

BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER
SCIENCE

SPECIALIZATION Computer Science in English

DIPLOMA THESIS

AI-Driven Hand Tracking Application for Real-Time Drawing

Supervisor
[Grad, titlu și Tudor-Dan Mihoc]

Author
Sali Arnold

2024

ABSTRACT

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 2 | Related Work | 2 |
| 2.1 | Image segmentation algorithms | 3 |
| 2.1.1 | Hough Transform | 3 |
| 2.1.2 | Viola-Jones Face Detection | 3 |
| 2.2 | Deep learning algorithms | 4 |
| 2.2.1 | YOLO | 4 |
| 2.2.2 | RESNET | 5 |
| 2.2.3 | MOBILENET | 5 |
| 3 | Approaches for better performance with machine learning models | 7 |
| 3.1 | Model comparisons | 7 |
| 3.1.1 | YOLOv3 performance | 8 |
| 3.1.2 | MobileNet performance | 8 |
| 3.2 | Transfer learning | 8 |
| 4 | Machine learning model specifications | 10 |
| 4.1 | Modified MobileNet | 10 |
| 4.1.1 | Transfer learning specifications | 10 |
| 4.1.2 | Model output | 10 |
| 4.2 | Data | 10 |
| 4.2.1 | Data specification | 10 |
| 4.2.2 | Data preprocessing | 10 |
| 5 | Application requirements and specifications | 11 |
| 5.1 | Application requirements | 11 |
| 5.1.1 | Functional requirements | 11 |
| 5.1.2 | Functional requirements descriptions | 13 |
| 5.1.3 | Non-functional requirements | 20 |
| 5.1.4 | System requirements | 20 |

| | | |
|-----------|---|-----------|
| 5.2 | Technical specifications | 21 |
| 6 | Application design and implementation | 22 |
| 6.1 | Graphical User Interface | 24 |
| 6.2 | Image processing | 27 |
| 6.3 | Functionality implementation | 27 |
| 6.3.1 | F1 implementation | 27 |
| 7 | Application testing | 30 |
| 7.1 | Testing techniques | 30 |
| 7.2 | Use case testing specifics | 30 |
| 8 | User manual | 31 |
| 8.1 | Starting the application | 31 |
| 8.2 | Starting the recording | 31 |
| 9 | Future Work | 33 |
| 9.1 | Machine Learning Model Improvements | 33 |
| 9.2 | GPU Utilization | 33 |
| 10 | Conclusions | 34 |
| | Bibliography | 35 |

Chapter 1

Introduction

Chapter 2

Related Work

The field of computer vision has evolved from natural vision. Its main purpose is to allow machines to understand visual information based on the natural way humans and other animals gain information from visual input.

This branch of computer science is widely used to gain insight into the real world, through different algorithms and computational techniques, ranging from simpler problems such as object detection in an image or video to more complex ones such as scene understanding and image generation. The solutions from this field open the door for new innovations, which are already present in some capacity in today's society such as a simple social media filter or self-driving cars.

The evolution of computer vision can easily be followed along with the evolution of computational power, given the high requirements of image processing. In the earlier days, such as the 1960s and 1970s, the first algorithms were very limited by the processing power available, but as time passed, more sophisticated ones were created, such as Hough Transform, the Viola-Jones face detection and machine learning.

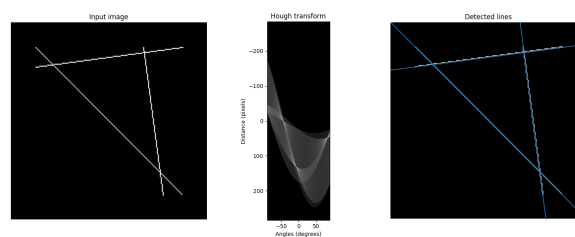


Figure 2.1: Hough transform calculation to find the lines of a triangle [Unk]

2.1 Image segmentation algorithms

2.1.1 Hough Transform

This technique was proposed by Paul Hough in 1962, as a method to identify patterns in images. [DH72]

The Hough Transform is used in computer vision and image processing. The core idea behind it is a voting process made by the curves in the transform space, creating cluster points or local maxima. These points represent the existence of a shape, and since the curves contain the parameters of the shapes, from those parameters the position of the shape in the input image can be detected, like it is shown in Figure 2.1.

The main strength of this method is that it is capable of identifying shapes even in slightly obscured or noisy data. This made this procedure a massive breakthrough in image processing. Since it's powerful to be able to detect shapes, the method is used in many sectors, from robotic navigation, by identifying edges and lines in the environment, to medical imaging, by detecting circular shapes potentially depicting different biological conditions and many more, where image processing can be utilized.

2.1.2 Viola-Jones Face Detection

Another breakthrough in computer vision was the face detection procedure proposed by Paul Viola and Michael Jones in 2001. [VJ01]

The main idea behind their approach lies in the usage of Haar-like feature and Adaptive Boosting.

The Haar-like features are simple rectangular regions in the input image data. The main goal of using these patterns is first to cut down on the necessary calculations by working with pixels, and second is to get a better understanding about certain regions in the image. By calculating the differences in the sum of pixel intensities between the regions, a contrast can be learned, an incredibly useful information which allows the detection of basic patterns, like edges, lines and textures.

Adaptive boosting is a machine learning algorithm used for boosting weak classifiers into a strong one. The main principle of this method is to iteratively train weak classifier, each focusing on different features. At each iteration pass, the algorithm selects the best performing classifier and merges it with the final strong one, weight proportionally to its performance. It also weighs incorrectly classified instances more for the next iteration for the purpose of shifting the focus on the most challenging parts.

With the help of these two procedures the method of Viola and Jones achieves

great accuracy in real-time, more specifically at 15 frames per second on a 700 MHZ Intel Pentium III, using approximately 50 thousand parameters. Since its low hardware requirements, this way of face detection is still used on very weak devices.

2.2 Deep learning algorithms

With the increase in computational power, image processing algorithms have also evolved in tandem. As seen in the Viola-Jones face detection algorithm, machine learning were already used in the early 2000s, be it at a smaller scale.

Since these algorithms now are able to utilize more resources, they can work with way more parameters, going from thousands to millions, allowing them to learn more complex features than before.

My choice for the following three deep learning architectures is that, they were significant achievements in the field of computer vision, while also resembling my chosen base model the MobilNet.

2.2.1 YOLO

YOLO (You only live once) is an object detection method, based on a Convolutional Neural Network backbone, the building blocks of which can be seen in Table 2.1, containing convolutional layers of varying sizes.

The main selling point of this algorithm, is that it achieves real-time performance with a single-pass approach. It divides the image into a grid, each predicting a bounding box, with an associated confidence score and class probability. Utilizing these parameters and various anchor boxes, which are bounding boxes with a pre-defined shape, size and aspect ratio, the model predicts the final bounding box for the searched cell. [RF18]

Since its real-time performance the method has been utilized in many projects, from autonomous driving, to surveillance systems.

| Type | Filters | Size | Output |
|---------------|-----------|------------------|-------------------------------|
| Convolutional | 64 – 1024 | $3 \times 3 / 2$ | $128 \times 128 - 8 \times 8$ |
| Convolutional | 32 – 512 | 1×1 | |
| Convolutional | 64 – 1024 | 3×3 | |
| Residual | | | $128 \times 128 - 8 \times 8$ |

Table 2.1: Building blocks for the YoloV3 CNN backbone, Darknet-53 [RF18]

2.2.2 RESNET

ResNet is a convolutional neural network proposed by Kaiming He and co. in 2015, achieving great performances in various computer vision tasks.

