BABEŞ-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
Dr.Mihoc Tudor-Dan, Lector Universitar

Author
Sali Arnold

2024

ABSTRACT

Besides audio, visual articulation of some thoughts can prove quite useful while recording videos. In order to create a more convenient environment in terms of the human-machine interactions, hand gesture recognition proves a great tool. It is a frequently researched field in computer vision with a variety of solutions, from line calculations to machine learning. In this paper the latter is utilised in a video recording application, which aims to create an environment where visual representation of certain ideas can be seamlessly made with static hand gestures. Taking other researches in to account in this field in order to achieve this feat, the TensorFlow pretrained Mobilenet object detection model is utilized, fitted on a custom dataset, created by me. While not perfect the method achieves on average 90+ percent on the five available gestures with reasonable fps on a video card, depending on camera and computer performance and good lightning from the front. For the video recording the opencv, for the visual part, the pyaudio, for the audio part, and the wave, for combining the two, modules are used. The software is only available on windows, because of the path choosing, running on other operating systems will result in undefined behaviour. The application is made with only five interaction, however the addition of additional ones is possible by retraining the deep learning model on the expended dataset and creating handling function for the new features.

Contents

Chapter 1

Introduction

Hand gestures, whether directly or indirectly have always been a big part of our communication. This is the reason why the field of detecting them has been frequently researched by those interested in computer vision. In the early days, it was achieved by a more straightforward approach, like calculating the contrasts between pixels, to segment the part that was needed.

As computer performances advanced so did this field, by enabling more complex and more nuanced computations. As years passed, the emergence of machine learning created a new approach, mostly with deep learning models, by leaving the majority of complexities to the artificial intelligence models. This approach allowed developers to create a more reliable identification method. This and the constant increase in computational power opens the door for real-time applications working with hand gestures. With this easier accessibility for those in need and a quantity of quality of life functionalities are achievable, like gesture based control of a software.

Utilizing this a lot of areas can be improved, by providing an easier environment to work in. One of the most important field in my opinion being teaching, structured or unstructured. It being an old field of research, a lot of ways of conveying information have been theorized and tested, however methods, and way of learning differs from person to person. Because of this difference providing an environment where the creation of different approaches is simplified, could prove useful for those interested in teaching others and also for people who want to learn by providing more choices. While videos are an amazing way of spreading information, and very advanced software's are available, providing different visuals without a third party accessory could still prove challenging. This is the main focus of my paper to create a video recording environment where creating simple visuals on the fly by drawing is made easier. While the main focus is on drawing by the means of hand gestures and hand position, other quality of life features, namely audio level changing by gestures are available.

Besides the size of the software, another limitation comes from the usage of the

pyaudio module. Since the recording and saving of audio data is not quite straight-forward, the application simply saves this at the end of the video recording. This means the video length will have an upper bound based on available memory.

While the usage of deep learning models seems promising in order to achieve higher detection precision compared to other methods, there was one big barrier for a long time, which is that running them was very expensive compared to the average hardware available for people. While this paper will dwell in that, a real deep dive is not made, only shown that real-time computation is possible. Although the application takes a considerable hit in frames per second, it is also very much dependent on the computers capabilities. Having said all that on a mid-range machine watchable performance is still achievable, if using the right model.

In order to create an application which is capable of providing the aforementioned gesture driven features, appropriate modules are used, namely wave, opencv and pyaudio for video recording and TensorFlow for the utilization of the AI model. In case of the machine learning part for better results an already pretrained model will be used, taking advantage of the transfer learning method.

In the following chapters I will go into more detail about the different parts of the software, starting with already existing research about image segmentation algorithms and deep learning models in chapter 2.

Following that I am looking into the performance of different machine learning models, taking into account research from the field.

In chapter 4 the specifications of the model used in the application will be presented alongside the descriptions about the data collection and preprocessing.

The second half of the paper contains the development and structure of the application, starting with the requirements and specifications. After that I will go into detail about the design and testing in the following two chapters.

Closing down the paper a chapter will be presented about possible improvements in the future, than drawing the conclusion in the final chapter.

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 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, solving problems such as object detection in an image or video, scene understanding and image generation. The solutions from this field open the door for new innovations, which can be seen in the everyday life in the form of image filters, self-driving cars and so on.

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. As time passed, more sophisticated ones were created, such as Hough Transform, the Viola-Jones face detection and machine learning.
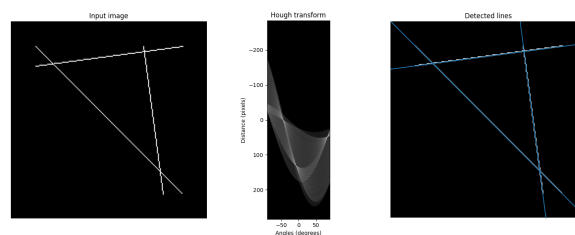

Figure 2.1: Hough transform calculation to find the lines of a triangle [Unk]

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 core idea behind it is a voting process made by the curves in the transform space, creating cluster points or local maxima, which represents the existence of a shape since the curves contain the parameters of it. From those parameters the position of the shape in the input image can be detected, 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, which made this procedure a massive breakthrough in image processing. It 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]. Their approach being based on the usage of Haar-like feature and Adaptive Boosting.

The Haar-like features are simple rectangular regions in the input image data, which are being used for decreasing the number of pixel calculations and for getting a better understanding of a specific region. By calculating the differences in the sum of pixel intensities between the regions, the contrast can be learned, which allows the detection of basic patterns, like edges, lines and textures.

Adaptive boosting is a machine learning algorithm used for enhancing weak classifiers, by training the model in iterations, focusing on one feature each time. At each pass, the algorithm selects the best performing classifier and merges it with the final one, with a weight proportional to its performance. By increasing the weight of incorrect classifications for the next step, it achieves a dynamically changing priorities, boosting the model even more.

With the help of these two procedures this method 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. Because of its low requirements it is still widely used on weaker 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. As more power machines become available the parameters used in these techniques increased accordingly, allowing the learning of more complex features. In this section I choose some deep learning architectures, which were significant achievements, and are still in use today.

