UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Technology

Barbara Wilson Soto

Video Face Swapping

Bachelor's Thesis (12 ECTS)

Curriculum Science and Technology

Supervisor(s):
Professor, PhD Gholamreza Anbarjafari

Data Scientist, M.A. Doğuş Karabulu

Tartu 2020

Video Face Swapping

Abstract:

Face swapping is the challenge of replacing one or multiple faces in a target image with a face from a source image, the source image conditions need to be transformed in order to match the conditions in the target image (lighting and pose). A code for Image Face Swapping (IFS) was refactored and used to perform face swapping in videos. The basic logic behind Video Face Swapping (VFS) is the same as the one used for IFS since a video is just a sequence of images (frames) stitched together to imitate movement. In order to achieve VFS, the face(s) in an input image are detected, their facial landmarks key points are calculated and assigned to their corresponding (X,Y) coordinates, subsequently the faces are aligned using a procrustes analysis, next a mask is created for each image in order to determine what parts of the source and target image need to be shown in the output, then the source image shape has to warp onto the shape of the target image and for the output to look as natural as possible, color correction is performed. Finally, the two masks are blended to generate a new image output showing the face swap. The results were analysed and obstacles of the VFS code were identified and optimization of the code was conducted.

Keywords:

Computer vision, Video Face Swapping, Face Detection, Digital Image Processing

CERCS: T111 Imaging, T120 computer technology, P170 Computer science

Pealkiri eesti keeles (title in Estonian)

Lühikokkuvõte:

Selles mallis kirjeldatakse ingliskeelse lõputöö mall, stiilid lehekülgede formaati. Dokumendis on kohatäitja igaks lõputöö osaks ja on lühikirjeldused, mis peab olema kirjas igas töö osas.

Näovahetusena mõistetakse käesolevalt lähtekujutiselt saadud ühe või mitme näo asendamist sihtpildil. Lähtekujutise tingimusi peab transformeerima, et nad ühtiksid sihtpildiga (valgus, asend).

Pildi näovahetus (IFS, Image Face Swapping) koodi refaktoreeriti ja kasutati video näovahetuseks.

Video näovahetuse (Video Face Swapping, VFS) põhiline loogika on sama kui IFSi puhul, kuna video on olemuselt ühendatud kujutiste järjestus, mis imiteerib liikumist. VFSi saavutamiseks tuvastatakse nägu (näod) sisendkujutisel, arvutatakse näotuvastusalgoritmi abil näojoonte koordinaadid, pärast mida joondatakse näod Procrustese meetodiga. Järgnevalt luuakse igale kujutisele image-mask, määratlemaks, milliseid lähte- ja sihtkujutise osi on vaja näidata väljundina; seejärel ühitatakse lähte- ja sihtkujutise kujud ja võimalikult loomuliku tulemuse jaoks viiakse läbi värvikorrektsioon. Lõpuks hajutatakse kaks maski uueks väljundkujutiseks, millel on näha näovahetuse tulemus.

Tulemusi analüüsiti ja tuvastati VFS koodi takistused ning seejärel optimeeriti koodi.

Võtmesõnad:

Arvuti nägemine, Video näovahetus, Näotuvastus, digitaalne pilditöötlus
CERCS: T111 Pilditehnika, T120 arvutitehnoloogia, P170 Arvutiteadus

TABLE OF CONTENTS

List of Figures

List of Tables

TERMS, ABBREVIATIONS AND NOTATIONS

Video Face Swapping (VSF) – a process in which a face from a target image is swapped with the face from a source image, the target image are frames from a video and after the process is done, the output images will be stitched together to form a new video.

Image Face Swapping (IFS) - a process in which a face from a target image is swapped with the face from a source image.

Artificial Intelligence (AI) – refers to the field and technology that simulates human intelligences in computers with the goal to get those computer to act and behave like humans would.

Machine Learning (ML) – a branch of AI focusing in allowing system to learn from experience automatically without having all the known or unknow parameters be programmed into the system.

Digital Image Processing (DIP) – processing digital images through algorithms in computers.

Bias/Variance Tradeoff (BVT) – then a prediction model has lower bias it will possess high variance and vie versa. This trade off happens when trying to minimize both the bias and variance of a prediction model to work beyond its original training set.