The main strength of this model is that it addressed the vanishing gradient problem, via the introduction of skip connections. The residual blocks, the main building blocks of this model, contain the shortcut connections, skipping one or more layers. With this, the network is able to learn residual functions, basically the difference between the desired output and the input data. This feature allows the ResNet architecture to increase the depth of the model, without the gradient vanishing. [HZRS15]

As can be seen in Table 2.2, the complexity of this model can grow to incredible levels, containing as much as 19.7 million parameters, meaning the number of calculation for one single pass through of this architecture is around that number.

| Number of layers | Number of parameters |
|------------------|----------------------|
| 20 | 0.27M |
| 32 | 0.46M |
| 44 | 0.66M |
| 56 | 0.85M |
| 110 | 1.7M |
| 1202 | 19.7M |

Table 2.2: Correlation between the size of the model and the number of parameters used [HZRS15]

2.2.3 MOBILENET

MobileNet is a convolutional neural network designed specifically for mobile and other devices with limited computational power.

| Model | ImageNet Accuracy | Parameters |
|----------------|-------------------|------------|
| Conv MobileNet | 71.7% | 29.3M |
| MobileNet | 70.6% | 4.2M |

Table 2.3: Difference in parameters between Depthwise separable and full convolutional MobileNet architecture [HZC⁺17]

At its core the architecture is built using depthwise separable convolutions. The standard convolutions is separated into two separated operations. The first one being a depthwise convolution, in which a single convolution filter is applied for each input channel, capturing features separatly. The second is pointwise operations,

which applies a 1×1 filter to the outputs of the the previous calculations, combining the results. This allows the model to significantly reduce the number of parameters needed, compared to normal convolution, thus decreasing the computational power needed. [HZC⁺17]

In Table 3.1 the incredible difference between the complexity of the two architectures. For a 1.1 percent loss the exponential decrease in parameter count is a fantastic exchange.

Chapter 3

Approaches for better performance with machine learning models

As the learning capabilities of deep neural networks increased over the years, so have their sizes in terms of the number of parameters. With this the resources needed to make the algorithms learn and to utilize them have also increased, to the point where utilizing them in real-time applications have become somewhat challenging on an everyday computer.

To be able to utilize a deep neural network algorithm in real-time, an architecture has to be chosen, which minimizes the number of parameters, hence also minimizing the number of computations for each pass through the network, while also maintaining a useful learning capacity.

To be able to compare the running time of some models, I am going to use the running time of one pass through the architecture in milliseconds. The goal is to use them in a real-time video processing application, with a capped frame rate of 24 frames per second, would give a running time needed of around 41 milliseconds. Achieving this with a model complex enough to somewhat accurately identify hand positions and gesture, might prove difficult, since most architectures have the number of parameters in the millions. To combat this, only every second image will be processed, creating a more comfortable window.

3.1 Model comparisons

Starting with a simpler architecture in terms of the deep learning neural network, in the study of John and co. [JBM⁺16] a high accuracy was achieved in real-time using representative frames, hence selecting better data for the machine learning model to predict from. Their algorithm clearly separated the two parts, the frame extraction running in around 70 milliseconds and the classification running in around 40

milliseconds, achieving an accuracy of 91 percent.

3.1.1 YOLOv3 performance

The YOLOv3 model is an efficient deep learning architecture, making it more than useful in achieving real-time performance on hand gesture detection and identification problems. In the approach of Mujahid, Awan and co. [MAY⁺21], this algorithm was thought from scratch on the Mindst dataset [Den12], achieving good results. They proposed a lightweight architecture which is built on the YOLOv3 model, achieving impressive results, with a approximate accuracy rating of 98 percent in real-time, although time specifications were not provided.

3.1.2 MobileNet performance

The approach of Wanga, Hua and Jina [WHJ21] consists of utilizing the architecture of MobileNet and Random Forest to identify hand gestures. They utilized the pre-trained parameters of MobileNet on the ImageNet dataset [DDS⁺09], then taking the output and running it through a Random Forest model to better extract features from the images. Their paper does not focus on performance in the context of the time needed for a prediction, that being said, since they use the MobileNet architecture as their backbone, which in a configuration of around 3.5 million parameters can achieve a pass-through time of around 35 milliseconds, it is safe to assume, that near real-time performance is achievable with their approach.

In Table 3.1 the comparison of the two models can be seen with regard to their accuracy ratings on three different hand based image datasets.

| Model | SLD Dataset Accuracy | SLGI Dataset Accuracy | Fingers Dataset Accuracy |
|--------------|----------------------|-----------------------|--------------------------|
| MobileNet | 74.25% | 94.12% | 97.02% |
| MobileNet-RF | 80.97% | 95.12% | 99.72% |

Table 3.1: MobileNet and MobileNet-RF accuracy, from the study of Wanga and co. [WHJ21], on the Sign Language Digital Dataset (SLD) [KSH22], Sign Language Gestures Image Dataset (SLGI) [VMGMST⁺23] and the Fingers Dataset

3.2 Transfer learning

Transfer learning is a machine learning technique where a pretrained model on a specific task and dataset is repurposed and fine-tuned for a different but similar problem. The pretrained values are taken for a new task, since the learned functions

are usable in the new context, like in the example of identifying animals and human faces, the pretrained values will help to extract certain features which characterizes these objects, like facial line structures.

Using this method the deep neural network will start in closer to optimal position when learning features, jump starting the process, and enhancing the overall accuracy of the model. It is also a very useful tool, when dealing with a smaller dataset, or when collecting more data proves to be more challenging, allowing the architecture to learn with less data. This is also the reason why this approach is used in this application, since collecting thousands of images and labeling every one of them would take up an incredible amount of time.

Chapter 4

Machine learning model specifications

4.1 Modified MobileNet

4.1.1 Transfer learning specifications

4.1.2 Model output

4.2 Data

4.2.1 Data specification

4.2.2 Data preprocessing

Chapter 5

Application requirements and specifications

The main goal of the application is to create an alternative way of interacting with a live video. For this purpose a deep learning neural network is used, more precisely a MobileNet model pretrained on the ImageNet dataset, with custom final layers for identification. The machine learning algorithm will identify hand gestures, acting as the controlling tools for the application.

5.1 Application requirements

The main focus of the application is the alternative way of interacting with the live video feed, providing feature for drawing with a finger, using simple hand gestures for clearing the drawings, for zooming in on a specific section of the video, for changing the volume of the recording and for taking a screenshot.

Besides the hand driven way of interactions the volume settings and the screenshot features all have a specific buttons and toggles, so they can be worked with normally with normal mouse actions.

The application can also be used as a simple video recording software, providing settings options for camera and microphone selection, changing the folder to which save the video files and screenshots.

All of the above is in visual format in the use case diagram below 5.1, which was created by an online tool called Lucidchart [Luc].

