# AIML-Exhibition Documentation

***Release 0.0.1***

**Lucas Broux, Francesco Anselmo**

**Aug 11, 2017**

# CONTENTS:

CHAPTER

ONE

# INTRODUCTION

An exhibition on machine learning and artificial intelligence is to take place at the Arup London offices. Discussions and experimentations have led us to develop two software applications to use as exhibits.

The first one is an animation that displays eyes which follow the track of the users as they enter the building.

The second one is a station that performs face recognition on the users, in order to provide with personalized recommendations. The database used for both these tasks is extracted from the 'Arup people' website.

In this file, we present and document the different modules implemented for both of these exhibits.

CHAPTER

TWO

# REQUIREMENTS AND INSTALLATION

All scripts are made using a python 3.5+ distribution, along with the following libraries:

- Numpy, Scipy, for numerical computations.
- Opencv, for image processing. Official site: http://opencv.org/
- Dlib, for image processing and machine learning methods. Official site: http://dlib.net/
- Panda3D, for the eye model. Official site: https://www.panda3d.org/
- Kivy, for the graphical user interface in the face recognition project. Official site: https://kivy.org/#home

CHAPTER

THREE

# STRUCTURE OF THE CODE AND IMPROVEMENTS TO IMPLEMENT

## 3.1 Structure of code

The code is implemented using the following structure:

- For the ‘Moving eyes’ application:

  ├── *peopleDetector.py* : tools for detecting people and faces in static images.

  ├── *streamProcessorEyes.py*: tools for optimized detection of people and faces in a video stream.

  ├── *positionFinder.py*: tools for conversion of array of rectangles into one single value corresponding to the position of the eyes.

  ├── *eyesModel.py*: the model of the eyes, taking in entry the position returned by *positionFinder.py*.

  └── *eyesMain.py*: the main class.

- For the ‘Face recognition’ station:

  ├── *databaseManager.py* : tools for the management of the database;

  ├── *filesToDatabase.py*: tools for creating a database based on images and descriptions of people.

  ├── *facialRecognition.py*: tools for detecting and recognizing faces, and drawing output frames.

  ├── *streamProcessor.py*: tools for optimized detection and recognition of faces in a video stream.

  ├── *facialRecognitionGui.py*: implementation of the graphical user interface.

  └── *logFileWriter.py*: tool to keep log of different actions.

## 3.2 Improvements to add

The improvements one could add to this project are:

- Defining to which extent we are allowed to use the ‘Arup people’ data; ask people for their authorization to use it for the face recognition application.
- Implementing a better eye model.
- Implementing a graphical user interface which is more in the style of the exhibition for the face recognition project.
- For each exhibit, testing in real conditions with different algorithms and different parameters to select the one that suits best our needs. In particular, the following things may be interesting for the eyes project:
  - Adding a way to smoothen the positions of the eyes by averaging it over a recent history.

  - Currently, the ‘Moving eyes’ are based on classifiers that detect faces and bodies. If the range of action desired for the eyes is not covered by this, it should be interesting to either:
    - Use the solution based on background substraction (movement detection) and adapt the tracking method to it.
    - Find or train other classifiers relevant to our needs to complement the implemented detectors (e.g. upper body detection, . . . ).
- And for the face recognition project:
  - Defining which recommendations to give to which profile.
  - Convert those recommendations into personalized lighting.
  - If the current profiles are not considered relevant, one can still change the different profiles and the corresponding reference words (or use another recommendation algorithm), and compute another database.

CHAPTER

FOUR

# EYE-TRACKING ANIMATION

We developped 4 modules for this eye-tracking exhibit.

1. *peopleDetector.py* implements algorithms capable of detecting people on a single image.
2. *streamProcessorEyes.py* implements algorithms capable of performing this same task on a video stream. For that, different methods can be used (resize entry image, only analyse a fraction of the frames, combine detectors with trackers, . . . ).
3. *positionFinder.py* converts the obtained locations into a position/angle for the eyes.
4. *eyesModel.py* produces a model of eyes able to adapt in real time to this position/angle.

## 4.1 The people detector

### 4.1.1 Methods used

The people detector class provides tools for the detection of people in a single image.

