Image Detection & Recognition on Raspberry Pi

Software Documentation – Version 1.0

Felipe Neiva Montalvão Melo
August 2016

1. Introduction

This software is a simple face detector and recogniser system developed for the **Raspberry Pi** platform. It is a real time application that detects faces over the frames captured by the camera. The user sees the camera capture on the screen and is able to ask the system to recognise or to save the detected faces to the image database. In order to detect the faces, Haar and LBP cascade classifier methods were used. The recognition process, in turn, was based on the LBP histogram model, which compares the LBP histogram of the detected face with the rest of the face images saved on the database. Naturally, this database should be created by the user by the process of saving the captured face images. This way, as the number of pictures on the database increases, the system gets more accurate on the recognition process.

Overall, the software was developed with the purpose of creating an image processing module able to fit a low power consumption device or an embedded robotic system. Despite the fact it was developed on a Raspberry Pi, code portability and interface compatibility were important aspects considered in this project. This way, it was decided to adopt the OpenCV library and the C++ as the programming language. Due to its use simplicity and wide range of tools, the OpenCV library was used to handle important image processing parts of the program. The result at the end of the project was a software with a satisfactory performance, accuracy and robustness. Furthermore, as an open source project, it also grants the possibility of being improved and implemented on other platforms, such as mobile devices and embedded systems.

This document provides a brief user guide for the first version of the application, as well as the description of its source code. In fact, it is a reference for developers who want to improve, adapt or use the software implemented in this project.

2. User Guide (on Raspberry Pi Only)

2.1. Requirements

- Platform: Raspberry PI
- Audio Output Device (earphones, speakers) connected to the raspberry pi through the 3.5 mm audio jack
- Raspberry Pi Camera
- Operating System: Raspbian

If the user wants the program to run with all of its functionalities, the following programs and applications should be installed:

- sox
- espeak
- v4l-utils
- modprobe
- opencv 3.1.0 (see opencv 3.1.0 installation tutorial file for a complete guide)
- python-dev

It is recommended to have these packages installed by the default Linux repository (apt-get).

2.2. Installation

The program itself does not require an actual installation, it simply needs to be copied to a directory chosen by the user along with its associated files and folders.

Procedure:

Extract the folders “bin” and “source” to a chosen directory. Do not delete, rename or move any of the extracted folders or files. The folder “source” contains the source code of the program, while the folder “bin” contains the actual program (executable file). In sequence there is an image of its organisation and a brief description of the present subfolders.

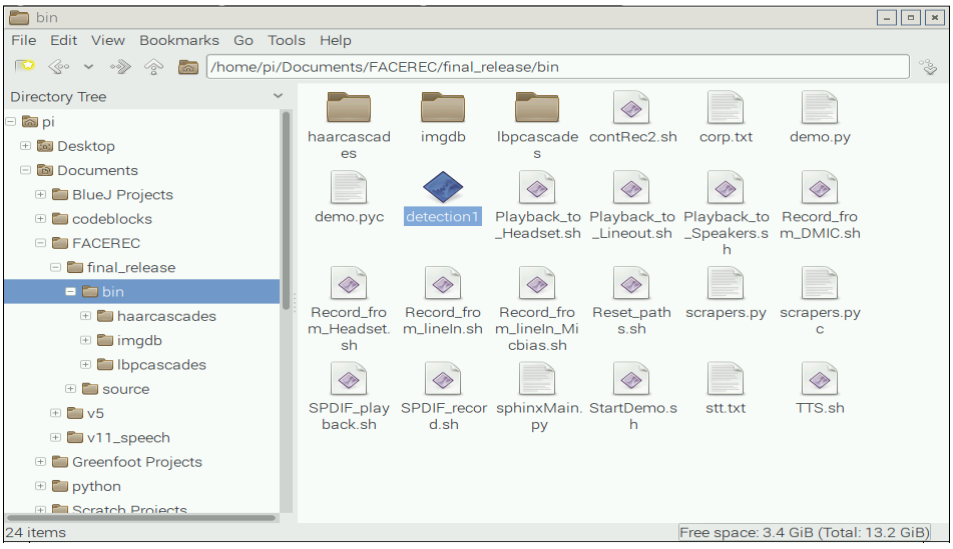

Figure 1: Program Installation Directory (“bin”)