5.1.1 Functional requirements

To see the functionalities of the applications in an easier way, all of the above mentioned features are represented and described as use cases. The list with them is presented in Table 5.1, while the descriptions will be in the ones after.

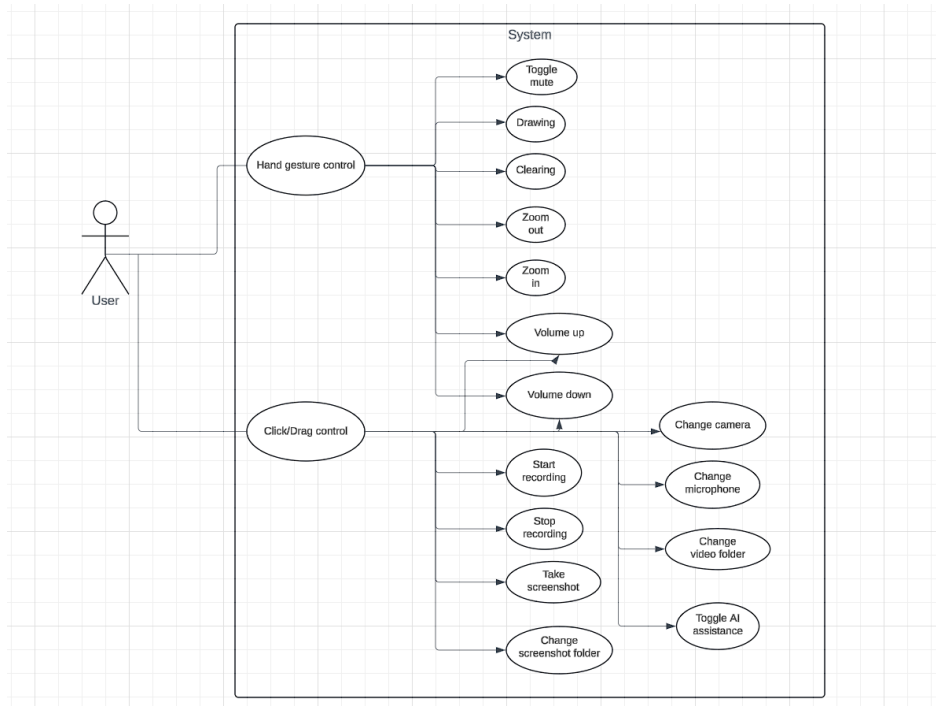


Figure 5.1: Use case diagram for the functionalities of the application

| Use Case | Name |
|----------|---|
| F1 | Start recording |
| F2 | Stop recording |
| F3 | Take screenshot |
| F4 | Change volume up |
| F5 | Change volume down |
| F6 | Change video folder path |
| F7 | Change screenshot folder path |
| F8 | Change camera used during recording |
| F9 | Change microphone used during recording |
| F10 | Toggle artificial intelligence assistance |
| F11 | Draw during recording |
| F12 | Clear screen after drawing |
| F13 | Toggle being muted |
| F14 | Zoom in a portion of the video |
| F15 | Zoom out |

Table 5.1: The application's functionalities represented as use cases

5.1.2 Functional requirements descriptions

The F1 requirement is responsible for starting the video capture process. The steps and requirements for this functionality can be found in the table below 5.2.

Prerequisites: The user chose a functioning camera and microphone; The user chose a path for the video to be saved in.

Post: The live feed of the video recording shows up on the screen; A pop-up error message is shown that the start was not successful.

| User | System |
|--------------------------------------|---|
| 1. The user presses the Start button | |
| | 2. The system checks for a functioning camera and microphone |
| | 3. The system checks if the folder path given exists or not |
| | 4 The system starts the video capture |
| | 4.1 The system presents the live feed on the screen |
| | 4.2 The system shows a pop-up error message about the failure |

Table 5.2: F1 Functionality steps

The F2 requirement is responsible for stopping the video capture process. The steps and requirements for this functionality can be found in the table below 5.3.

Prerequisites: The video recording is running successfully.

Post: The live feed on the screen will stop and turn into a black image and the video is present in the chosen folder; A pop-up message is shown about the failure of stopping the capturing.

The F3 requirement is responsible for taking a screenshot from the live feed. The steps and requirements for this functionality can be found in the table below 5.4.

Prerequisites: The video recording is running successfully; A valid path is chosen as the screenshot folder.

Post: A pop-up message shows the screenshot was successfully taken and the picture will be present in the given folder; A pop-up error message is shown.

The F4 requirement is responsible for changing the volume up. The steps and requirements for this functionality can be found in the table below 5.5.

Prerequisites: The volume is not at 100 percent; For hand gesture control the recording is running and the AI assistance is turned on.

Post: The volume of the audio will be changed in the current or next recording.

| User | System |
|-------------------------------------|---|
| 1. The user presses the Stop button | |
| | 2. The system stops the recording process |
| | 3. The system creates the final video file in the selected folder |
| | 3.1 A black image is shown instead of the live feed |
| | 3.2 The system shows a pop-up error message about the failure |

Table 5.3: F2 Functionality steps

| User | System |
|---|--|
| 1. The user presses the Screenshot button | |
| | 2. The system saves the last frame as an image in the chosen folder |
| | 3.1 A pop-up message is shown that the screenshot was taken successfully |
| | 3.2 The system shows a pop-up error message about the failure |

Table 5.4: F3 Functionality steps

| User | System |
|---|---|
| 1.1 The user moves the volume slider to the right | |
| 1.2 The user show a specific hand gesture | |
| | 2. The system saves the volume modifier accordingly |

Table 5.5: F4 Functionality steps

The F5 requirement is responsible for changing the volume down. The steps and requirements for this functionality can be found in the table below 5.6.

Prerequisites: The volume is not at 0 percent;For hand gesture control the recording is running and the AI assistance is turned on.

Post: The volume of the audio will be changed in the current or next recording.

| User | System |
|--|---|
| 1.1 The user moves the volume slider to the left | |
| 1.2 The user show a specific hand gesture | |
| | 2. The system saves the volume modifier accordingly |

Table 5.6: F5 Functionality steps

The F6 requirement is responsible for changing the folder path for the video files. The steps and requirements for this functionality can be found in the table below 5.7.

Prerequisites: The video recording is not running.

Post: The folder path for the video files will be changed accordingly and the user is sent back to the main window.

| User | System |
|---|--|
| 1. The user presses the Settings button | |
| | 2. The current window is changed to the Settings window |
| 3. The user presses the "Choose Path" button in the "Video folder path" section | |
| | 4. The windows File explorer opens |
| 5. The user chooses a folder | |
| | 6. The system changes the path to the chosen one without saving it |
| 7. The user presses the Save button | |
| | 8. The system saves the setting changes to the configuration file |
| | 9. The system switches back to the main window |

Table 5.7: F6 Functionality steps

The F7 requirement is responsible for changing the folder path for the screen-

shots. The steps and requirements for this functionality can be found in the table below 5.8.

Prerequisites: The video recording is not running.

