prototypes 11](#_Toc98686491)

[Prototype 1 11](#_Toc98686492)

[Prototype 2 11](#_Toc98686493)

[Prototype 3 15](#_Toc98686494)

[Prototype 4 19](#_Toc98686495)

[Testing 19](#_Toc98686496)

# Analysis

## Problem Definition

Over the years, the digital gap between humans and computers has decreased dramatically. With more programs requiring an account to be created, the need for a more secure method of keeping your accounts safe has increased. This has led to me looking into facial authentication as the new way of storing accounts. This leads to you not having to remember any of your passwords, which will remove people forgetting their passwords and increasing the security of their accounts by not having to write their passwords down on piece of paper or on a password manager. Every face is biometrically different therefore cannot be replicated by someone else in order to gain access to unauthorised information. As more people have phones and laptops, the percentage of people who have webcams/cameras increases, this leads to being able to access their faces to utilise biometrics.

## My clients

My typical users will be IT staff of companies. This is due to the data that is being accessed needing to be secure from unauthorised access. Utilising my software means that the users will be able to quickly access their accounts via a secure method (facial authentication). In the workforce, most companies typically require you to change your password once every 4 months, this can lead to people forgetting what their current password is and having to have their password written down elsewhere. If the client switches to using facial recognition, the security of their account will increase whilst also being quicker to access. This can lead to the productivity of employees increasing as the time taken to login will decrease.

## Requirements

### User requirements

* The user will be able to register their face to the database
* The user will be able to login using their account

### System Requirements

* The system will be able to identify a set of pre-defined faces
* The system will be able to predict a face.
* The system will be able to work correctly over multiple devices
* The system will be able to backup data to the cloud

### Hardware Requirements

* The system requires a webcam

### Requirement Justification

* Face will need to be able to be registered to the database to be able to be trained into the algorithm, allowing for their face to be identified.
* The user login is required by the system to allow for the user label to be added, this is needed in order to make sure anyone using the system has their face in the dataset to stop false positives.
* Having the system being able to identify a pre-defined set of faces assures me that the system is reliable on returning the same answers and can be a set to test the confidence of the program on.
* Having a certainty of >=85% allows for the system to keep the number of a false positives to a small number whilst having a relatively short training process.
* Having the functionality of being able to work over multiple devices means that the system won’t have to be individually trained on each device, therefore increasing the speed at runtime.
* Being able to backup data to the cloud allows for the program to get data that is needed in order to function from a central storage.

## Interview

## Research

### Existing Solutions

[Veritone: AIWare](https://www.veritone.com/aiware/enterprise-ai-platform/)

AI platform designed specifically for large-scale enterprise problems. Has taken part in government problems, utilising their AI engines to meet the “California Racial and Identity Profiling Act”. The software also utilises a variety of AI engines to allow for you to train custom facial recognition libraries utilising their API, or deploying their AI engines to your applications using dockers or their GraphQL API. However, Veritone utilises an annual subscription plan to utilise their AI engines, this ranges from: $72,000 for their basic plan to $180,000 for their premier plan each year. This places the software behind a paywall, meaning the software will only be accessible by the more developed companies.

[PimEyes](https://pimeyes.com/en)

Facial recognition and reverse image search software. Currently priced at $263.90/year for the cheapest plan of 25 searches a day, ranging to $2634.30/year for unlimited searches a day. The intention of the program is to find where any images of you are being used on the internet.

This application is useful for maintaining internet privacy, but with a free subscription you are only limited to being told the number of websites that contain your image, you need to upgrade to the monthly plan to show the websites that have your image.

[Blippar](https://www.blippar.com/) – Halos

Augmented reality application, designed in using facial detection in order to generate face profiles to be used within their main application. These face profiles are generated from obtaining facial features using your webcam, Halos claims to have a 99.6% accuracy, and acts much faster than other existing facial recognition engines. Also, Blippar does not require an annual fee, therefore is accessible to anyone with a smartphone.

## Success Criteria

|  |  |
| --- | --- |
| Criteria | Evidence |
| The system will be able to accurately identify faces with a degree of 85% | The degree is a measurable constant therefore can be directly checked |
| The system will be able to correctly identify a set of pre-defined faces | This can be measured by comparing the program’s answers against the labels of the original database. |

# Design

## Decomposition

Diagram

Description automatically generated

## User Interface

### Sign Up Screen

Graphical user interface, text

Description automatically generated

Whilst designing this layout, I considered the characteristics that my project requires in order to function, the user’s name to be assigned to the User ID. The user can enter their name in these simple entry fields, which have been made to remove as many areas as possible of human error, in order to allow for the algorithm to have accurate labels to train the data on. By including this screen, it allows for the program to know who the intended recipient is and fills all major fields in for the LBPH to run effectively.

### Welcome Screen

Text

Description automatically generated

The welcome screen consists of 4 major buttons, the “i” displays a screen showing the instructions on how to correctly add your face to the database. “Add Face To System” initiates the webcam allowing for you to manually add your face to the dataset, after the face has been added, the add face to system button will be removed to restrict the user from being able to add duplicate IDs of the same person to the system. The “Test facial recognition” button is used in order to test LBPH on the user in order to identify the person using the app.