To do this, we identified two main methods:

- Using a background substractor to detect movement against a reference image.
- Using classification algorithms (such as a Support Vector Machine) on image features such as Histograms of Oriented Gradients (HOG). This requires pre-training classifiers, which can be found on the internet.

The first method was implemented using opencv, while the second one was implemented twice, the first time with the opencv tools and the second time with the Dlib library.

We heuristically tested these methods on videos of people walking. We noticed that:

- The background detection method is very quick, but not always accurate especially when the camera moves a little.
- The classification methods using opencv and dlib are slower but of the same order of speed. The opencv method makes more detections, which are not always accurate. The Dlib method makes fewer detections, which are almost always right, but it misses a more important part of people.

### 4.1.2 Documentation - *peopleDetector.py*

The goal of this program is to implement different methods to detect pedestrian in images.

We copy functions from the face_recognition api (https://github.com/ageitgey/face_recognition).

The detectors must implement the function getLocations(image)

which takes an image as argument and returns the locations of detected people in the image.

`peopleDetector.`**`drawLocations`**(*image*, *locations*, *color=(255, 0, 0)*)
Draws the found locations in the image.

**Parameters**

- **`image`** – The considered image.
- **`locations`** – The locations, as [[x1, y1], [x2, y2], [x3, y3], [x4, y4]] array.
- **`color`** – The RGB color of the resulting drawing.

**`class`** `peopleDetector.`**`peopleDetectorBackSub`**(*fgbg=<BackgroundSubtractorKNN 0000022B56AA74D0>*)
A class for the detection of people using background subtraction with opencv tools.

**`getLocations`**(*image*)
Get the locations of the detections.

**Parameters** **`image`** – The considered image.

**`class`** `peopleDetector.`**`peopleDetectorCV`**
A class for the detection of people using the opencv tool.

**`getLocations`**(*image*)
Returns the locations of the detections.

**Parameters** **`image`** – The considered image.

**`class`** `peopleDetector.`**`peopleDetectorDlib`**(*detectors=[<dlib.dlib.fhog_object_detector object>, <dlib.dlib.fhog_object_detector object>]*)
A class for the detection of people using Dlib.

**`getLocations`**(*image*, *number_of_times_to_upsample=1*)
Returns all locations for all detectors.

**Parameters** **`image`** – The considered image as numpy array.

**`getLocationsByDetector`**(*detector*, *image*, *number_of_times_to_upsample=1*)
Returns the locations of the detected objects in the image.

**Parameters**

- **`detector`** – The detector used for the detection.
- **`image`** – The considered image as numpy array.
- **`number_of_times_to_upsample`** – Used to refine detection but increases time of computation.

## 4.2 The stream processor

### 4.2.1 Methods used

The stream processor class provides tools for the detection of people in real time in a video stream.

We implemented two methods:

- The first one consists in fixing one detection method as described above and applying it successively to the frames. For faster results, one can analyse only a fraction of the frames and resize the analysed frames.

- The second one consists in combining the use of detectors and trackers to allow for an even faster and more accurate computation. The trackers are implemented in the Dlib library and the used method is described in [Danelljan, Martin, et al. “Accurate scale estimation for robust visual tracking.” Proceedings of the British Machine Vision Conference BMVC. 2014.]. All it requires is a box in the first frame, which it will follow for an arbitrary amount of time. A great advantage of this method is its speed. The developped algorithm is as follow:
  - We fix a maximal number of trackers to use simultaneously, along with a maximal number of iterations for each tracker. We also fix a ratio to define the resize factor of each analysed frame.
  - For each frame:
    - If the number of active trackers is inferior to the fixed maximal number, we analyse the frame with a detector. For each detection in this frame, and as long as possible, we initialize a new tracker.
    - We combine all the locations returned by the active trackers by applying a fast non maxima suppression algorithm to eliminate the overlapping boxes.

Experimentally, we noticed that the second method seems to realize faster and more accurate detections.

### 4.2.2 Documentation - *streamProcessorEyes.py*

The purpose of this module is to implement functions that return in real time the locations of detected people in a video stream.