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 values and various anchor boxes, which are bounding boxes with a predefined shape, size and aspect ratio, the model predicts the final bounding box for the searched object. [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	$64 - 1024$	$3x3/2$	$128x128 - 8x8$
Convolutional	$32 - 512$	$1x1$	
Convolutional	$64 - 1024$	$3x3$	
Residual			$128x128 - 8x8$

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 comes from addressing 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. This allows the network to be able to learn on a much larger scale compared to previous designs, by being able to increase the depth 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, although in those cases the hardware requirements can also get out of hand.

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 for reasonable performance on limited hardware.

The architecture is built using two techniques: the depth-wise and point-wise convolutions. In the first one a single convolution filter is applied for each input channel, capturing features separately, while the second applies a 1x1 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 difference between the complexity of the two architectures can be seen, tested on the ImageNet dataset, trading around 1 percent accuracy for around seven times less parameters, greatly decreasing the computational quantity.

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

Table 2.3: Difference in parameters between the two method and fully convolutional MobileNet architectures [HZC+17]

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 and depth. With this the accuracy and speed of the models have increased, allowing for more and more use cases, like real-time predictions even 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 and overall size, while also maintaining a useful accuracy. There are a handful of such architectures, from which I will compare three.

For comparing the performance of the 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 reasonable frame rate of 24 frames per second, it means a running time needed of around 41 milliseconds. In practice this metric is dependent on a lot of variables, so it is not a hard requirement, just something to put the performances of the architectures in contrast.

3.1 Model comparisons

3.1.1 Representative frames with deep learning

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, instead of a whole segment, with a specific algorithm, 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.

3.1.2 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 an approximate accuracy rating of 98 percent in real-time, although time specifications were not provided.

3.1.3 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 pretrained parameters of MobileNet on the ImageNet dataset [DDS+09], than 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, from personal testing, it is safe to assume, that near real-time performance is achievable with their approach.

In Table 3.1 the comparison of two MobileNet models, one with and one without a random forest ending, can be seen with regard to their accuracy ratings on three different hand based image datasets. From the results we can conclude that both architectures can achieve good accuracy. Combining that with the low hardware requirements of the model, we get a perfect combination for a usage in real-time applications.

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

Creating and training a model from scratch requires a lot of resources in terms of hardware and data, even with smaller models in order to achieve high performance, so in order to ensure good results I will utilize transfer learning with a pretrained MobileNet architecture.

4.1 MobileNet specifications

The used model is downloaded from the TensorFlow Model Garden [Tenb], specifically the ssd-mobilenet-v2-fpnlite-320x320-coco version. The ssd, meaning single shot detector, is an architecture which uses the MobileNet model as its base. Fpn means feature map network, which is a method to extract information from multiple resolution levels in the image and perform the object detection on those. Coco is the name of the dataset on which the model is initially trained and the 320x320 means the image resolution for training.

4.1.1 Model specifications

The architecture being a single shot detector means that one pass through is enough for predicting the objects and their bounding boxes [LAE$^+$16].

In figure 4.1 we can see the overall architecture of a general ssd fpn model. In the case of this application the first few convolutional layers, the deep learning back-

Figure 4.1: Architectural diagram of the SSD MobileNet model [LAE$^+$16]

bone, is replaced with the MobileNet architectures first few layer, which have been discussed in section 2.2.3. The following few convolutional layers are the feature map network part. In this section the features processed by the machine learning model are made smaller and smaller, with every one of them having a connection to the prediction layer, hence enabling the architecture to learn with different ratios.

The predictions are done with help of 3x3xp kernels, where p is the number of channels in the feature map, which produce category scores or shape offsets for the bounding boxes. In the end filters are used at each cell in the feature maps to calculate 4 offsets relative to specified default bounding boxes [LAE$^+$16].

4.1.2 Original dataset

The model was pretrained on the COCO dataset, which contains around two hundred thousand labeled object in around 80 different categories, specifically made for the purposes of object detection by a team of researchers [LMB$^+$15].

4.1.3 Model output

This architecture outputs a number of values regarding bounding boxes in raw and relative, value between 0 and 1, form and the predicted classes with their probabilities. In the case of the application three values are the most important: the class index, the prediction confidence and the relative bounding boxes. The confidence and index are needed for all hand gestures, while the box coordinate values are used when calculating the pixel positions for drawing.

4.2 Data

In order to have creative freedom over the hand gestures, a custom dataset was made by me with five hand gestures and 20 pictures each. Some examples can be seen in figure 4.2. The images than were labeled by hand with labelimg tool [Tzu] and transformed into TF records, which is a specific format for the tensorflow object-detection api when teaching models with the script provided by them [Tenc].

4.3 Training

The training of the model was done with the help of the provided script and configuration file by the tensorflow api [Tend]. The configurations were personalized with the relevant paths to the label and record files and with the output path. The

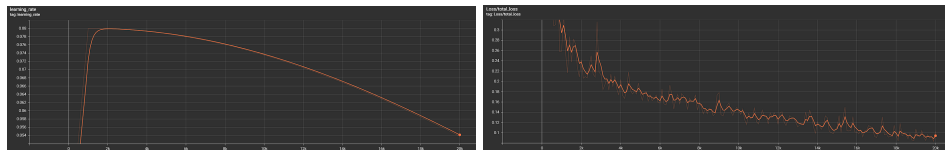Figure 4.2: Pictures for each functioning hand gesture in the dataset

Figure 4.3: The changes of the learning rate and loss values throughout the training process

training was conducted with a batch size of 8 and 5 classes, decreasing learning rate and 20000 steps on an Nvidia 1660 Ti gpu.

4.4 Metrics and running specifications

During training the learning rate slowly decreased in order to achieve a better approximations of the function as the model got closer to a local minimum, and the loss of the prediction have declined in a nice curve up until to a sufficient point, where it is low enough and the risk of over-fitting is not that high as can be seen in figure 4.3.

As for the accuracy in real usage, the model can predict with around a 90 percent accuracy when the hand gesture features are visible. This means that the hands are not far enough from the camera, around 3 meters, that details can be barely visible, and an adequate light source is behind the camera.

For saving the application a script was provided by the api [Tena], which exports the trained model in the format of the base tensorflow library, which makes the running of it easier in an application. To utilize the model the only thing that has to be done is to load it inside the application and add a batch layer to the image.