- **imgdb:** Keeps the face images saved by the program execution
- **haarcascades:** Stores the Haar cascade classifier files (.xml) used on the detection process
- **lbpcascades:** Stores the LBP cascade classifier files (.xml) used on the detection process

Note: The file “detection1” is the binary file that runs the software. All of the other files present in the folder “bin” are the required configuration files/scripts used for the correct software functioning.

2.3. Usage

Opening Application:

Execute the file “detection1” found in the folder “bin” by double-clicking it on the file explorer or open it via terminal by calling it with no arguments.

Interface

The figure 1 shows a typical program execution screen. It consists of four windows, which are:

- **Main window**: Shows the camera capture and displays the texts that are prompted to the user. It also displays square contours (cursors) over the detected faces.
- **Console:** Displays information about the program execution such as the framerate (Average FPS) and error messages.
- **Detected Face Display:** Shows the last detected face that was captured by the camera.
- **Closest Match Face Display:** Presents the closest match to the face shown on the *Detected Face Display* after the recognition process.

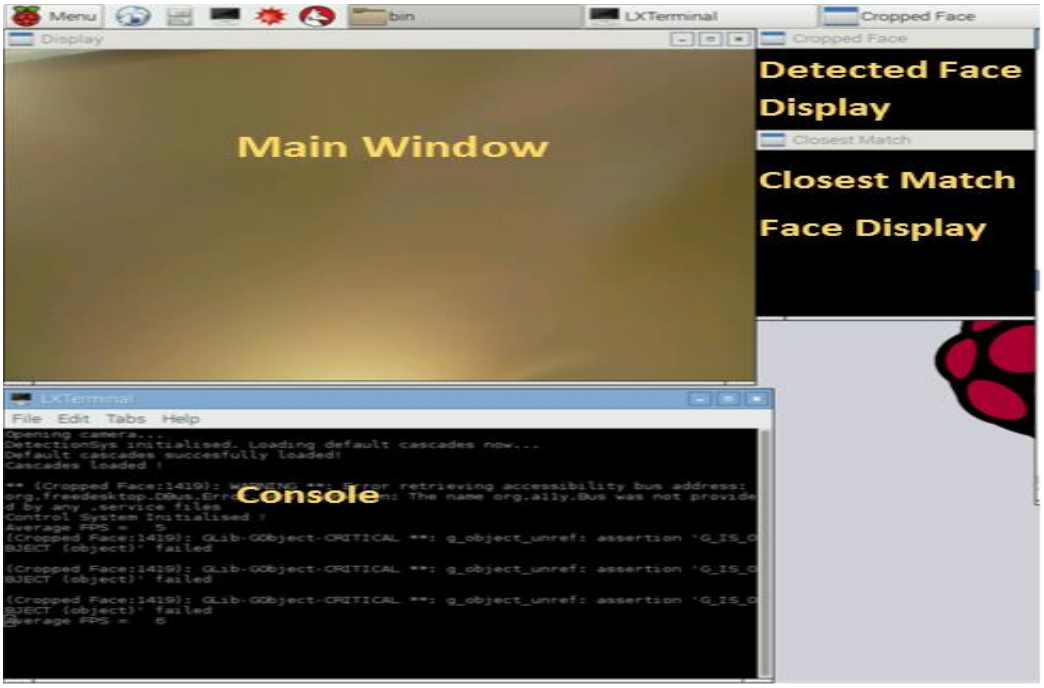

Figure 2: Application Window Arrangement

Controls:

The program is very simple to be used and interacts with the user during its execution. All of the user commands should be input by the keyboard. The table below provides the list of available commands:

Command	Key	Description
Save Detected Face	s	Save the detected face that is being displayed.
Recognise Detected Face	r	Recognise the detected face that is being displayed.
Change Focus	c	Makes the face detector try to change the focus to the next face detected on the frame. Only works during the main loop (detection)
Exit	esc	Exits or cancel interactive processes that occur during the program (works as the No). Cancel the name writing. Exits the program when executed at the main loop (detection)
Yes	y	Answer yes during interaction
No	n	Answer no during interaction
Continue	Enter	Continues program execution after some interactions

Table 1: User Command Keys

Notes:

a. In order to input the commands described on the table above, the *Main*, the *Detected Face Display*, or the *Closest Match Face Display* window should be selected (focused in). In order to do so, simply click on the window.

b. During saving process, the user needs to input the desired subject name on the screen. In order to do so, the main window need to focused and then a name can be typed normally with the use of a keyboard. The type letters are displayed on the screen. Not all characters are accepted as inputs.

c. Even though it is possible to manually save face images to the program database, it is not recommended to do so, since it may disorganise its structure. Also, a manually saved image may have a format (dimensions, colour map) that is incompatible with the program standard. It is better to save all of the images by the use of the program's own save function.

3. System Organisation and Implementation

The purpose of this part of the document is to provide a reference for developers. Just in sequence there is a description of the conceptual part of the project development and, afterwards, a code reference guide will be presented. All of the information presented in this section will help the formation of a clear idea of how the system was projected and implemented.

This Detection & Recognition system was projected to be a composition of several modules responsible for processing specific tasks coordinated by a central controller. The image below shows the diagram representing the whole system structure formed by the interconnected modules. These modular interconnections are illustrated by the black arrows in the image in sequence. They indicate both direction and type of the data that is being carried between the blocks.