We first implement a class to define a video stream. It must implement the function

getCurrentFrame()

which returns the current frame of the stream as np array.

Then we implement classes that analyse such streams. They must implement the function

getCurrentLocations()

which returns the locations of detected people in the current frame.

**class** `streamProcessorEyes.`**`streamProcessorFromDetector`**`(`*video_stream*, *detector*, *resize_factor=4*, *process_every=2*`)`

This class allows for the processig of a stream using only the given detector and applying it to every analysed frame.

**`getCurrentImageSize`**`()`

This function returns the current size of the image as [width, height].

**`getCurrentLocations`**`()`

This function returns the locations detected in the current image.

**class** `streamProcessorEyes.`**`streamProcessorWithTracker`**`(`*video_stream*, *detector*, *nb_trackers=5*, *tracking_time=100*, *resize_factor=4*, *process_every=2*`)`

For this stream processor, we use trackers (implemented in the dlib library) to improve the speed of the computations.

**`getCurrentImageSize`**`()`

This function returns the current size of the image as [width, height].

**`getCurrentLocations`**`()`

This function returns the locations detected in the current image.

**class** `streamProcessorEyes.`**`webcamStream`**(*webcam_number=0*)

This class implements the video stream of a webcam.

**`close`**()

Closes the process of the stream.

**`getCurrentFrame`**()

Returns the current frame of the stream, as np.array.

## 4.3 The position finder

### 4.3.1 Methods used

The position finder converts the locations of detections as rectangles into a single position in [-1, 1] to be interpreted by the eyes model.

The current algorithm is as follow: for each set of rectangle locations:

- We take the first rectangle in the list.
- We compute the barycenter of this rectangle on the horizontal axis.
- We return a number between -1 and 1 corresponding to the relative location of this barycenter in the image.

This algorithm may be improved for instance by:

- Averaging successing positions for a smoother movement.
- Cluster computation.
- Using trackers.
- Returning a 2D position if the eye model allow for both horizontal and vertical movement.

Ultimately, the choice to improve this algorithm must be made based on the experimental results on the exhibition site.

### 4.3.2 Documentation - *positionFinder.py*

The purpose of this module is to implement funtions that return in real time positions of detections.

For each implementation, we want the class to implement a function

getCurrentPosition()

which returns a value in [-1, 1] corresponding to the current position of the detection.

**class** `positionFinder.`**`deterministicFinder`**

This class implement a finder that periodically returns values in [-1, 1].

**`getCurrentPosition`**()

Returns the wanted position.

**class** `positionFinder.`**`positionFinderFromStreamProcessor`**(*stream_processor*)

This class implements a finder that uses in real time the results of the stream processor.

**`getCurrentPosition`**()

Returns the current position of the found detection, based on the results of the stream processor.

**Returns** A value in [-1, 1] corresponding to the location of the detection in the image. -1 correspond to a detection at the very left of the image and +1 at the very right. If nothing is detected, returns 0.

CHAPTER

FIVE

# FACE RECOGNITION STATION

We developped several modules for the face recognition station:

1. One module is dedicated to the management of the used database, and another one is dedicated to the initialization of the database.
2. One module is dedicated to the implementation of the different facial recognition functions required for the software. For this, we use the library Dlib, and we copy functions from the face_recognition api (https://github.com/ageitgey/face_recognition)
3. One module is dedicated to the processing of the video stream in real time.
4. One module is dedicated to the graphical user interface.
5. FInally, one module implements a tool for keeping log of the different user actions if wanted.

## 5.1 The database

### 5.1.1 Description of the database

The database consists in two folders and one python file, of the form:

Database/

├── Images/

├── Info/

└── File.npy

such that:

- The Images/ folder contains all the face images of the people.
- The Info/ folder contains all the descriptions of the people.
- File.npy represents a numpy array of the form [(encodings, name, link, profile)], where:
  - *encodings* is the array of encodings computed by the neural network used for the face recognition algorithm.
  - *name* is the name of the corresponding person, as a string.
  - *link* is the link to the corresponding image in the Images/ folder.
  - *profile* is the profile of the corresponding person, as a string.