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 trained on a custom hand gesture dataset. 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 features for drawing on the video and clearing it, for toggling mute and for incrementally changing the audio level up and down using simple hand gestures shown in chapter 4.

Besides the drawing and clearing functionalities all others have a specific buttons and sliders as well, allowing the user to completely ignore the machine learning part if they choose to. In that case the application works as simple video recording tool with settings options for camera and microphone input and path choices for video and screenshot folders. This is shown in a more compact and visual way in the use case diagram below 5.1, which was created with the help of an online diagram creation 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 Table5.1, while the descriptions will be in the ones after.
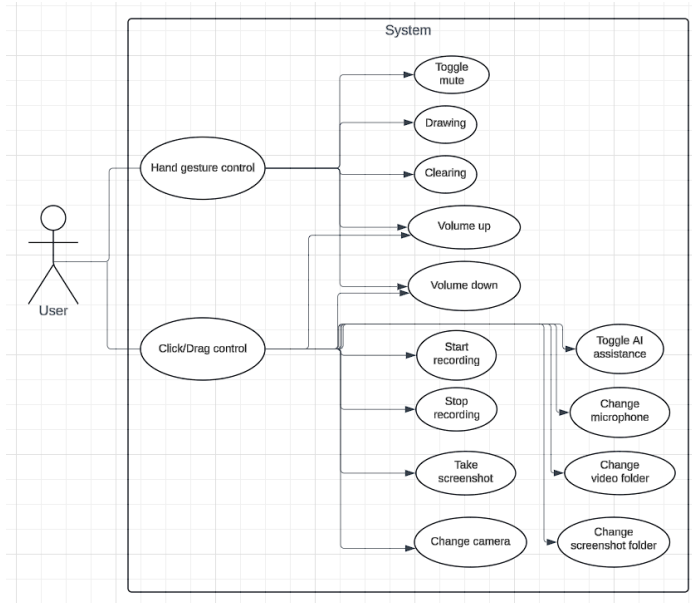
Figure 5.1: Use case diagram for the functionalities of the application

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

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 message is shown to inform the user that the recording has started and in case the recording could not be started.

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 with appropriate message
	4.2 The system shows a pop-up error message about the failure at start or during recording

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 stop.

User	System
1. User presses the Stop button	
	2. The system stops the recording
	3. The system creates the final video file in the selected folder
	4 A black image is shown instead of the live feed and appropriate message is presented

Table 5.3: F2 Functionality steps

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 message is shown in case of success and error.

User	System
1. User presses the Screenshot button	
	2. The system saves the last frame in the chosen folder
	3.1 A pop-up message is shown when successful
	3.2 The system shows a pop-up error message about the failure

Table 5.4: F3 Functionality steps

The F4/F5 requirements are responsible for changing the volume down and up respectively. The steps and requirements for these functionalities can be found in the table below 5.5.

Prerequisites: The volume is not at 0 percent(down)/100 percent(up); 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(down)/right(up)	
1.2 The user show the second(down)/first(up) hand gesture in figure 4.2	
	2. The system saves the volume modifier accordingly

Table 5.5: F4/F5 Functionality steps

The F6/F7 requirements are responsible for changing the folder path for the video/screenshot files. The steps and requirements for these functionalities can be found in the table below 5.6.

Prerequisites: The video recording is not running.

Post: The folder path for the video/screenshot 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 Screenshot/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 5.6: F6/F7 Functionality steps

The F8/F9 requirements are responsible for changing the used camera/microphone. The steps and requirements for these functionalities can be found in the table below 5.7.

Prerequisites: The video recording is not running.

Post: The chosen camera/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 5.8.

Prerequisites: None.

Post: The AI assistance will be toggled on or off.

The F11 requirement is responsible for hand gesture driven drawing during video recording. The steps and requirements for this functionality can be found in the table below 5.9.

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.

The F12 requirement is responsible for clearing the video feed of any drawing

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/Microphone section	
	4. The systems shows a list of available cameras/microphones
5. The user chooses a camera/microphone from the list	
	6. The system changes the camera/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 5.7: F8/F9 Functionality steps

User	System
1. The user checks or unchecks the AIAssistance button	
	2. The system toggles the feature on or off

Table 5.8: F10 Functionality steps

User	System
1. The user shows the fourth hand gesture in figure 4.2	
	2. The system whitens the pixels in a small radius near the end of the fingers of the user

Table 5.9: F11 Functionality steps

during recording. The steps and requirements for this functionality can be found in the table below 5.10.

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 the fifth hand gesture in figure 4.2	
	2. The system clears the video feed of any hand drawing

Table 5.10: 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 5.11.

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 the third hand gesture in figure 4.2	
	2. The system toggles between being muted and the previous audio level before the mute

Table 5.11: F13 Functionality steps

5.1.3 Non-functional requirements

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 couple frame, depending on the 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 needs a working camera and microphone and it only runs on the Windows operating system, 10 or 11 64-bit version, running on other versions may result in undefined behaviour.

By testing the application on a Ryzen 7 4800h and NVIDIA GeForce GTX 1660 Ti, 8 and 30 percent being used respectively, the minimum requirements are Intel Core i7-9750h and a dedicated NVIDIA video card for a smooth experience. The application can run solely on CPU as well, but in that case the lag might too much for a watchable video. In terms of RAM the application needs a minimum of 2600 MB, more is necessary depending on the length of the recording.

In order to run the application using the graphics card the 11.2 version of CUDA has to be installed and setup up, which is a tool developed by NVIDIA for general parallel programming on their GPUs.

In terms of storage, the application only needs around 1.8 gigabytes. The real impact will come from the saved video and screenshot files.

5.2 Technical specifications

For the purpose of easier integration of the deep learning model, the development language is python, where most frameworks used for machine learning are written.

For creating and working with the SSD MobileNet architecture, I use the TensorFlow [AAB+15] framework. 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.

The opencv library [Bra00] is used for capturing and working with visuals. It is open-source computer vision library written in mostly C++ and it is often used in computer vision projects, making the integration with the model easier. The pyaudio module [oT], a cross-platform library, is used for capturing the audio, while the wave module [Pyh] is used for saving it.