Post: The folder path for the screenshots will be changed accordingly and the user is sent back to the main window.

| User | System |
|--|--|
| 1. The user presses the Settings button | |
| | 2. The current window is changed to the Settings window |
| 3. The user presses the "Choose Path" button in the "Screenshot folder path" section | |
| | 4. The windows File explorer opens |
| 5. The user chooses a folder | |
| | 6. The system changes the path to the chosen one without saving it |
| 7. The user presses the Save button | |
| | 8. The system saves the setting changes to the configuration file |
| | 9. The system switches back to the main window |

Table 5.8: F7 Functionality steps

The F8 requirement is responsible for changing the used camera. The steps and requirements for this functionality can be found in the table below 5.9.

Prerequisites: The video recording is not running.

Post: The chosen camera to be in use will be changed and the user is sent back to the main window.

The F9 requirement is responsible for changing the used microphone. The steps and requirements for this functionality can be found in the table below 5.10.

Prerequisites: The video recording is not running.

Post: The chosen microphone to be in use will be changed and the user is sent back to the main window.

The F10 requirement is responsible for toggling the AI assistance during video recording. The steps and requirements for this functionality can be found in the table below 5.11.

Prerequisites: The video recording is not running.

| User | System |
|--|--|
| 1. The user presses the Settings button | |
| | 2. The current window is changed to the Settings window |
| 3. The user presses the drop-down list in the Camera section | |
| | 4. The systems shows a list of available cameras |
| 5. The user chooses a camera from the list | |
| | 6. The system changes the camera to the chosen one without saving it |
| 7. The user presses the Save button | |
| | 8. The system saves the setting changes to the configuration file |
| | 9. The system switches back to the main window |

Table 5.9: F8 Functionality steps

| User | System |
|--|--|
| 1. The user presses the Settings button | |
| | 2. The current window is changed to the Settings window |
| 3. The user presses the drop-down list in the Microphone section | |
| | 4. The systems shows a list of available microphones |
| 5. The user chooses a microphone from the list | |
| | 6. The system changes the microphone to the chosen one without saving it |
| 7. The user presses the Save button | |
| | 8. The system saves the setting changes to the configuration file |
| | 9. The system switches back to the main window |

Table 5.10: F9 Functionality steps

Post: The AI assistance will be toggled on or off and the user is sent back to the main window.

| User | System |
|---|---|
| 1. The user presses the Settings button | |
| | 2. The current window is changed to the Settings window |
| 3. The user presses the toggle button in the "AI assistance" section to change status | |
| | 4. The system toggles the service on or off without saving the change |
| 5. The user presses the Save button | |
| | 6. The system saves the setting changes to the configuration file |
| | 7. The system switches back to the main window |

Table 5.11: F10 Functionality steps

The F11 requirement is responsible for hand driven drawing during video recording. The steps and requirements for this functionality can be found in the table below 5.12.

Prerequisites: The video recording is running and the AI assistance is turned on.

Post: Given a specific hand gesture the system draws on the video feed following the users finger, visible both on the live feed and in the recording.

| User | System |
|---|--|
| 1. The user shows a specific hand gesture on the camera | |
| | 2. The system whitens the pixels in a small radius near the finger of the user |

Table 5.12: F11 Functionality steps

The F12 requirement is responsible for clearing the video feed of any drawing during recording. The steps and requirements for this functionality can be found in the table below 5.13.

Prerequisites: The video recording is running and the AI assistance is turned on.

Post: Given a specific hand gesture the system clears the video feed of any drawing.

| User | System |
|---|---|
| 1. The user shows a specific hand gesture on the camera | |
| | 2. The system clears the video feed of any hand drawing |

Table 5.13: F12 Functionality steps

The F13 requirement is responsible for toggling being muted via hand gesture during video recording. The steps and requirements for this functionality can be found in the table below 5.14.

Prerequisites: The video recording is running and the AI assistance is turned on.

Post: Given a specific hand gesture the system toggles between being muted and the previous audio level.

| User | System |
|---|---|
| 1. The user shows a specific hand gesture on the camera | |
| | 2. The system toggles between being muted, audio level 0 percent and the previous level before the mute |

Table 5.14: F13 Functionality steps

The F14 requirement is responsible for zooming in a portion of the video controlled by a hand gesture. The steps and requirements for this functionality can be found in the table below 5.15.

Prerequisites: The video recording is running and the AI assistance is turned on; The video feed is not zoomed in.

Post: Given a specific hand gesture the system zooms in a portion of the video where the users hand is.

| User | System |
|---|--|
| 1. The user shows a specific hand gesture on the camera | |
| | 2. The system zooms in to the portion of the video where the users hand is present |

Table 5.15: F14 Functionality steps

The F15 requirement is responsible for zooming out of the zoomed in video feed.

The steps and requirements for this functionality can be found in the table below 5.16.

Prerequisites: The video recording is running and the AI assistance is turned on; The video feed is already zoomed in.

Post: Given a specific hand gesture the system zooms out, showing the full video.

| User | System |
|---|--|
| 1. The user shows a specific hand gesture on the camera in the zoomed in portion of the video | |
| | 2. The system zooms out, showing the full video feed |

Table 5.16: F15 Functionality steps

5.1.3 Non-functional requirements

The application only has one version for the windows operating system, the results of running it on Linux, MacOS or any other operating system is undefined, they have not been test.

The processing of the input images from the video feed is done in real-time with the frame rate being determined by the input camera. The hand gesture identification and finger tracking is done on every second or third frame, depending on frame rate of the recording in the current moment.

The saving of the video files and images are done by only accessing those particular folders, with a naming convention of Recording Screenshot with the current date and time, so conflicts with other existing files is almost impossible, at the very least highly unlikely.

5.1.4 System requirements

The application only runs Windows operating system, needing the Windows 10 or 11 64-bit version, running on other versions may result in undefined behaviour.

In terms of the CPU the application was tested on Ryzen 7 4800h with a clock speed of 2.9 Gigahertz, from which 20-25 percent was used while running. From this data, an assumption can be drawn that as a baseline, a hardware similar to an intel core i7-9750h with a clock speed of 2.6 Gigahertz is advised. By having a weaker CPU the application may run into lagging issues when it comes to the rel-time performance of image processing.

The RAM used by the application is relatively small, using only 300 megabytes from a 3200 Megahertz unit.

In terms of storage, the application only needs around 300 Megabytes. The real impact will come from the saved video and screenshot files, depending on the file type in which they are saved.

To be able to use the application, the user also needs a working camera and a working microphone.

5.2 Technical specifications

For the purpose of easier integration of the deep learning model, the language used to develop the application is python, since most frameworks used for machine learning are written in this.

For creating and working with the MobileNet architecture, I use the TensorFlow [AAB⁺15] and Keras framework [C⁺15]. TensorFlow is an open-source project developed by Google giving an easy API for developing machine learning models efficiently, by wrapping up the more complex C++ and CUDA core functionalities. It also supports Keras, which is an open-source neural network library, providing an easy interface for building and teaching models, while also containing several known ones, so they can be accessed with ease.

For the video capturing and the opencv library [Bra00], which is open-source computer vision library written in mostly C++ and it is often used in machine learning projects. Because of that reason, the data, which is given back is easy to process and use in different models, making the development process that much easier.

Since the opencv library used for the capturing the visuals does not record audio as well, I use the pyaudio library, designed to capture the input of audio devices [oT]. This is a cross-platform python library published by the Massachusetts Institute of Technology for the purpose of recording and playing sounds.