We initialize the database with the folders Images/ and Info/. Once initialized, the database relies almost only on the numpy file, except if it is required to obtain the full profile picture or description of the person.

To obtain the folders Images/ and Info/, we scrapped the 'Arup people' pages using the headless browser Phantomjs alongside with the selenium package in python.

### 5.1.2 Documentation

#### Management of the database - *databaseManager.py*

The purpose of this module is to implement the database of faces and information.

We will implement 4 functions :

- Add(picture, person_name, picture_file_name) to add a person in the database.
- Remove(link, hardRemove = True) to remove a person from the database. the hardRemove parameter defines whether we remove physically the information from the disk.
- GetImage(name) to retrieve a picture of a person knowing their name.
- GetProfile(name) to retrieve the profile of a person knowing their name.

Our implementation is based on an array of the form

[(encodings, name, path_to_image, profile)]

for every stored image.

**class** `databaseManager.`**`database`**(*file_name='Database\London_database'*)

A class for the management of the database.

**`add`**(*frame*, *face_name*, *file_name*, *check_name=True*)

Attempts to update the database adding picture in path 'self.folder_name_images/face_name/file_name.jpg'. It computes and returns 2 booleans : name_already_exists and one_face_detected, and the database is updated iff (one_face_detected && !name_already_exists).

**Parameters**

- **`frame`** – The image.
- **`face_name`** – The name of the person in the image.
- **`file_name`** – The filename we want to give to the image.
- **`check_name`** – Boolean to decide whether we check if the name is already in the database.

**Returns** Two booleans, name_already_exists and one_face_detected, whose value show whether the database was successfully updated or not, and the correponding problem if not.

**`getImage`**(*name*)

Returns the image corresponding to the name.

**Parameters** **`name`** – The considered name.

**Returns** The corresponding image. Returns None if no image is found (even though it should not happen).

**`getProfile`**(*name*)

Returns the profile corresponding to the name.

**Parameters** **`name`** – The considered name.

**Returns** A string corresponding to the profile. Either precomputed profile or 'Unable to match description with profile' if name is not in the database (even though it should not happen).

**remove** (*link*, *hard_remove=True*)
Remove the corresponding link image from the database. If hard_remove, we also delete physically the image from the computer.

**Parameters**

- **link** – The link to the image we want to erase.
- **hard_remove** – Parameter to decide whether or not we physically erase the image from the computer.

## Initialization of the database - *filesToDatabase.py*

The goal of this program is to convert the data present under the directory 'images_path' with the following architecture:

```
person-1
├── image-1.jpg
├── image-2.png
...
└── image-p.png
...
person-m
├── image-1.png
├── image-2.jpg
...
└── image-q.png
```

and the data present under the directory 'London_info' with the following architecture:

```
person-1
└── info.txt
...
person-m
└── info.txt
```

The program will save the result under a file 'infos_path' containing the necessary data, i.e. an array

[(encodings, name, path_to_image, profile)]

To do that, we define here a basic recommender system. We also call for the functions of the facialRecognition module.

**class** filesToDatabase.**basicRecommender** (*folder_name_info*, *fdist='default'*, *categories='default'*)
A class for basic computation for recommendations.

We want it to implement 1 function:

computeProfile(name),

which assigns a category to the given name.

The different categories we aim at are by default:

- Business leader.
- Techical leader.
- Digital leader.
- Digital designer.
- Digital analyst.

If the algorithm does not find a category, it returns the string :

'Unable to match description with profile'.

**`computeProfile`**(*name*)

Computes in which profile to classify the person. For that, we look into its description.

**Parameters** **`name`** – The name of person.

**Returns** The string representation of the found profile.

## 5.2 The facial recognition tools

### 5.2.1 Methods used

We use the face recognition algorithm implemented in the Dlib library, which is already adapted to python with the face_recognition api (https://github.com/ageitgey/face_recognition).