For the creation of the graphical user interface, the cross-platform Qt framework is used [Gro95]. With the help of this library interfaces can be built in a modular fashion, making development easier and it is written in C++, which provides good performance.

Besides the major libraries, the win32api package [Mic] is also used to access certain computer specifications, like the width and height of the screen, for aligning the application at launch. To store the settings of the user, the json format is used, with the help of the json library [Fun], so the structure of the storage is easy to read for both the application and the user, in case a manual change would be needed.

Chapter 6

Application design and implementation

For a clear application structure with different layers each providing specific services, I utilize Object Oriented Programming. It is a paradigm based on objects, 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, with possible limitations providing a way to minimize outside access.

Inheritance provides the ability for one instance to derive the attributes and functionalities of other classes, enabling generalization.

Polymorphism builds upon the concept of generalization, making it possible to access 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, which is using the same function name with multiple signatures. The second, dynamic, is achieved at runtime, using a common derived interface, the exact type of an object will only be known when created.

Abstraction is the concept of hiding unnecessary complexities, by moving implementation details under an umbrella object. This can happen in multiple ways, depending on what scenario is it used in, like data oriented or object implementation.

With the help of OOP I use layered architecture, similar to the Model View Controller design pattern, achieving a clear structure, with clear segmentation of responsibilities as seen in image 6.1.

The layering enables the application to have three distinct parts, with different responsibilities. The top layer is the view unit, which is responsible for setting up the

<<start>> UIApp

height: int
width: int
left: int
top: int
title: string
mainLayout: Layout
mainPage: QWidget
settingsPage: QWidget
stackedWidget: StackedWidget

changeScreen(index)
initUI()

UIRecorder

changePixmap: pyqtSignal
soundChangeSignal: pyqtSignal
errorSignal: pyqtSignal
recorder: Recorder
settings: map
stopped: bool

switchAISupport(isChecked)
handleCommand(command)
run()
stop()

MainPage

height: int
width: int
left: int
top: int
recorder: Recorder
settings: map
livePicture: PixelMap
mainLayout: Layout
screenshotBtn: QButton
settingsBtn: QButton
startBtn: QButton
stopBtn: QButton
switchSignal: pyqtSignal
videoSizes: array<pari <int, int> >
volumeLiveLabel: QLabel
volumeNameLabel: QLabel
volumeSlider: QSlider
uiRecorder: UIRecorder
...

initUI()
changeVolumeLevel(level)
emitSwitchSignal()
getCurrentVideoGeometry()
reloadSettings()
setImage(image, manual=False)
setAudioLevel(level)
errorHandling(value)
switchAISupport(value)
startRecorder()
stopRecorder()
takeScreenshot()

SettingsPage

height: int
width: int
left: int
top: int
audioDeviceList: array<pari <int, string>
audioSelectionBox: QComboBox
audioSelectionLabel: QLabel
backBtn: QButton
cameraSelectionBox: QComboBox
cameraSelectionLabel: QLabel
filePath: QLineEdit
filePathBtn: QButton
filePathLabel: QLabel
mainLayout: Layout
saveBtn: QButton
screenshotFilePath: QLineEdit
screenshotFilePathBtn: QButton
screenshotFilePathLabel: QLabel
switchSignal: pyqtSignal
videoDeviceList: array<int>
settings: map

changeAudioSelection(index)
changeCameraSelection(index)
changeFilePath()
changeScreenshotFilePath()
emitSwitchSignal()
getAudioDeviceList()
getVideoDeviceList()
initUI()
reloadSetting()
saveChanges()

Recorder

audioCaptureDevice: PyAudio
audioFrameRate: int
audioFrames: array<bytes>
audioNumberOfChannels: int
audioStream: Stream
audioVolumeLevel: float
captureDevice: VideoRecorder
videoWriter: VideoWriter
lastVideoFrame: numpyArray
videoFileName: string
videoSize: pair<int, int>
settings: map
...

startRecorder()
stopRecorder()
takeScreenshot()
getCurrentFrame(assistance=False)
saveAudio()
setAudioVolumeLevel(audioLevel: int)
mergeAudioVideo()
setSettings(settings)
drawLines(frame)
dynamicFPSHandler(frame)
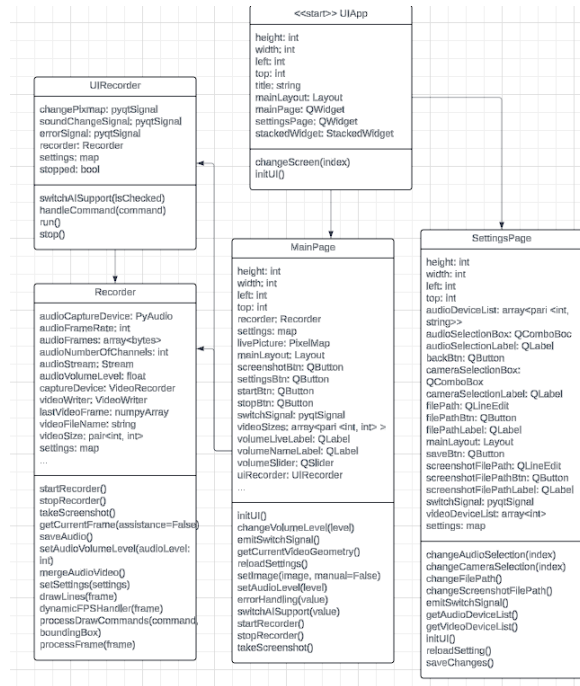processDrawCommands(command, boundingBox)
processFrame(frame)

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

graphical user interface, with which the user interacts. Under it comes the controller layer, which includes the logic behind the application features. At the bottom the data layer is present, which retrieves the images and the audio.

While the top two layers are clearly present in the application, seen in image 6.1, the last one implemented with the help of the opencv pyaudio and wave modules, in order to retrieve and save the data. Objects from these modules are saved in the Recorder class, the controller, and they act as repositories, a middleman for interacting with data.

To create a clear object for the logical steps, I utilize the facade design pattern [Gurb] in the Recorder, hiding most of the complexities behind three simple functions: start, stop and getCurrentFrame. With the help of some parameters and setter functions, the controller can simply be used by the upper layer.