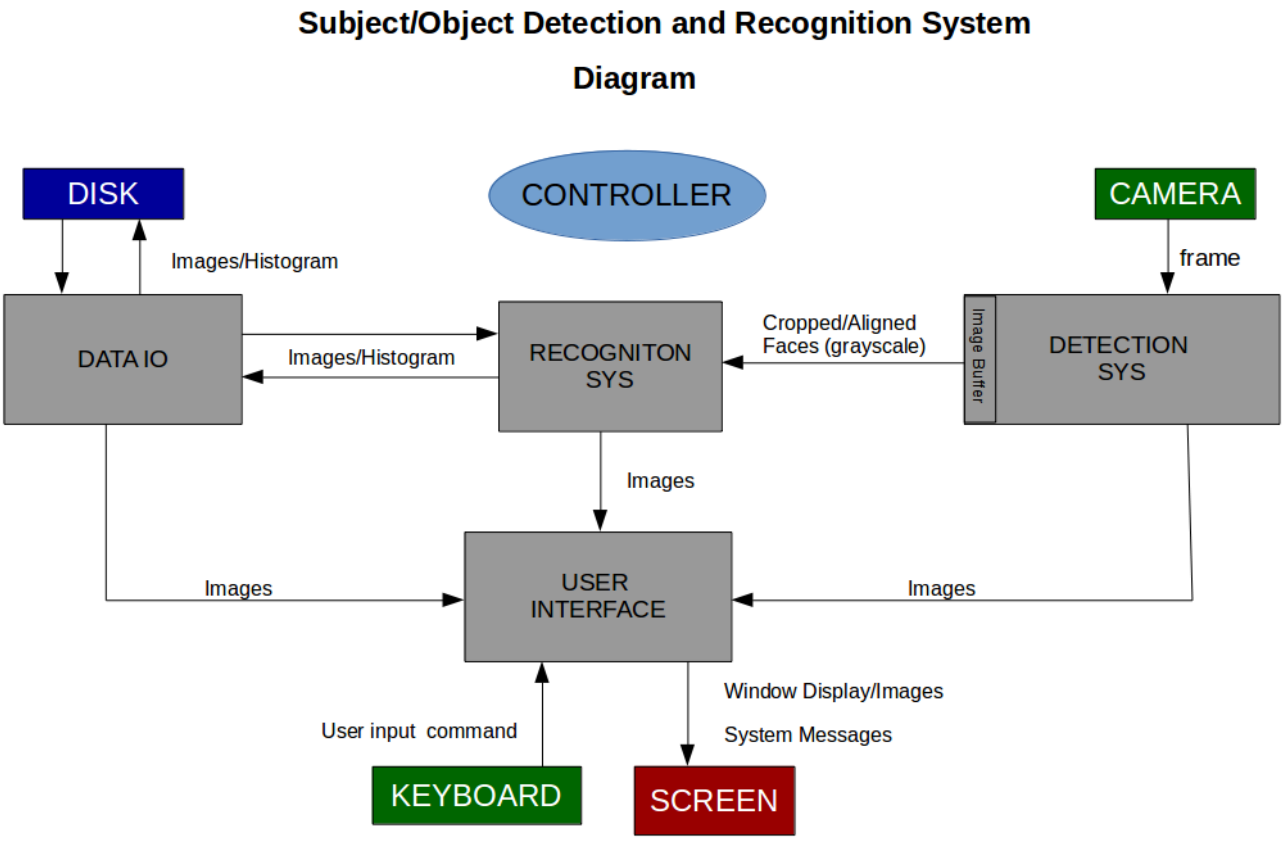

Figure 3: System Structure Diagram

As it can be seen in the image, the modules exchange only data among themselves. The control signals are, then, exclusively handled by the controller, which is connected to all of the modules. It is evident that this project design approach is very similar to a digital system's. Indeed, this architecture could be implemented on a hardware as a digital device following the exact design. However, as this project final product is an application, this project was strictly implemented by terms of software. Yet, with the use of abstraction and the power of an object oriented programming language, it was possible to easily develop the project system into a software application.

Taking advantage of the C++'s object oriented programming paradigm, it is possible to generate a software that attends satisfactorily the planned system structure and specification. Using an object oriented language, it is natural to associate the system modules to objects with their own

encapsulated processing algorithms. A module, being responsible for its own specific task, does not need to consider internally what the other ones are doing in order to process its own task. They simply have to provide an output to the control signals and data provided as input to them. As long there is an interface match and a correct control of the operations, the modules can be connected, forming a full integrated system. This idea was, hence, used to actually develop the software. In sequence, a reference of the most relevant classes and their member functions/variables will be presented, showing then the real implementation of the code.

Although most of the classes have encapsulated algorithms, there are some parts of the code that are considered more relevant for the software building and will, therefore, be described with more details. This way, developers who may want to change the code will find easier to understand and, if it is the case, modify crucial parts of the code.

3.1. Source Code

The description of the classes and their most relevant methods come here. Some classes will only have a brief description of their most relevant attributes while others will have a more detailed one. The code is divided into several header (.hpp) and source (.cpp) files. The files have the same, or at least very similar, names in relation to the classes declared/implemented on themselves. There are some classes that were written into two homonymous pair of files (.hpp and .cpp) and others that were placed into to a single file (.hpp or .cpp).

I. Definitions Header (definitions.hpp) – Basic Global Defined Values

These are the default values and definitions used in the code. Every defined value is expressed by a capital letter symbol. Default images resolution, working directories and window names are example of values defined into this file. It is recommended to change the values here than changing them directly into the code.

In this code reference, whenever a default defined valued is mentioned, it will come into capital letters. E.g.: WORKING_DIR, WIDTH, HEIGHT.

II. CameraSys – Camera setup and capture

- **`Constructor: CameraSys()`**

  Executes a terminal command in order to make the Raspberry Pi camera work as a regular USB camera.

- **`int start(const double width = 320, const double height = 240, const double fps = 12)`**

  Opens the camera and sets it up to operate with the given parameters (width, height and fps limit). Returns 0 if camera could be opened and set up. Returns 1 otherwise. The default value for width, height and fps is 320l, 240 and 12 respectively.