### Webcam Screen

A picture containing text

Description automatically generated

The webcam screen consists of an inner constraint and the “Take Photos” button. The inner constraint is used in order to display the webcam image, whilst keeping a border on the GUI to allow for user interaction. The “Take Photos” button will initiate the algorithm to save images of the user present in the webcam. Having the “Take Photos” button as a physical button instead of an automatic process will allow for the test data to be more appropriate due to the user being prepared instead of the program taking images as soon as the webcam loads.

## Algorithms

### Local Binary Pattern Histograms

Algorithm used in order to classify textures, using 4 parameters: (Radius, Neighbours, GridX, GridY) Radius is the radius of pixels to encounter around the central pixel. Neighbours is the number of points to build the local binary pattern on. GridX is the number of cells horizontally, and GridY is the number of cells vertically.

The algorithm works by creating 3x3 grids on a greyscale image and converts them into a saturation value between 0 and 255. The central value is then compared to the surrounding squares, if the saturation of the pixel is greater than the central pixel, the pixel is assigned the value 1 otherwise it’s assigned 0.

Next, the binary values of the neighbours are concatenated together to form a binary string of length 8. This is then converted into a denary number which is set as the saturation of the central pixel. This is then repeated throughout the entire image moving the central pixel across by 1 to create a new image made from the compared pixels.

Each grid can then be converted into a histogram by forming grids using the GridX and the GridY parameters. These histograms can then be concatenated together to form a long histogram representing your image.

These histograms can then be used in order to classify the face stored with the ID label against the face present in the webcam by calculating the Euclidean distance between peaks and choosing which image is closest to the target image. This will provide a high accuracy, due to the degree of accuracy on the histogram’s Y axis.

## Data Structures

* The image for analysis should be collected as an array of row x columns for the given image in the video stream. The array should be stored as a 2D array, allowing for iteration between each row.
* The labels for comparison will be stored as a dictionary key to allow for easy lookup of the person when the histograms are compared
* The histograms will be stored as 1D arrays of the peaks of the 10 bins, this will allow for the program to keep a record of each user’s histogram to enable faster comparisons on future uses.

## Test Data

* Test data throughout the development will be images of people in different backgrounds, in order to test whether background noise effects the confidence of the algorithm.

## Sub Routines

### Package Management

Text

Description automatically generated

The code extract above checks to see if all required libraries are installed before executing the program to allow for no crashes to occur during execution due to missing packages. This works via trying to import the library, if the library does not exist in the Python/modules folder, the program will create a PowerShell script to install the library and outputs any error or the final output to windows’ inbuilt bin to keep the screen clear for the user.

### Uploading to cloud

Text

Description automatically generated

Using the pyrebase library, which utilises Google’s Firebase service, I can upload images and histograms to the cloud to be used by other machines. Having all important files on the cloud leads to the overall footprint of the program being smaller, this is better as the reading time for the algorithm is quicker if the program does not need to index through lots of files.

Text

Description automatically generated

Within the above extract, the program gets the list of folders within the local path that the outputted photos are stored. Next, I attempt to download the first image of the user’s dataset to verify if the folder already exists, if the folder exists, the image is deleted from your system and the program moves onto the next folder. However, if the folder has not been uploaded to the cloud datastore, the program will enumerate through the images in each folder and upload them to their respective directory on the cloud datastore.

### Taking Images

Text

Description automatically generated

I will be able to take effective images to be used for training, by using facial detection to identify the location of the face within the webcam. This can then allow for the photo to only be focused on a set square around the face, enhancing the features for LBPH to be trained on. By removing these excess details, I believe that the percentage accuracy of the algorithm will increase due to the noise around the face being removed, this will lead to a cleaner histogram being made when the image is processed.

### Updating GUI

Text

Description automatically generated

The program will be built using the inbuilt python library Tkinter, Tkinter will allow for me to easily create a simple graphical interface for the end user to interact with. This will help the program’s functionality as this removes the ability for users to input incorrect data into shell code. The graphical interface will also provide an easier to understand layout for the program leading to a better understanding of how to operate it.

# Development and Testing

## Prototypes

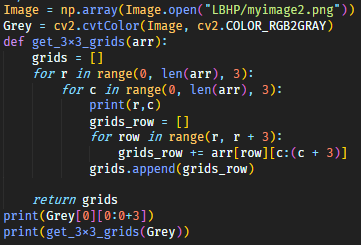
### Prototype 1

Back-end

Initial file storage was made locally on the user’s computer. This was done using relative file paths from the executable. However, by implementing it in this way, the program’s file size becomes very large quickly due to the resolution of the images being high quality and being stored locally on the user’s device. Another problem that this implementation restricts is the multi-user criteria that was defined. This is due to the algorithm having to be individually trained on each computer as the trained data is not shared between computers. This results in very slow use times and varying confidence based on the assumptions that each computer makes.

### Prototype 2

Local Binary Point Histograms



Creating a 3x3 grid

In order to carry out local binary point histograms, the image needs to be a greyscale image, this is done by using OpenCV’s convert colour function. This allows for me to convert my data taken from the webcam into a suitable format for training.