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 with each other [Gro]. With the help of this functionality, since it allows child to parent communication, the pages are broken down to independent objects, with a master class holding them together, providing an architecture where adding more pages is quite simple. To overcome
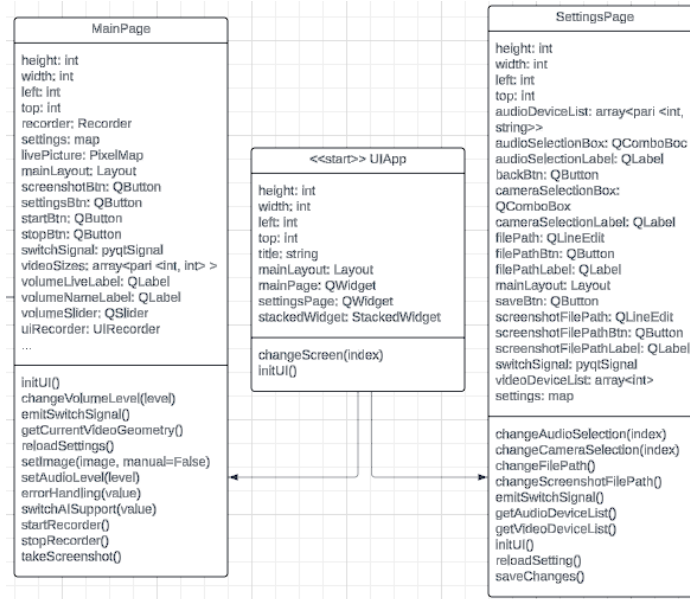
Figure 6.2: Main design of the Graphical User Interface, made with [Luc]

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 image 6.2.

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]. This is clearly present in the UI design 6.2, where the main element is built from different page blocks, showing one at a time.

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]. This role is also up to he UIApp element, since the objects from which its built do not know about each other, they have to invoke the other through the the third party when switching pages.

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 the Main page will not call the Recorder directly, instead creating a QThread object, UIRecorder, as can be seen in image 6.3.

This complicates the communication between the two view and controller layers, so in order to solve the problems the Observer design pattern [Gurd] is utilized with the help of the already mentioned QT signaling system. This is achieved by the Main page subscribing to the changePixmap slot of the UIRecorder, which will send the retrieved frame back in an event based fashion.

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 the volume slider, the third an empty label, used for pop-up messages and
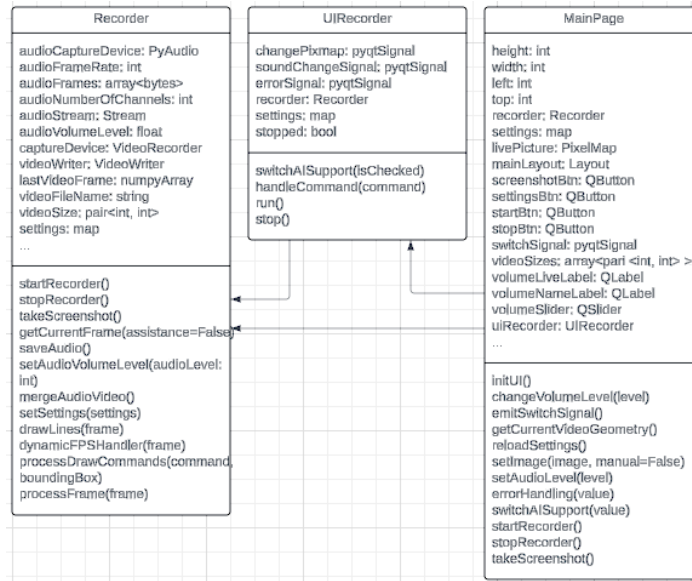
Figure 6.3: The design of calling the controller layer in UI, made with [Luc]
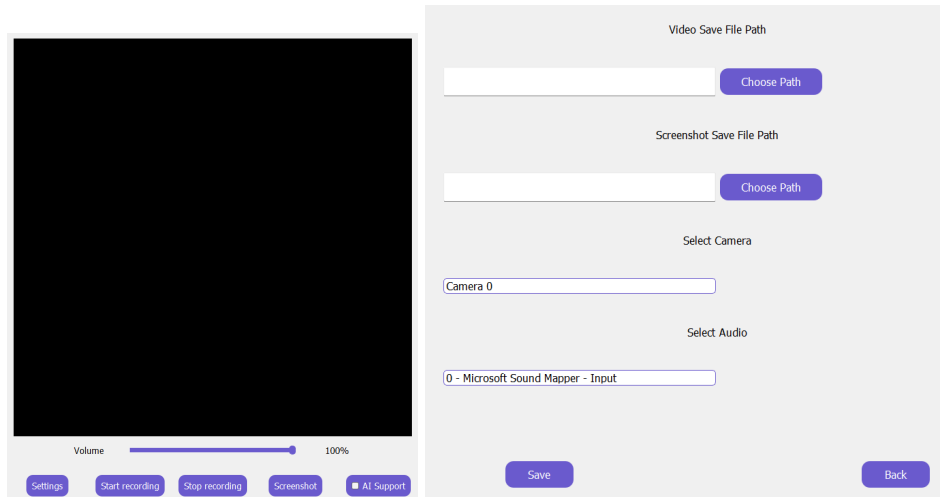

Figure 6.4: The layout of the main page

the final row the navigation and functionality buttons as seen in the first image in figure 6.4.

The Setting page is built with the same design in mind. The different sections occupy two rows each and are horizontally centered, while the last row contains the two buttons, one for saving the changes and one for navigating back to the Main page as seen in the second image 6.4.