- `cv::Mat getFrame(double rot_angle = 0)`

  Returns a frame captured by the camera rotated by the angle given by the parameter `double rot_angle`. The accepted values of rotation angle are ±0, ±90, ±180, ±270 and ±360. Any other value passed as an argument will return a frame with no rotation.

III. DataIO – Disk Input and Output Operations. Used for saving/loading face images.

Public Variables:

```
vector<cv::Mat> m_images;
vector<int> m_numeric_labels;
vector<std::string> m_addresses;
```

Relevant Public Member Functions:

- `void loadImages(const std::string folder = WORKING_DIR)`

  Looks for the face image database on the folder given by the input argument (WORKING_DIR is the default address). All the images found are loaded by the program and allocated on the memory. The loaded images are stored on the internal object variable `vector<cv::Mat> m_images`. An ID number is given to the images according to the subject they belong to. Images which belong to the same subject have the same ID number. They are stored on the public variable `vector<int> m_numeric_labels` and occupy the same index positions of their associated images saved on the images vector. Finally, the addresses of the image files on the disk (including their names) are stored on the `vector<std::string> m_addresses`. Again, the index relative positions are the same.

- `void saveImage(const std::string& label, const cv::Mat image)`

  Saves the image given by the second argument (`const cv::Mat image`) on the image database, which is located at WORKING_DIR. The first parameter (`const std::string& label`) defines the subfolder to which the image will be saved and its name. The subject name and the file name should come in the same string divided by a comma.
  E.g.: If the string "person_a,photo1" is passed as the `const std::string& label` argument, the image file will be located at the subfolder "person_a" and it will have the name "photo1.jpg". On the other hand, if only the subject name is passed, like "person_b", the image will be saved on the subfolder "person_b" and will receive an automatic name.

- `static string name_from_address(string address)`

  This function is used to extract the name of the subject related to that image from its address on disk. The argument is the face image address and the return value is the string with the name of the subject related to the image.

IV. DetectionSys – Module responsible for detecting faces on frames, cropping and aligning them

An object of this class is the correspondence to the face detection module in the system. The detection algorithm uses the OpenCV CascadeClassifer objects and their related functions.

- **`Constructor: DetectionSys()`**

  Basically loads the LBP and Haar classifier files that are used for detecting faces.

- **`void DetectionSys::processFrame(Mat original_frame)`**

  Processes the cv::Mat frame given in the argument, storing the extracted faces on an internal object variable. The faces can then be accessed by the use of the function `vector<Mat> DetectionSys::getCAimages()`. It also draws rectangles enclosing the detected faces over the frame that was passed as the argument. The detected face that is focused on is highlighted. Currently, it only crops and aligns the highlighted face. However, the code can be easily modified if multiple faces need to be extracted at the same time. The algorithm implemented here detects the faces present in the frame with LBP classifiers. Then, the picture(s) to be extracted is passed to the function `Mat DetectionSys::getAlignedFace(Mat gray_frame, Rect2f face)` in order to be cropped/aligned.

- **`Mat DetectionSys::getAlignedFace(Mat gray_frame, Rect2f face)`**

  The first argument is the frame from which a face image is going to be extracted. The second argument is the rectangle that encloses the face to be extracted on the frame. This function returns the cropped and aligned face. It is used internally by the function `void DetectionSys::processFrame(Mat original_frame)`.
  In order to crop and align the images, the function applies eyes detection algorithm over the face region provided by the rectangle in the second argument. The eyes detection works with Haar classifiers. After finding the position of the eyes, the image can be aligned and then cropped using proportions related to the distance between the eyes. These proportions are defined in the definitions header.

V. RecognitionSys – Module responsible for recognising the face image

- **`Constructor: RecognitionSys()`**

  Sets up an OpenCV LBPH recognition model.

- **`void RecognitionSys::trainLBPH(const vector<Mat> images, const vector<int> labels)`**

  Trains the LBP histogram recogniser with the images given in the first argument labelled with the integers passed on the second argument. The LBP training data is stored within the model.