The algorithm works in 4 times:

1. *Detection of all the faces in the frame*. For this, the Dlib library implements an algorithm based on Histogram of Oriented Gradients (HOG) computations, image pyramids and a linear classifier. Note that this is the same method that we used for the people detection in the 'Moving eyes' exhibit. For more information: [Histograms of Oriented Gradients for Human Detection by Navneet Dalal and Bill Triggs, CVPR 2005], [Object Detection with Discriminatively Trained Part Based Models by P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, No. 9, Sep. 2010]
2. *Processing of the faces*. The Dlib library computes landmarks of the face, which are then used to process an affine transformation of the face such that the eyes and nose are centered. For more information: [One Milisecond Face Alignment with an Ensemble of Regression Trees by Vahid Kazemi and Josephine Sullivan, CVPR 2014]
3. *Computation of encodings*. The Dlib library then uses a pre-trained neural network to compute 128 features, whose purpose are to characterize faces. The used network is a variation of the one presented in [Deep Residual Learning for Image Recognition by He, Zhang, Ren and Sun] and the training dataset contains ~3 million faces, corresponding to 7485 unique identities.
4. *Comparison with known faces*. We then compare those encodings with those precomputed in the database, using L2 norm. We return the name of the closest match if the associated distance is less than a fixed tolerance (usually 0.6).

### 5.2.2 Documentation - *facialRecognition.py*

The purpose of this module is to implement the algorithm of face recognition.

For that, we use the Dlib library. We also copy functions from the face_recognition api (https://github.com/ageitgey/face_recognition).

The face comparator must implement the function

> analyseFrame(self, frame, database)

that takes in entry a frame and a database such as the one implemented in databaseManager.py, and returns the list

> [(name, distance, location)]

corresponding to the list of the closest name, the corresponding distance and location, for each location detected in the image.

**class** `facialRecognition.`**`faceComparator`**(*tolerance=0.55*)

A class to implement useful functions regarding the detection and drawing of pictures.

**`analyseFrame`**(*frame*, *database*)

Returns the name of the person corresponding to closest face in the database, along with the corresponding distances, and the corresponding locations.

**Parameters**

- **`frame`** – The image to analyse.
- **`database`** – The database to search.

**Returns** A list [(name, distance, location)] corresponding to the identified names and distances in the image.

**`drawResult`**(*frame*, *result*, *color_box=(255, 0, 0)*, *color_text=(255, 255, 255)*)

Display the obtained results.

**Parameters**

- **`frame`** – The image to modify and display.
- **`result`** – A list [(name, distance, location)] corresponding to the identified names and distances in the image.
- **`color`** – The color we draw.

**Returns** A modified frame: we draw a box around the face, and a label with the name and distance below the face.

**`findSimilarFaces`**(*frame*, *database*, *nb_faces=3*)

Returns the closest matches of the person in the frame relatively to the database. The function requires that there is one and only one person in the database. If this is not the case, it returns an exception "Wrong number of faces". The function should return different names, even though it is possible that some people have more than one picture of their face in the database.

**Parameters**

- **`frame`** – The image to analyse.
- **`database`** – The database to search for matches.
- **`nb_faces`** – The number of results retrieved from the database.

**Returns** A list [(distance, name)] with nb_face elements, sorted by distance.

## 5.3 The stream processor

### 5.3.1 Methods used

The stream processor analyses the images in real time, and actualizes two variables:

- The current locations of the detected faces, along with the corresponding names and matching distances.
- One variable that corresponds to the name of the current person in front of the camera, with the specifications that:
  - Once the name is detected, it is not changed until no face has been detected for a fixed number of frames.
  - If there are several people in the frame for a fixed amount of time, a name can still be detected, and it will correspond to the closest match among those people.

The algorithm we use is as follow:

- We fix ratios corresponding to the fraction of frames to analyse and to the resize factor of analysed frames. We also fix a maximal number of frames to keep as close history (10 by default), and a threshold corresponding to the tolerance of the current name detection (2.5 by default, depends on the size of close history). The current name is initialized at None.
- For each analysed frame:
  - We apply face recognition to detect faces locations along with matching name and distance to match.
  - We actualize the frame history.
  - We extract the tuple (name, cumulative closeness) maximizing the cumulative closeness to their respective matches over the close history.
  - If the current name is None, and the cumulative closeness bigger than the threshold, we change it to the corresponding name.
  - Alternatively, if no face has been detected for the recent history, we set the current name back to None.