6.2 Image processing implementation

The image processing is contained within the Recorder object. As mentioned before a deep learning model is used to detect the position and category of five hand

```
#Running the detection
detections = self.model(inputArray)

# Formatting the detection
detectionBoxes = detections['detection_boxes'][0].numpy()
detectionClasses = detections['detection_classes'][0].numpy().astype(numpy.int32)
detectionScores = detections['detection_scores'][0].numpy()
returnClass = None
boundingBox = None

#Looping through the detections
for i in range(len(detectionBoxes)):
    if detectionScores[i] >= self.predictTreshold:
        #Get the name of the predicted class
        for c in self.classes:
            if c['id'] == detectionClasses[i]:
                returnClass = c['name']
                break
        yMin = int(detectionBoxes[i][0] * self.videoSize[1])
        xMin = int(detectionBoxes[i][1] * self.videoSize[0])
        yMax = int(detectionBoxes[i][2] * self.videoSize[1])
        xMax = int(detectionBoxes[i][3] * self.videoSize[0])
        boundingBox = [yMin, xMin, yMax, xMax]
return returnClass, boundingBox
```

```
#If there was a prediction handle it, after a couple of frames from the last detection
if command is not None:
    #In order to bypass accidental detections, a number of detections are needed from the same hand gesture
    if self.currentPrediction == command:
        self.predictConfidenceCounter -= 1
    else:
        self.currentPrediction = command
        self.predictConfidenceCounter = self.predictConfidenceCounterMax - 1

    if self.predictConfidenceCounter <= 0:
        self.processDrawCommands(command, boundingBox)
        if command != "draw":
            self.predictFrameWaitCounter = self.predictFrameWaitCounterMax
            self.currentPrediction = None
```

```
#Dinamicly dial down the rate at which the AI model processes images in case of too much delay
if self.frameRepeatedCounter > 10:
    self.frameRepeatedCounter = 0
    if self.frameTimeMax < 5:
        self.frameTimeMax += 1
```

Figure 6.5: Code snippets from the hand gesture prediction part

gestures. This model is loaded from the start in order to be readily available when needed. The detection is done in the getCurrentFrame method, with a parameter toggling the use of it. It is done in two waves, the first being predicting the classes and bounding boxes. This is simply done by running the AI model, then retrieving the necessary information, than multiplying the relative bounding boxes from the model with the image dimensions as can be seen in the first image from figure 6.5.

In order to minimize ghost detections and provide some breathing room between hand gestures, two counters are implemented, one calculating similar detections in a row and one for waiting after a successful hand gesture was detected, seen in image two 6.5.

For maximizing the fps in the video a dynamic dial is implemented in image three 6.5, which will increment a counter, stating how many frames have to skipped between model uses up to a maximum of five in order for the AI assistance to be usable.

6.3 Fixing fps count dynamically

Since the recording of the audio and visual data is done separately, keeping a steady fps is needed for merging them correctly at the end. For this reason I implemented a simple fps handler as seen in image 6.6. With two time variables at the beginning and end of the frame calculations, it calculates how many milliseconds have passed, than compares that with the value of the camera's FPS. If the current running time is too slow, the current frame is repeated and a new audio batch is saved to keep up.

```
def dynamicFPSHandler(self, frame):
    #This function is responsible for keeping track of the
    while self.fpsTimeRemainder >= 1./self.videoFrameRate:
        self.videoWriter.write(frame)
        self.fpsTimeRemainder -= 1./self.videoFrameRate
        audioData = self.audioStream.read(1024)
        chunk = numpy.fromstring(audioData, numpy.int16)
        chunk = chunk * self.audioVolumeLevel
        audioData = chunk.astype(numpy.int16)
        self.audioFrames.append(audioData)
        self.frameRepeatedCounter +=1

    #Dinamicly dial down the rate at which the AI model pr
    if self.frameRepeatedCounter > 10:
        self.frameRepeatedCounter = 0
        if self.frameTimeMax < 5:
            self.frameTimeMax +=1
```

Figure 6.6: Repeating frames to keep up with the cameras fps

```
#Creating the capturing device for the visual part of the video
try:
    self.captureDevice = cv2.VideoCapture(self.settings["cameraChoice"])
except Exception:
    self.captureDevice = None
    return 1
```

```
@pyqtSlot(int)
def errorHandling(self, returnValue):
    # Depending on the returnValue handle the possible errors
    if returnValue == 1:
        self.stopRecorderUtil("Invalid camera settings, recording cannot be started!")
    elif returnValue == 2:
        self.stopRecorderUtil("No video path found, cannot start recording!")
    elif returnValue == 3:
        self.stopRecorderUtil("Invalid audio settings recording cannot be started!")
    elif returnValue == 4:
        self.stopRecorderUtil("Video or Audio input error!")
```

Figure 6.7: Error handling in the controller and view layers

6.4 Error Handling

The error handling inside the application is done using try/except statements in the controller. In case of an exception a value will be returned to the UIRecorder, which will reset the controller if the issue occurred during recording, stop itself and send the value to the main page, where message will be presented to the user, depending on the integer. The messages are shown with the help of the QTimer object, which enables the application to clear the message label after a certain period of time. An example of an exception catching and the decoding of the integer value to a user message can be seen in images 6.7.

6.5 Functionality implementation

6.5.1 Recording implementation

The video recording starts with the initialization of the thread UIRecorder from the main page, which will initialize the opencv and pyaudio object in the controller layer and call the getCurrentFrame method of the Recorder class in while loop, until its stopped. From here the image is transformed to QImage, the image class of the QT framework, to be sent back to the main page and presented to the use, as shown in the first image in figure 6.8.

In the controller layer, the retrieval of the data done simply by reading values from the pyaudio and VideoCapture "repository" objects, after which comes the image processing part if it enabled. The next step of the getCurrentFrame function

```
#Getting the current frame from the recorder
while not self.stopped:
    frame, command = self.recorder.getCurrentFrame(self.aiSupport)
    #Handle the returned command, mainly the ones which change the sound volume
    self.handleCommand(command)

    if frame is not None:
        #Converting the image for the pyqt module
        rgbImage = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        h, w, _ = rgbImage.shape
        convertToQtFormat = QImage(rgbImage.data, w, h, QImage.Format_RGB888)

        #Send the image to the mainPage
        self.changePixmap.emit(convertToQtFormat)
```

```
#Handle the volume commands: mute, voicedown, voiceup
if command == "mute":
    self.soundChangeSignal.emit(0)
elif command == "voicedown":
    self.soundChangeSignal.emit(-5)
elif command == "voiceup":
    self.soundChangeSignal.emit(5)
elif command == "ReadError":
    self.stopped = True
    self.errorValue = 4
    self.errorSignal.emit(self.errorValue)
```

```
#Create the paths of the audio and video files
audioPath = self.settings["savePath"] + "/TempAudio.wav"
videoPath = self.settings["savePath"] + "/TempRecording.mp4"

#Create instances of VideoFileClip and AudioFileClip
videoEditor = VideoFileClip(videoPath)
audioEditor = AudioFileClip(audioPath)

#Merge the audio and video clips
fullVideo = videoEditor.set_audio(audioEditor)

#Write the merged video file to the saved file
fullVideo.write_videofile(self.videoFileName)
```

Figure 6.8: Recording implementation

is saving the processed image directly to a file, and the audio to a numpy array, after it has been multiplied by the audio level variable, a number between 0 and 1.