However, using this implementation led to index errors when given different sized images which didn’t directly divide into 3x3 grids, this was corrected by:

Text

Description automatically generated

This gave the program the capability to zero fill excess columns to create 3x3 grids. This was needed as the null pixels around the image led to the binary generated not being 8 bits long. This created incorrect pixel values which meant that the accuracy of the program will decrease, leading to more false positives when the face is located near the edge of the image.

Conversion of points

Text

Description automatically generated

A close up of a person

Description automatically generated with medium confidenceA black and white image of a person's face

Description automatically generated with low confidence

This led to the final image being converted into the image on the right side. This, however, was not the intended image as the resolution had decreased from 192x192 to 67x67. This was due to the image moving 3 pixels along after each 3x3 grid was made. This meant that some pixels were not being represented as the threshold pixel, therefore not being considered into the Local Binary Point algorithm. This created problems as crucial human features were having their quality reduced, this led to the histograms being less descriptive for the different test images, hence leading to a decrease in confidence of the LBP algorithm. This was then corrected via:

Text

Description automatically generatedText

Description automatically generated

Which produced the following images:

A close up of a person

Description automatically generated with medium confidenceAn aerial view of a city

Description automatically generated with medium confidence

This image had the same resolution as the initial image, therefore had correctly converted the stock image from the webcam into its LBP form, where each pixel has been considered from the adjacent pixels of the stock image with bounds of 0 to 255 on each pixel. This image can be used in order to create histograms out of, which can be used to statistically show the correct face to the person within the webcam.

Query Histogram’s array:

[0.14921682 0.01566364 0.17559769 0.07666941 0.01731245 0. 0.07749382 0.18878813 0.18219291 0.11706513]

Chart, bar chart, histogram

Description automatically generated

Histograms of some LBP arrays compared to the query image

The histograms are generated as lists of floating-point values, which correlate to groupings of pixels within the local binary pattern image. These histograms can be utilised in order to provide a concise array to allow for comparisons between different users. The histograms peaks are used in conjunction with Pythagoras in order to calculate the distance between the peaks of the query image and the peaks of the user’s image. This distance is then averaged throughout the histogram to provide the distance relative to the query image. The smallest Euclidian distance is the image that the algorithm chooses as the best fit against the query image.

### Prototype 3

GUI-Implementation

One of the important aspects of my program is the ability to be able to add a face to the system via the GUI, which was outlined in user requirement #1. This will be used in order to provide a gateway for the user to easily access the files needed for use and to have a front-end system that can be easily navigated for facial recognition. The GUI will be using the canvas feature in Tkinter to allow for me to develop images to be overlayed onto each other to create the GUI. For example, the welcome screen looks like this:

Text

Description automatically generated

Text

Description automatically generated

However, an initial bug with this design was that some backgrounds prioritised their canvas over other canvases. This led to the webcam screen not displaying properly as shown here:

Text

Description automatically generated

This was later fixed via setting the aboveThis parameter of the canvas to None when the canvas is called. This fixed the problem as the canvases are now given priority over each other when called, allowing for the elements to be brought to the front when needed and then re-written when the new screen is called.

Finally, the GUI had to be updated to allow for images to be dynamically taken from the webcam in order to calculate the histogram for the image. This is done by first using OpenCV’s “frontalface haarcascade” to calculate the region of the image where the face is located.

A screenshot of a computer

Description automatically generated with medium confidence

The image of the region where the face is located is then saved onto the user’s hard drive, to allow for the LBP algorithm to have a reference image for the calculation. The image is cropped in order to remove as much background noise from the image and to increase the computational speed of the program. By removing the background noise, the histogram will be more accurate as more of the face will be represented in the histogram, hence providing more accuracy when using the Euclidian distance.

**A screenshot of a computer

Description automatically generated**

Text

Description automatically generated

Finally, the webcam is displayed on the GUI by converting the webcam’s frame to a RGB image to be displayed within a label on the GUI. This provides a real-time update for the webcam to allow for the user to check the image and ensure correct alignment before submitting the frame for identification. The frame is utilised on a different thread due to the GUI not responding when the image is utilised on the same thread as the GUI.

The frame is then submitted to identify face to respond with the guess of the image in the frame, and if the logged in user is the same as the guessed person.

Text

Description automatically generated

Output:



However, the program still inaccurately guesses people when the histograms are very similar, this can be fixed by increasing the sample size of the number of histograms to use per person. This in turn however will increase the time taken for the faces to be identified. Currently, the program references the first 10 histograms per person in the dataset. If the size is increased to 25, then the accuracy will dramatically increase as the number of comparisons between each person in the dataset and the query image will increase.

### Prototype 4

Finally, the program will need to be optimised in order to allow for the time taken for the identification to be decreased to a reasonable level. This can be achieved via calculating the array of histograms on adding the face to the dataset, this will improve the overall computational speed due to only having to calculate each person once, instead of on every execution. This can be accomplished via moving the subfunction that creates the histograms to be called when the face is added to the dataset. This will create a delay after adding the face but the identification time will be decreased.

## Testing