### 5.3.2 Documentation - *streamProcessor.py*

The purpose of this module is to implement functions that return in real time the name, distance, and locations of people in the video stream.

We first implement a class to define a video stream. It must implement the function

getCurrentFrame()

which returns the current frame of the stream as np array.

Then we implement classes that analyse such streams. They must implement the function

getCurrentAnalysis()

which returns the list [(name, distance, location)] corresponding to the list of the closest name, the corresponding distance and location, for each detected location in the image.

**class** `streamProcessor.`**`streamProcessor`**(*video_stream*, *face_comparator*, *nb_frames_in_history=10*, *closeness_threshold=2.5*, *resize_factor=4*, *process_every=2*)

This class implements the analysis of the stream with the following methods:

- We analyse only a fraction of the frames.
- Each analysed frame is internally resized (yet displayed with normal size).

- We average the results over a number of frames.

**drawCurrentFrame**(*database*)
Draw the current analysis on the current frame and return result as np.array.

**Returns** A tuple (clean_frame, drawn_frame) of np.arrays corresponding to the current frame.

**getCurrentAnalysis**()
Returns the current analysis, as [(name, distance, location) for each locations].

**getCurrentName**()
Returns the current identified name.

**reinitializeCurrentName**()
Reinitializes the value of the current identified name.

**class** streamProcessor.**webcamStream**(*webcam_number=0*)
This class implements the video stream of a webcam.

**close**()
Closes the process of the stream.

**getCurrentFrame**()
Returns the current frame of the stream, as np.array.

## 5.4 The graphical user interface

### 5.4.1 description

The graphical user interface is implemented using the kivy library.

### 5.4.2 Documentation - *facialRecognitionGui.py*

The purpose of this module is to implement the graphical user interface for the face recognition application.

For that, we use the kivy library.

**class** facialRecognitionGui.**FaceRecognitionGui**(***kwargs*)
This class defines the global layout for the graphical user interface.

**class** facialRecognitionGui.**MainApp**(***kwargs*)
Main class, corresponding to the application.

**_addNameToOutputFrame**()
Updates the output frame by adding the identified name. It asks if the suggested name is right and creates two buttons 'Yes' and 'No'.

**Parameters** **name** – The identified name for the person in front of the camera.

**_buttonCommandCaptureFace**()
Button command, if the user clicks on 'Capture face'. Takes a screenshot of the face and adds it to the database. If the database is successfully updated, it should ask to add more images.

**_buttonCommandGetRecommendation**()
Button command, if the user asks to have personalized recommendations. It modifies the text to display the user profile, and creates a new button to return to the beginning. Ultimately, it should trigger an action to change the lights of the exhibition and invite the user to visit the highlited exhibits.

**`_buttonCommandNameConfirmed`**()
Button command, if the user confirms the proposed name. It modifies the text to propose a recommendation based on the user profile, then modifies the commands for the two new 'Yes' 'No' buttons.

**`_cvToKivy`**(*frame*)
Converts cv2 image to texture, so that it can be displayed in the layout.

**Parameters** **`frame`** – The cv2 image to convert.

**`_reInitializeOutputFrame`**()
Reinitializes the output frame.

**`build`**()
Builder for the class. We initialize the stream processor. We then schedule the clock. We return the layout.

**`update`**(*dt*)
Updates the situation of the app.

**Parameters** **`dt`** – Time interval.

**`class`** `facialRecognitionGui.`**`OutputLayout`**(***kwargs*)
This class defines the layout for the output frame.

## 5.5 The log file writer

### 5.5.1 Documentation - *logFileWriter.py*

The purpose of this module is to implement a class allowing us to keep logs of the different actions realised.

The analysis of such files may help to interpret the reception of the exhibit.

**`class`** `logFileWriter.`**`logFile`**(*file_name='log.txt'*, *keepLog=True*)
A class to keep logs of the different actions.

**`message`**(*string*)
Append message to the file.

**Parameters** **`string`** – The considered message to log.

# PYTHON MODULE INDEX