For the creation of the graphical user interface, the cross-platform Qt framework is used [Gro95]. It is a widely used service for creating high quality interfaces and is written in C++, providing good performance while supporting multiple languages like python. It provides various tools and modules, providing a relatively easy environment for UI development.

Besides the major libraries and framework, the win32api package [Mic] is also used to access certain computer specifications, like the width and height of the monitor screen, to align the application at launch.

To store the settings of the user, the json format is used, so the structure of the storage is easy to read for both the application and the user, in case a manual change would be wanted. For this the built in json library [Fun] is used in python.

Chapter 6

Application design and implementation

In order to achieve a clear structure for the application, with layers, which provide specific services, I utilize Object Oriented Programming. It is a paradigm based on object, meaning sets of data and procedures neatly packed together. The four principles on which it is built on are encapsulation, inheritance, polymorphism and abstraction [Bla94].

Encapsulation means that the data is bundled together in one coherent package. In the case of classes it means that the data and the methods provided to manipulate them are under one roof. The limitation on these fields are also part of the principle, providing a way to minimize the outside access to certain information if needed.

Inheritance provides a way to better generalize classes, by enabling to derive the attributes and functionalities from a preexisting one. For example let's take a car, which has certain methods and variable, than we would like to create a specific version, like an Audi. In this case we can inherit all the car specific things by using inheritance, while also leaving a way for the Audi class to create new features.

Polymorphism is the concept of accessing multiple types of objects through one common interface. This can usually occur in two different scenarios. The first one, static, is at compile time, for example method overloading is using the same function name for multiple methods. In this case the compiler decides when to use which based on the signatures of the procedure. The second scenario, dynamic, is at runtime, when the specific type of a variable will only be known when it gets saved. For this purpose a common interface is used, be it an actual interface or a parent class, and all the objects which implement the interface or inherit the parent class, will be seen as such by the program.

Abstraction is the concept of hiding unnecessary complexity from the user, by moving the implementation details under an umbrella object. For example looking at simple service, which creates a desk from different types of components, by

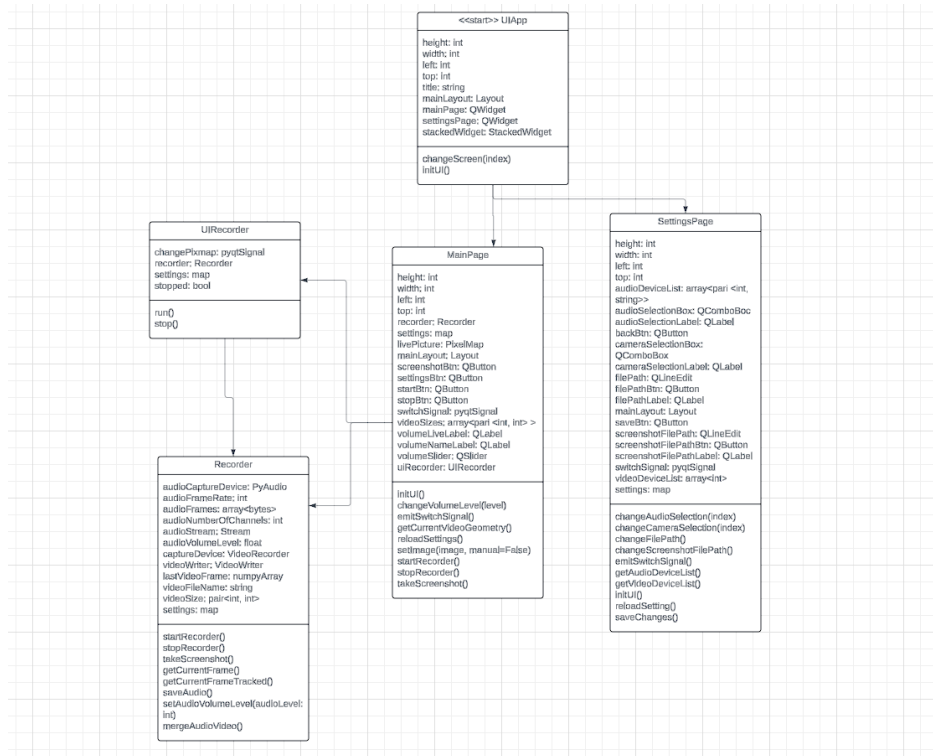


Figure 6.1: UML class diagram showing the relationships between the layers, made with [Luc]

wrapping all the implementation in one class, we hide the complexity from the end user, only needing to work with a couple of methods from the service.

With the utilization of OOP, a clear architecture is achievable for the application. I use a layered architecture similar to the Model View Controller, MVC, design pattern, visible in image6.1.

The MVC pattern is used to clearly separate the tree parts of the application for easier overview and implementation. The top layer is the view unit, be it a Graphical User Interface or any other, with which the user directly interacts with. The layer underneath is the controller one, which implements all the logical steps and calculation the application may need. It interact with the view and model layer. The last layer is the unit responsible for storing the data used, either in memory or in any other way.

In the case of this application the view and control layers are clearly present, with the help of a GUI and the Recorder service, see in image6.1. The last layer is utilized with classes from the opencv and wave modules. The VideoWriter object is used to save the captured visual footage on disk, while the wave module is for saving the audio in file. To not create another class just for wrapping this modules, resulting in some decrease in performance, I opted to leave this part in this layer. Besides this, there is also a one frame buffer present, storing the last captured visual frame. It is made in this layer for simplicity, since it is used for the screenshot functionality,

creating the model layers, would just result in efficiency loss and more boilerplate code.

In the second layer, which contains the Recorder class, I utilize the facade design pattern [Gurb]. The main premise of it is that more complex systems are hidden away from the end user, providing a more simple platform to work with. From the methods shown in image6.1, the four methods called from the view layer are the start, stop and get frame function, all of them providing the more complex system, such as saving the video files, creating and utilizing the model layer objects and taking advantage of the Machine Learning model.

The three way relationship between the MainPage, UIRecorder and Recorder is discussed in the next chapter 6.1

6.1 Graphical User Interface

The Graphical User Interface utilizes the signal based communication of the QT framework, for inter widget discussion. This is an event based system, which allows independent objects to communicate in case when something of interest happens in the application [Gro]. With this feature breaking the different pages down to different object altogether, making it simple for possible future additions.

To overcome the complexities which come with such a solution, two design patterns are used, namely the Composite and the Mediator, both of which are represented in the UI design in image6.2.

The composite design pattern states that the structure of what you are building can be represented in a tree, like a box containing more boxes and so on [Gura]. In the case of this application this pattern is in image6.2, in how the three widget classes are built. The main part, which is basically the control unit of the UI, the UIApp consists of the two page widget, from which the currently selected one is called to be shown.

The Mediator design pattern is used to keep the dependencies of different objects in check, by not letting them directly talk to each other, only through third object, the mediator[Gurc]. In the GUI this is done between the pages, when the switch happens between them, in the context of reloading the settings. When the switch occurs the two widget communicate with UIApp class as with their controller, since the change is not done by the pages. However in the mean time, the settings variable does need to be reloaded every time a switch occurs to ensure up-date settings at all-time, and that is where the mediator role comes in. For example when the Settings page releases control, it signals the Main page through the UIApp, the mediator, to update its settings.