- `double RecognitionSys::findClosestMatch(const Mat& img, int& index)`

  Returns the value of confidence found on the process of recognising the face image (a low value means the tested image and the closest match are very similar). The label of the closest match is stored on the second argument.

VI. UserInterface – Module responsible for handling user I/O operations

- `Constructor: UserInterface()`

  Opens the application windows on the screen. The windows positions and identifiers are defined in the definitions header. The windows identifiers are their own names, which are passed as strings at the moment of their creation with the OpenCV function.

- `void UserInterface::displayImage(const string& window_name, Mat frame)`

  Displays the image provided by the second argument (cv:Mat frame) on the window identified by the first argument. If this window is still not open, it is automatically set up and the image is displayed normally.

- `void UserInterface::putFormatedWindowText(cv::Mat frame, const string& text)`

  Writes down a formatted text on an image being displayed on a window. The first argument is the image to have the text printed on and the second is the string containing the text.

- `const string UserInterface::win(CameraSys* cam, const string& text)`

  Pauses the program waiting for a string typed by the user on the application main window. Displays the camera capture on the main screen with an optional text printed over and the typed characters. The first argument is a pointer to a camera object (CameraSys) and the second is the optional text to be displayed. Returns the string typed after enter is pressed. If esc is pressed, returns an empty string.

- `void UserInterface::speak(string speech)`

  Uses the additional script “TTS.sh” to output an audio with a speech synthetized from the string passed in the first argument.

- `int UserInterface::getWindowKey()`

  Returns the key value that the user typed on the application window. If any key was pressed, returns -1. This is a non-blocking function. It is better used inside a while loop if the intention is to constantly check if a key was pressed.

VII. SystemStructure – Auxiliary System Structure

A simple object created to group all of the system modules. Its constructor initialises the modules. The use of this function allows the controller object to have an easier access to all of the modules of the system.

VIII. ControlSystem – Controller module of the system

- `Constructor: ControlSystem(SystemStructure* System)`

  It is initialised with a pointer to a SystemStructure object. It will make the controller have access to the whole system.

- `int ControlSystem::run()`

  Runs the system continuously. Returns 0 if the user quits the application by pressing esc. It coordinates the whole application process.

- `int ControlSystem::saveProcedure(const vector<Mat>& CAface, Mat frame, const string& name)`

  Enters inside a subroutine that will process the image saving. The first argument is a vector containing on the first position the face image to be saved, naturally already aligned/cropped. The second is a frame to be displayed and the third is the name of the subject or the subject and the file name in the format [subject,filename]. If the string of the third argument is empty, this routine will prompt the user to type a name for the subject. The return value is 1 in the case image was saved and 0 otherwise.

- `void ControlSystem::recognitionProcedure(const vector<Mat>& CAface)`

  Subroutine related to the recognition process. Asks the user if recognition was correct and if the face image recognised should be saved. The parameter of the function is a vector containing in the first position the face image to be recognised.

- `void ControlSystem::unknownFaceProcedure(const vector<Mat>& CAface)`

  Used by `void ControlSystem::recognitionProcedure(const vector<Mat>& CAface)` as an auxiliary subroutine in the case the face could not be recognised. It is triggered when the confidence level of the recognised image overpasses the THRESHOLD_REC.

- `void ControlSystem::knownFaceProcedure(const vector<Mat>& CAface, string guess)`

  Used by `void ControlSystem::recognitionProcedure(const vector<Mat>& CAface)` as an auxiliary subroutine in the case the face was recognised.

IX. FrameRate – Frame Rate Calculator (Measuring tool, not part of the system)

- `void start(bool disp = true)`

  Starts the framerate calculator. It runs on a thread which counts up to a second and then returns the number of frames counted within this one second. The framerate is displayed constantly in the output console. The frames should be count manually with the use of the function `void count_frame()`

- `void stop()`

  Stops the framerate calculator closing its associated thread.

- `void count_frame()`

  Function used to count the frames. It should be put inside a loop.