At the end of the function we call the fps handler portion before returning the current frame and the retrieved hand gesture class as command to the UIRecorder, for it to forward the changes to the UI for user feedback as shown in the second image 6.8.

When the recording is stopped, the command cascades down from the view to the controller layer, where every variable will reset and as a final step the audio file saved and merged with the visual. The merging is done with the help of the moviepy module, which loads both temporary files, concatanetes the audio to the visual and outputs a final mp4 file, image three 6.8.
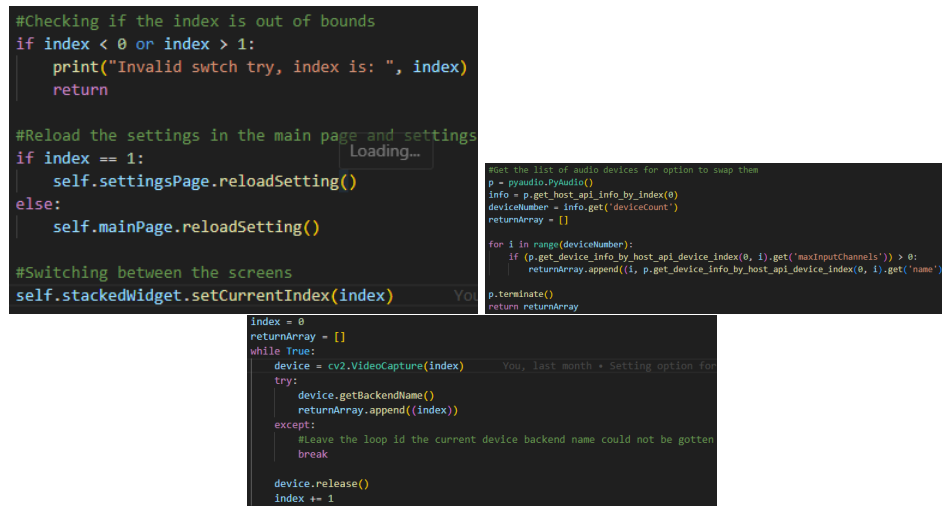
6.5.2   Screenshot implementation

For better performance in this functionality the current frame is always saved in the Recorder object, so only an image write has to be done with the help of the opencv module's imwrite method.

6.5.3   Volume change implementation

The level change in the audio data is done by multiplying it with a values between 0 and 1 as mentioned in 6.5.1. The change of the audio level can be separated into two different scenarios. The first one is done with the slider found in the main page. In this case a function is connected to the slider in an event based fashion, which will directly set the volume variable of the Recorder object.

The second case is via the hand gesture controls. After the successful prediction of the class, that class is sent back to the UIRecorder as command, which will decode

```
#Checking if the index is out of bounds
if index < 0 or index > 1:
    print("Invalid swtch try, index is: ", index)
    return

#Reload the settings in the main page and settings
if index == 1:
    self.settingsPage.reloadSetting()
else:
    self.mainPage.reloadSetting()

#Switching between the screens
self.stackedWidget.setCurrentIndex(index)
```

```
#Get the list of audio devices for option to swap them
p = pyaudio.PyAudio()
info = p.get_host_api_info_by_index(0)
deviceNumber = info.get('deviceCount')
returnArray = []

for i in range(deviceNumber):
    if (p.get_device_info_by_host_api_device_index(0, i).get('maxInputChannels')) > 0:
        returnArray.append((i, p.get_device_info_by_host_api_device_index(0, i).get('name')))

p.terminate()
return returnArray
```

```
index = 0
returnArray = []
while True:
    device = cv2.VideoCapture(index)
    try:
        device.getBackendName()
        returnArray.append((index))
    except:
        #Leave the loop id the current device backend name could not be gotten
        break

    device.release()
    index += 1
```

Figure 6.9: UIApp screen change and settings implementations

it, and send back integer values to the main page, as seen in image two 6.8. That value is sent as the parameter to the same function from the first case, from where the steps become the same as before. The up and down propagation of the data was an intentional design choice, in order to avoid writing almost duplicate functions for the variable change in the MainPage , which would have been necessary otherwise, since the UI elements are event based, so changing the slider values would call the first case no matter what.
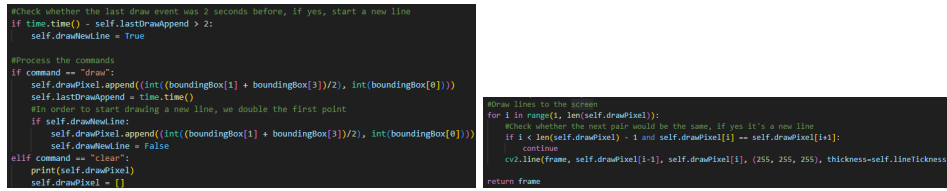
Every time a change occurs the current volume level is stored, the only exception being in the case of mute, when the variable is simply put to zero or to the stored value, depending on the initial audio level.

6.5.4 Changing the settings

Switching from one page to the next is done with the help of QT signal as discussed in 6.1, by sending an integer value, the index of the current page, to the UIApp mediator. This class will than refresh the settings, by loading them from the json file, and change the stacked widget index, changing the page which is shown to the user, as shown in the first image 6.9.

For choosing the path for the video and screenshot folder I use the tKinter module, which opens the File explorer in windows. From there the folder is chosen, and path is written in the text field. For the microphone options I retrieve the available devices from the pyaudio module. For the camera options, since there is no built in functionality, I try to open a VideoRecorder with different indexes until one that fails, creating a list if eligible indexes for working cameras. The implementations are presented in images two and three 6.9.