Just how the image6.2 suggest, the UIApp plays the controller role in the User

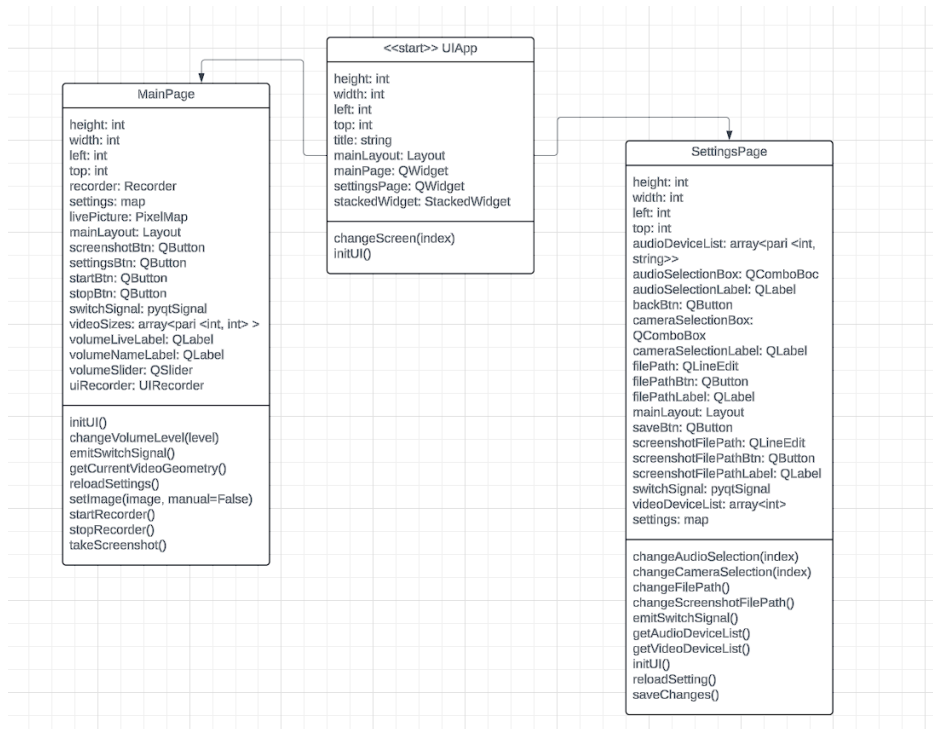


Figure 6.2: Main design of the Graphical User Interface, made with [Luc]

Interface design, making sure the widgets are shown and that they can send each other information about the occurrences of certain events.

Since the QT framework by default runs on one thread, when updating the live feed, the rest of the application becomes near unusable, since the recorder is always running. To combat this Main page, where the main functionalities can be found will not call the Recorder directly, instead creating a QThread object, UIRecorder, as can be seen in image 6.3.

Now running that the calling is done on a different thread, the returned frame has to be sent back to the Main page for visualization. For this a simplified version of the Observer design pattern is used, provided by the QT framework's signaling system. This feature allows widgets to connect to other ones in an event driven fashion. In the application the Main page will subscribe/connect to the UIRecorder's signal, changePixmap 6.3, and will be informed when the UIRecorder emits a signal. In this case the signal will be the new frame returned by the Recorder, enabling it to show it to the user.

The GUI has a minimalist look, providing the interactive elements in rows, using the Grid system of QT. In the Main page, the first row contains the live feed, the second contains the volume slider and the final row contains the navigation button Settings and the three functionality ones for starting and stopping the recording and taking a screenshot 6.4.

The Setting page is built with the same design in mind. The different sections

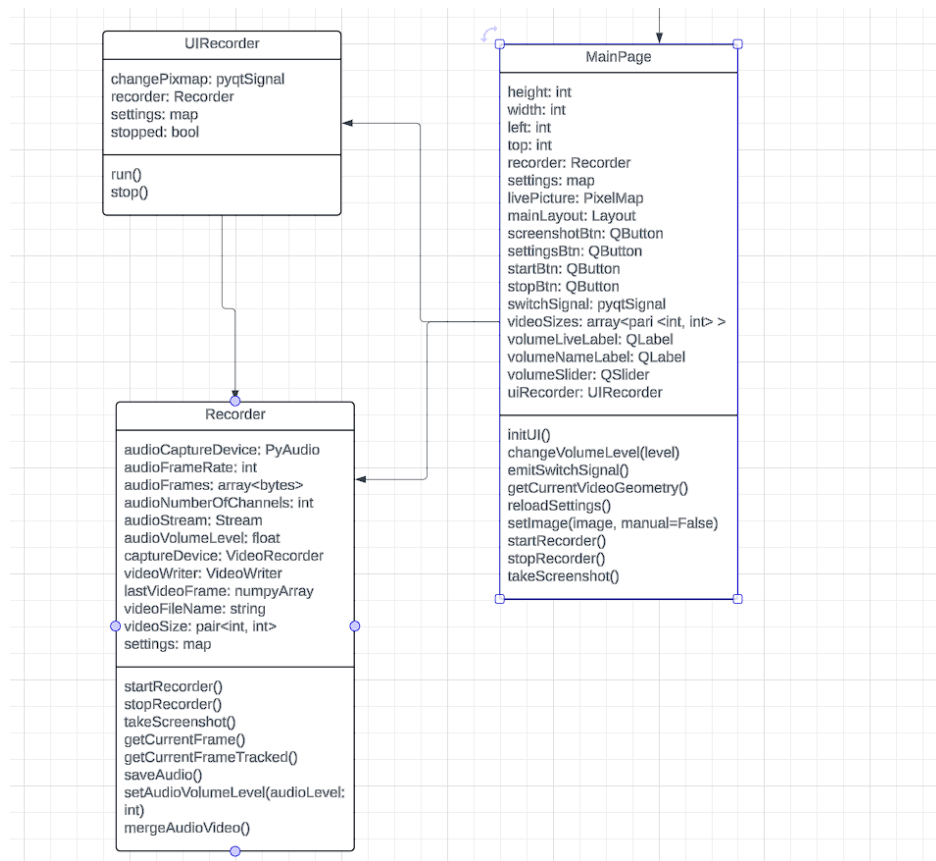


Figure 6.3: The design of calling the controller layer in UI, made with [Luc]