X. SpeechHandler – Attempt of implementation of the Speech Recogniser module

This part of the code is related to the Speech module that was developed at the end of the project. It was actually just a test made and therefore it does not implement a stable code. Still, it is presented here as a reference for possible future modifications and use. It actually provides an interface of communication with the real speech recogniser system, which is run by python scripts on a background application. Please see sections **4.1** and **5.2** for more details.

- `const string SpeechHandler::getText()`

  Sends a request to the speech recogniser program and checks if it responds. The request type of this function is a string, which is returned by the function in the case it is received. If the text is not received, an empty string is returned instead.

- `const int SpeechHandler::getCommand()`

  Sends a request to the speech recogniser program and checks if it responds. It indicates the speech module that the type of the requested value is an integer number. This received number is a code that corresponds to an action the program should take. If the number is not received, the value of -1 is returned instead.

- `const int SpeechHandler::getYesNo()`

  Sends a request to the speech recogniser program and checks if it responds. It indicates the speech module that the type of the requested value is Boolean (yes or no). This typo is represented by integer values, which is returned by the function in the case it is received. If the number is not received, the value of -1 is returned instead.

3.2. Additional Code: Configuration Scripts

The configuration files and scripts present in the folder "bin" were written by another developer (please consult the files for more information). They are related to a Speech Recognition module that was experimentally integrated with the system at the end of the project. Even though the application right now does not have a speech module, it uses the script file "TTS.sh" to output synthetized speech. At the moment it is the only file among these additional ones that is being actually used. The rest of them can still be useful for future implementations of the speech recognition module. For more details about the integration attempt that was made, please see section **4.1**. On the appendix, section **5.2**, there is a list of the additional code files along with a brief indication of their usage.

4. Limitations and Possible Improvements

4.1. Speech Detection System Integration

At the end of the development of the program, an attempt of integration with a speech processing unit was made as an addition to the original project plan. The additional system would allow the user to interact with the application by speaking. Besides voice commands, another few functionalities would be added and the application would become not only a face recogniser but also a virtual assistant which the user could talk to.

The speech part was basically developed on a Linux environment and used shell and python scripts. In order to integrate the two systems, adaptions had to be done on both programs so they could communicate with each other. The simplest way found to interconnect the scripts (speech recognition module) and the compiled program (image processing module) was to establish a communication protocol over text files.

Two text files were used to implement this protocol. If one of the programs wanted to send a message to the other, it would write down the message to the files. If it wanted to receive a message from the other part, it would read the files. Apparently simple, the actual communication system was quite hard to implement due to synchronisation issues. Even though, it ended up being implemented. However, it presented a quite unstable behaviour which caused a lot of execution problems. For that reason, it was decided that, for a stable release, the application would not include the speech recognition feature.

Nevertheless, the source code is provided and so the program is still open for modifications, allowing the speech to be implemented properly.

If the developer wants to modify the code and add the speech recognition functionality, there are two suggested ways:

a) Following the architecture standard of this project and, thus, implementing the speech recognition part as another module of the whole system. In this case, the user should create and add classes/functions related to the speech recognition module to the original C++ code. Both speech and image processing units would now be parts of the same system, which would run as a single compiled program. The communication between the two modules would be much easier to implement,

however, the developer would have to create a whole new C++ code related to the speech recognition.

b) Keeping the speech and face recognition as two separate programs that communicate with each other. As the scripts of the speech system were already written, the module would not need to be implemented again. A robust and efficient communication protocol would have to be defined though. Adaptions would have to be done to both codes (image processing and speech part).

4.2. Threaded Execution

Apart from the Frame Rate calculator, which runs concurrently with the other program routines, the whole system runs into a single stream. This causes the program to freeze sometimes and also reduces the performance of the system (framerate). This way, the system would perform better if some of its routines were parallelised. For example, the image capture and processing could run concurrently with the user interface operations. The use of a multi-threaded execution would require expressive modifications to the original code but would certainly generate a much smoother user experience.