When the settings have been chosen the user can press the Save button which

Figure 6.10: The implementation details of the drawing functionality

will save them in a json file.

6.5.5 Drawing implementation

The drawing functionality is driven by the fourth and fifth hand gestures shown in figure 4.2, and it is a special case in the prediction process, since this gesture will not create a pause between detection for the sake of a smoother drawing.

For every predicted draw class, the top middle point is calculated from the bounding box coordinates and then saved in an array. This means that for clearing the screen a simple clear of the point array is enough. To show the drawing the application iterates through these points and draw lines between them with the help of the opencv module, shown in the first image 6.10. This results in a continuous figure even when predictions are slow due to hardware limitations. In order to be able to start multiple figures, if there was a delay between draw class predictions, the middle point is saved twice, than at the draw stage checked, creating one skip. This achieves a break in the line, so the whole drawing will not be continuous, as seen in image two 6.10.

Whenever we have points saved the lines have to be drawn, so the corresponding function is called at every step, and since the method just returns when the array is empty, it is called whether or not the AI assistance is turned on, for the sake of a cleaner code.

Chapter 7

Application testing

7.1 Testing techniques

The application was tested using exploratory testing, with little to no scripting. The method is based on trying out different scenarios, analyzing the results, learning the reasons behind them, and finally redesigning parts of the application with newly gained information. Since the application uses a lot of modules from different creators, scripting proved to be challenging in a lot of cases, so most of the testing was done by hand. Most of the error handling was a direct result of the testing process, to ensure application robustness.

7.2 Functionality testing

The functionalities were mainly tested based on the functional descriptions provided in section 5.1.2. Every one them was analyzed under ideal conditions and under different edge cases.

Recording was tested with different settings option and missing one, while also taking into consideration abrupt stops, and hardware changes during the process. Stopping was tested thoroughly to ensure that the video is saved in the right folder, with all of the changes done to the frame and audio, including frame by frame checks of the merged video to ensure consistency. Since the screenshot taking is a relatively simple feature, the testing mainly consisted by trying out edge cases, like the start and ending of the recording and checking if the image was saved successfully.

The volume changes were tested both during and outside the recording process, except the hand gesture controls, which are only present during recording, checking the audio levels in the final video.

Since the settings are saved in a json file, the changes made were also tested out-

side the application, checking if they are imported correctly. The most important checked was the correctness of the saved data, in every step, so most of the functionalities was checked with a debugger, to check for settings changes.

The drawing part was tested with different edge cases regarding the frequency of the hand gestures, trying to arrive at a sweat spot when it comes to starting a new line. Besides this the functionality was tested in different lighting and distances for gaining insight about the performance of the model.

7.3 Model testing

To understand the performance of the model, the most important information was always the training and evaluation data during learning. However this data does not guarantee that in reality the model will perform how it supposed to be, since image conditions are always different. For all hand gesture features, multiple scenarios were tested, including low lighting, different backgrounds, far distance and obstacles. With this I was able to understand the performance and the limitations of the model better, which was crucial in the training phase.

Chapter 8

Future Work

8.1   Machine Learning Model Improvements

The current model performs quite well in the ideal conditions stated in chapter 4. Most of the shortcomings in other situations stem from the training data. In order to improve upon the model the dataset should be improved with approximately thousand of images, with different image conditions, including lighting, background distance and different obstacles, which takes some time, but is the next improvement step of the application.

8.2   GPU Utilization

The tensorflow framework has built-in gpu utilization, however setting it up is independent of the application, because of the CUDA drivers. Another improvement is to try and achieve GPU utilization automatically, and looking into other types of GPU tools, since the current version only supports NVIDIA made graphics cards, which can run the 11.2 version of CUDA.

Chapter 9

Conclusions

In this this thesis I demonstrated that real-time hand gesture control with deep learning models, in order to provide, firstly, a convenient interaction with a live video feed and second, a drawing tool for better self expression, is possible locally on a mid-range machine.

I presented the the importance of deep learning models in the field of computer vision, while also exploring the performance metrics and shortcomings of different architectures, in order to find one which balances accuracy and quick runtime, ending up with the MobileNet model. After successfully finding a good model I went into the details of the applications, providing detailed descriptions about each functionality, while also providing hardware and software requirements. This continued with explained design choices describing the whole application and the solutions to some problems coming from some of the modules used. At the end some implementation details are provided with testing specifications.

With the constant rise of hardware, machine learning has become a very promising option for image processing. Even though most deep learning models are still too expensive to run on generic machines in real-time, hence most applications utilize massive servers and internet access, this thesis presents that the technology is now in a usable state for mass adoption even on local applications.

Even though this solution is not without its own problems, and it is not at the same level of accuracy as more advanced models, the machine learning and hardware sectors are in constant evolution, meaning the future, in terms of improvements, looks promising.

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.

[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 Fundation. 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`.

[Gurd]   Refactoring Guru. Observer. `https://refactoring.guru/design-patterns/observer`.

[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).

[LAE+16]   Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.

[LMB+15] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bour- dev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[Luc] Lucid. Lucidchart. `https://www.lucidchart.com`.

[MAY+21] Abdullah Mujahid, Mazhar Javed Awan, Awais Yasin, Mazin Abed Mohammed, Robertas Damaševičius, Rytis Maskeliūnas, and Kar- rar 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/`.

[Pyh] Pyhton. Wave module. `https://docs.python.org/3/ library/wave.html`.

[RF18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improve- ment, 2018.

[Tena] Tensorflow. Export script. `https:// tensorflow-object-detection-api-tutorial. readthedocs.io/en/latest/training.html# exporting-a-trained-model`.

[Tenb] Tensorflow. Model garden. `https://www.tensorflow.org/ tfmodels`.

[Tenc] Tensorflow. Tfrecords. `https:// tensorflow-object-detection-api-tutorial. readthedocs.io/en/latest/training.html# create-tensorflow-records`.

[Tend] Tensorflow. Training script. `https:// tensorflow-object-detection-api-tutorial. readthedocs.io/en/latest/training.html# training-the-model`.

[Tzu] Tzutalin. Labelimg. `https://github.com/HumanSignal/ labelImg`.

[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.