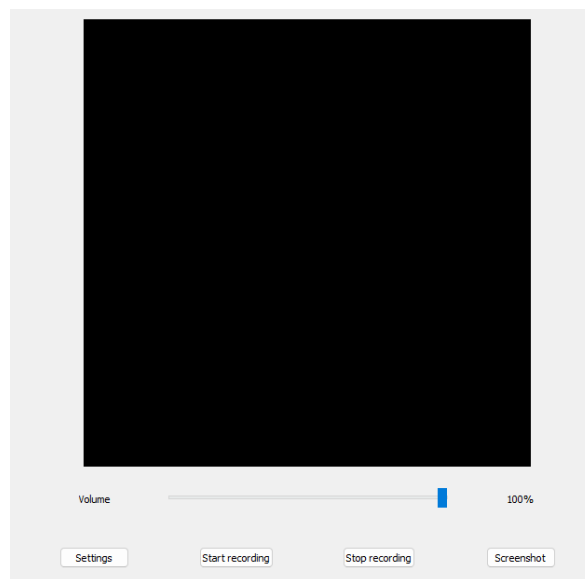


Figure 6.4: The layout of the main page

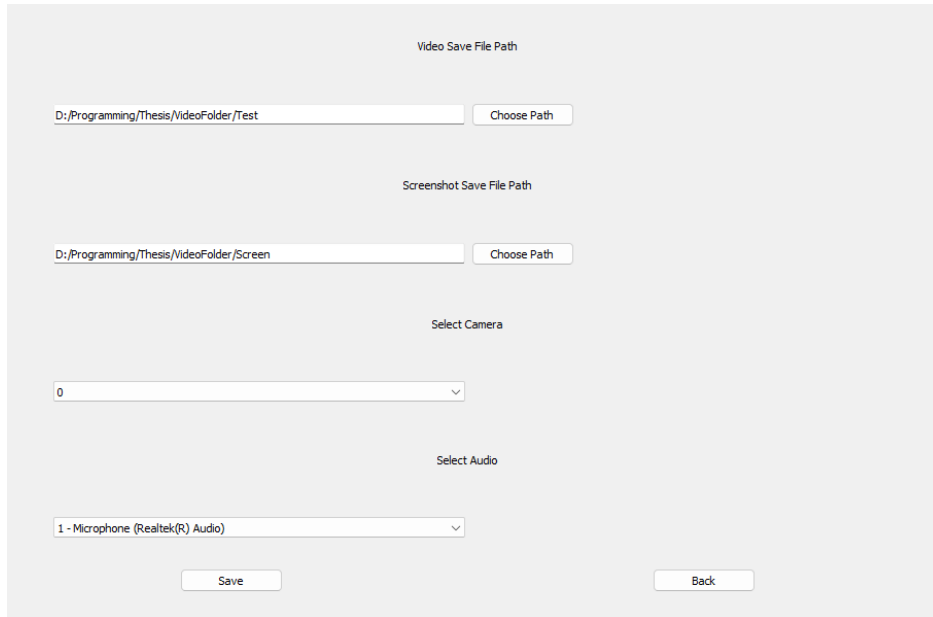
The image shows a settings page with a light gray background. It contains four main sections, each with a title and a text input field. The first section is 'Video Save File Path' with the path 'D:/Programming/Thesis/VideoFolder/Test' and a 'Choose Path' button. The second section is 'Screenshot Save File Path' with the path 'D:/Programming/Thesis/VideoFolder/Screen' and a 'Choose Path' button. The third section is 'Select Camera' with a dropdown menu showing '0'. The fourth section is 'Select Audio' with a dropdown menu showing '1 - Microphone (Realtek(R) Audio)'. At the bottom, there are two buttons: 'Save' on the left and 'Back' on the right.

Figure 6.5: The layout of the settings page

occupy two rows each and are horizontally centered, while the last row contains the two buttons, one for saving the changes, one for navigating back to the Main page. This design creates a minimalist horizontally look, keeping the theme from the Main page 6.5.

6.2 Image processing

The image processing is contained within the Recorder object. It has two functions to get the current recorded frame, a simple one and one where the deep learning model is utilized. The GUI calls the perspective model, depending on whether the AI assistance is toggle on or not. The model is loaded in this layer from the start of the application, for quicker response time when needed, than the function works with the predicted hand gesture and if needed calculates where the current drawing pixels have to be placed in the image.

6.3 Functionality implementation

6.3.1 F1 implementation

The F1, start recording, functionality utilizes a QThread, in order to enable the Main page to function properly during video recording. In this case the UIRecorder does not communicate directly with the Main page, but through the QT provided event based signal system. Besides the multi threaded nature of this feature, the flow of

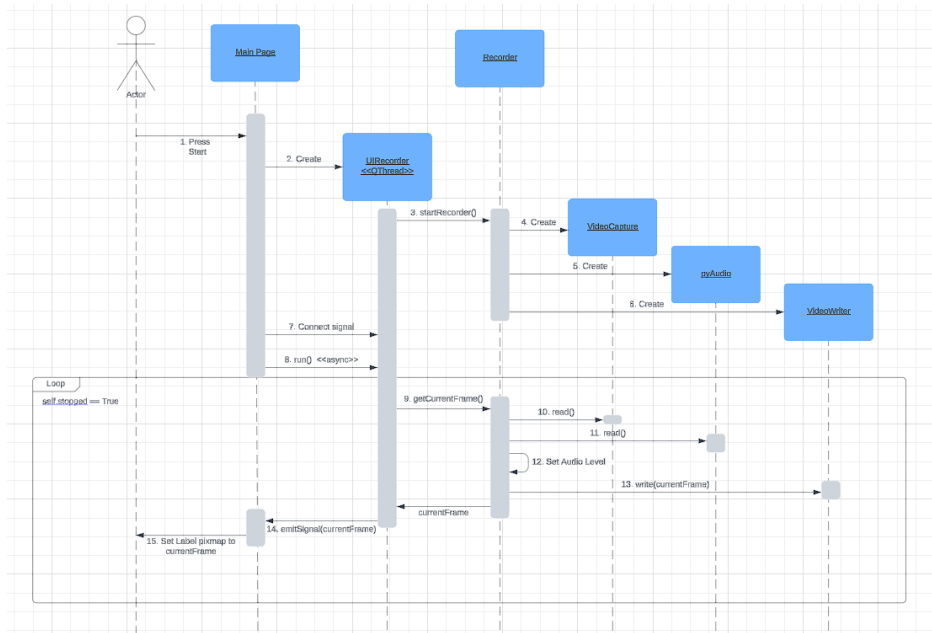


Figure 6.6: Sequence diagram of functionality F1, made with [Luc]

the calls is still quite linear, as can be seen in image6.6.

The first step after the user hits the Start button is to set up the recording environment. To achieve this the MainPage widget will create a UIRecorder object which inherits the properties of QThread. The class saves all the parameters sent to it, like the pointer to the Recorder object and the settings map variable, than at the end of the initialization process it calls the Recorder's startRecorder function. This method will create the VideoCapture, VideoWriter and pyaudio object with the necessary configuration, couple of them being hard coded, like the number of bytes in an audio stream chunk, the rest is stored in the settings variable, like the chosen camera and microphone indexes. After the initialization of the thread is complete the MainPage widget connects its signalSlot, the setImage method, to the UIRecorder's changePixmap signal, which will be the communication channel between the two objects. And finally through an asynchronous call the MainPage widget runs the UIRecorder.

The running UIRecorder works in an infinite loop until the F2 functionality stops it. In this it gets the current frame by calling the getCurrentFrame or getCurrentFrameTracked methods of the Recorder. The main premise of both of the functions is that they read the input from the VideoCapture and pyaudio object. The getCurrentFrame processes the audio by adjusting the volume level, multiplying the audio array by a float between 0 and 1. The getCurrentFrameTracked does the same thing with the addition of processing the image with the help of the machine learning model. After the processing the write method of the VideoWriter object is called to save the visual frames to the initialized video file, than the current frame is returned

to the UIRecorder. When the current frame is returned a signal is emitted through the `changePixmap` variable, sending the frame to the MainPage widget as a QImage object, which then sets the pixel map of the live feed label to the returned image.