4.3. Frame selection/skipping

Currently the application was programmed to execute the detection process over every single frame captured by the camera. This causes the frame rate to drop hugely since the detection algorithm consumes a large processing power. If only some frames would be picked and processed, instead of every single captured one, the overall performance of the system would be improved. Depending on the framerate increase, the resolution of the captured frames could also be enlarged.

4.4. OpenCV Functions Parameters Calibration

The face detection and recognition OpenCV functions used by the application have some adjustment parameters. These parameters, passed as function arguments, modify the performance/accuracy of the algorithms. If they were set up optimally (manual or automatically), the face detection and recognition system would respond faster and with more accuracy. Changes on lighting conditions affect the system performance, especially the detection algorithm.

5. Appendix

5.1. List of Source Code files

- **Camera – Basic Camera Configuration and Frame Capture**
  1. CameraSys.hpp

- **Controller**
  2. ControlSystem.hpp
  3. ControlSystem.cpp

- **Disk Input/Output -**
  4. DataIO.hpp
  5. DataIO.cpp

- **Detection – Face Detection/Extraction Module**
  6. DetectionSys.hpp
  7. DetectionSys.cpp

- **Recognition - Face Recognition Module**
  8. RecognitionSys.hpp
  9. RecognitionSys.cpp

- **User Interface – User Interface I/O Operations**
  10. UserInterface.hpp
  11. UserInterface.cpp

- **Main**
  12. d1_main.cpp

- **Auxiliary**

  - **Common Used Headers**
    13. common_headers.hpp

  - **Program Default Parameters and Definitions**
    14. definitions.hpp

  - **Framerate Calculator Module**
    15. FrameRate.hpp

  - **Geometric Tools Used by the Face Detection Module**
    16. GeometricAux.hpp
    17. GeometricAux.cpp

  - **String tokenizer**
    18. Tokenizer.cpp
    19. Tokenizer.hpp

- **Overall System Structure**
  20. SystemStructure.hpp

21. SystemStructure.cpp

- **Prototype Module used to communicate with the speech recogniser process (in development)**
  22. SpeechHandler.cpp
  23. SpeechHandler.hpp

5.2. List of Configuration Files/Scripts

1. **contRec2.sh –** Record audio from configured device continuously
2. **corp.txt –** Speech Recogniser Dictionary
3. **demo.py –** Speech Program demonstration (python script)
4. **demo.pyc -** Speech Program demonstration (python code)
5. **Playback_to_Headset.sh -** Configuration file. Execute this shell script to make the Headset the playback device
6. **Playback_to_Lineout.sh -** Configuration file. Execute this shell script to make the Lineout the playback device
7. **Playback_to_Speakers.sh -** Configuration file. Execute this shell script to make the Speakers the playback device
8. **Record_from_DMIC.sh -** Configuration file. Execute this shell script to make the DMIC the record device
9. **Record_from_Headset.sh -** Configuration file. Execute this shell script to make the Headset the record device
10. **Record_from_lineIn.sh -** Configuration file. Execute this shell script to make the LineIn the record device
11. **Record_from_lineIn_Micbias.sh -** Configuration file. Execute this shell script to make the LineIn and the Mic the record device
12. **Reset_paths.sh –** Restore default audio configuration (playback/record devices)
13. **scrapers.py –** Speech recognition online services (python script)
14. **scrapers.pyc -** Speech recognition online services (python code)
15. **SPDIF_playback.sh** - Configuration file. Execute this shell script to make the SPDIF the playback device
16. **SPDIF_record.sh** – Configuration file. Execute this shell script to make the SPDIF the record device
17. **sphinxMain.py** – The actual speech recognition program (python script)

18. **StartDemo.sh** – Shell script that starts the execution of the speech module in background and the Face Detection and Recognition program

19. **stt.txt** – List of codes related to extra operations of the speech module

20. **TTS.sh** – Shell script used for outputting synthetized speech. It is called from inside the Face Detection and Recognition program to execute the audio reproduction. This file should be in the same folder of the Face Detection and Recognition program executable file.