In image6.6 the AI assistance free version is presented. With image processing enabled there would be two changes made, firstly the function call at step 9 would change to `getCurrentFrameTracked()`, secondly there would be an extra step present between 12 and 13, called `processVisualInput()`. This method would be responsible for making the predictions with the help of the Deep Learning model and taking actions accordingly by manipulating the visual frames or by changing the volume level in the Recorder.

Chapter 7

Application testing

7.1 Testing techniques

For testing the application the most used technique was exploratory testing. This method can also be well described by trial and error, since its building blocks are trying out different scenarios of the application, analyzing the result, learn why things succeeded or failed and finally design the application in future with new found knowledge from the previous step. In the case of the application the testing was done by hand and by one person.

7.2 Use case testing specifics

In the case of the F1 functionality, starting the recording, the main focus was that the frames are correctly shown and saved. To test this, a thorough check was made to ensure the route of the current frame by using debug tools. The saving was checked during recording by copying the file and also after it finished in order to check that the images are saved correctly. To check the audio the route was checked and the final output after the capture was done. The final step was to ensure that the correct folder was used as the saving destination by changing the path multiple times between recordings, and checking the video files as mentioned before.

Chapter 8

User manual

This chapter is a guide, going through all the steps for using the application

8.1 Starting the application

8.2 Starting the recording

When being on the Main page, seen at image8.1, starting the recording is as simple as pressing the "Start recording" button. After 1 or 2 seconds, the recording start, which can be seen when the black square changes to the live camera feed, like in image8.2.

While the video is being recorded, the visual part is saved in real-time to the folder selected in the settings, named as TempRecording, as seen in image8.3. This file is not the whole video, the audio part is merged with the visual when the recording stops, and opening it until it is finished is not advised, since the behaviour of the application in that case is unknown.

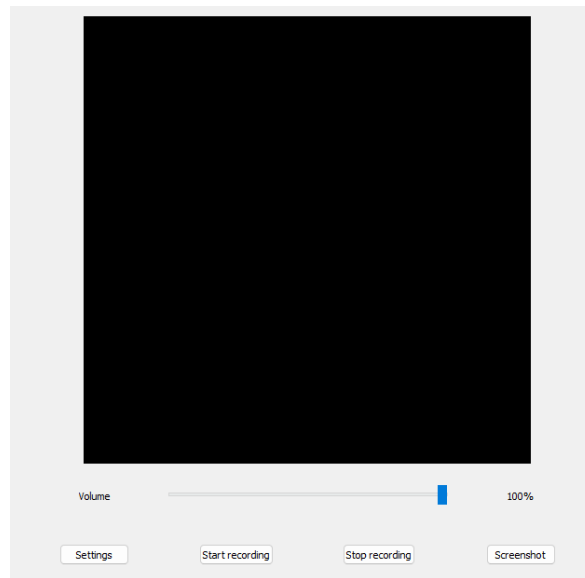


Figure 8.1: How the Main page looks right after starting the application

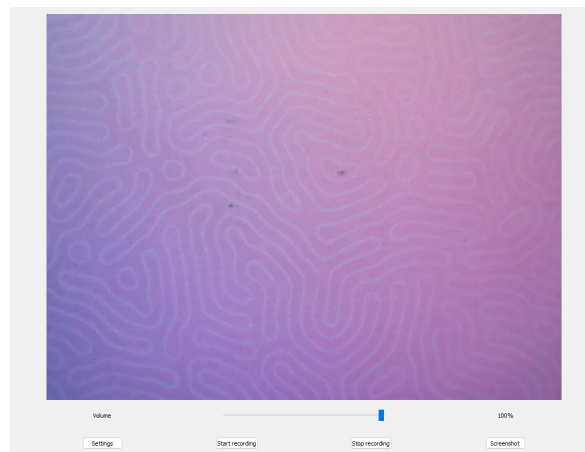


Figure 8.2: How the Main page looks when recording a video

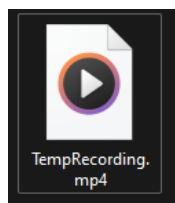


Figure 8.3: The visual file in the video folder

Chapter 9

Future Work

9.1 Machine Learning Model Improvements

9.2 GPU Utilization

Chapter 10

Conclusions

Bibliography

- [AAB⁺15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [Bla94] Günther Blaschek. *Principles of Object-Oriented Programming*, pages 9–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.
- [Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [C⁺15] Francois Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015.
- [DDS⁺09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [Den12] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
- [DH72] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, jan 1972.

- [Fun] Python Software Foundation. Json library. <https://docs.python.org/3/library/json.html>.
- [Gro] Qt Group. Qtsignal. https://www.tutorialspoint.com/pyqt/pyqt_signals_and_slots.htm.
- [Gro95] Qt Group. Qt. <https://www.qt.io/>, 1995.
- [Gura] Refactoring Guru. Composite. <https://refactoring.guru/design-patterns/composite>.
- [Gurb] Refactoring Guru. Facade. <https://refactoring.guru/design-patterns/facade>.
- [Gurc] Refactoring Guru. Mediator. <https://refactoring.guru/design-patterns/mediator>.
- [HZC⁺17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.
- [HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [JBM⁺16] Vijay John, Ali Boyali, Seiichi Mita, Masayuki Imanishi, and Norio Sanma. Deep learning-based fast hand gesture recognition using representative frames. In *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8, 2016.
- [KSH22] Maria Kopf, Marc Schulder, and Thomas Hanke. The Sign Language Dataset Compendium: Creating an overview of digital linguistic resources. In Eleni Efthimiou, Stavroula-Evita Fotinea, Thomas Hanke, Julie A. Hochgesang, Jette Kristoffersen, Johanna Mesch, and Marc Schulder, editors, *Proceedings of the LREC2022 10th Workshop on the Representation and Processing of Sign Languages: Multilingual Sign Language Resources*, pages 102–109, Marseille, France, 2022. European Language Resources Association (ELRA).
- [Luc] Lucid. Lucidchart. <https://www.lucidchart.com>.
- [MAY⁺21] Abdullah Mujahid, Mazhar Javed Awan, Awais Yasin, Mazin Abed Mohammed, Robertas Damaševičius, Rytis Maskeliūnas, and Karar Hameed Abdulkareem. Real-time hand gesture recognition based on deep learning yolov3 model. *Applied Sciences*, 11(9), 2021.

- [Mic] Microsoft. Win32api. <https://learn.microsoft.com/en-us/windows/win32/api/>.
- [oT] Massachusetts Institute of Technology. Pyaudio module. <https://people.csail.mit.edu/hubert/pyaudio/>.
- [RF18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.
- [Unk] Unknown. Straight line Hough transform. https://scikit-image.org/docs/stable/auto_examples/edges/plot_line_hough_transform.html. Online; accessed 23 March 2024.
- [VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.
- [VMGMST⁺23] María Villa-Monedero, Manuel Gil-Martín, Daniel Sáez-Trigueros, Andrzej Pomirski, and Rubén San-Segundo. Sign language dataset for automatic motion generation. *Journal of Imaging*, 9(12), 2023.
- [WHJ21] Fei Wang, Ronglin Hu, and Ying Jin. Research on gesture image recognition method based on transfer learning. *Procedia Computer Science*, 187:140–145, 06 2021.