Support Vector Machines (SVM) – SVM models used in supervised learning to analyze data and helps with data regression and classification.

Histogram of Oriented Gradients (HOG) – an object detection feature descriptor employed in computer vision.

Euclidean distance (ED) – the ED is the distance among two points in Euclidean space.

Ordinary Procrustes Analysis (OPA) – the Procrustes matching between a minimum of two observations using.

INTRODUCTION

In current times, our relationship with technology is quite complex, constant communication and sharing of information have become the norm. At the same time our privacy has never been more important, there are photos or videos of almost everyone alive somewhere on the internet (Mahajan et al., 2017). Our sense of security and privacy are changing and with it our concepts of reality are changing as well. The phrase “one image is worth a thousand words” alludes to a past illusion that images were irrefutable evidence towards whatever statement one was trying to prove. This concept started changing in the 1980 when the first photo editing computer programs were released. Before these software programs existed, manipulating images was an extremely risky process and only few professionals were capable of doing it (Rossner and Yamada, 2004). As time passed, photo editing softwares kept improving and digital photo manipulations became more mainstream (Story, n.d.). A similar situation happened with videos, but the process was still too expensive, therefore these technologies were mainly used by entertainment companies, like movie/TV studios or big companies (Microsoft, Apple) which had enough capital to cover the costs. Thus, the belief that a photo or a video was a reliable source of information was still alive and well during these times.

The idea of who could use technology and how accessible it was, changed on the 9th of January 2007, the first time the iPhone was shown to anyone outside of Apple (Hjorth, 2012). While the original iPhone was a game changer, it’s content was extremely limited since it only came with 16 pre-installed apps, this limited the end user experience and considering that back then the original iPhone was behind in many technical specs in contrast with its competitors, Apple needed to go the extra mile to keep their momentum going. In the original release one of the 16 apps pre-installed was Google maps, which was the best iteration of the program in any platform Google maps had at the time. Not only was the app able to fully use all the new functionalities of the iPhone, it made sense to the users to use the program on the go, rather than the desktop version. This hinted at a necessity that previously handheld devices weren’t even able to diagnose. This necessity was fulfilled when a year later Apple released iOS2, this update introduced the App store and opened the gates for third party applications to be created and shared in a marketplace, this allowed users to personalize their phones and transformed the way people interacted with their handheld devices. Users had a computer in the palm of their hands and as long as the use was within the hardware

capabilities, end users were only limited by the creativity of third party developers, who in turn created the apps available for iPhone, is not a wonder that Android followed with its own marketplace for third party apps for its phones later on.

This environment permitted the rise of social media “tech companies” like Facebook, this in turn, started competition of which company was capable of not only retaining the most users but also having them use the App for as long as possible. This resulted in an incredible period of digital image processing and computer vision innovations due to the basic component of social media, which is sharing content (images and videos) between it’s users. Among these innovations, face swap became a hit with mobile users around 2016. MSQRD was a company that launched its app for iOS and Android that allowed it’s end users to add filters to the faces as well as swapping faces with their friends to create “video selfies”. This company made headlines because three months after it was launched Facebook bought it to gain an edge over its competitor Snapchat, the actual figure is unknown but the figure is estimated to be in the millions of USD. Only a year later, most social media had integrated similar versions of such face filters and face swaps functions. Then, in 2017 synthetic media like Deepfakes (deepfakes, 2020) became mainstream knowledge when it swapped porn stars faces with celebrities and the public started to ask if their identities could be falsified through similar technology as well. Thanks to the help of powerful AI and machine learning techniques, fake content capable of fooling the human eye became a reality (Kietzmann et al., 2020).

Regardless if the intent is to create new algorithms or improved old ones, in order to develop new ways for users to interact with their apps and phones or do further research in order to better understand the dangers of face swapping and how to prevent them. Face swapping technology will keep improving as companies and institutions race to create better and faster face swapping algorithms and programs, while simultaneously investing millions of USD in this technology. Current face swapping programs involve different AI and Machine Learning (ML) techniques and methods (Dale et al., 2011)(Korshunova et al., 2017)(Chen et al., 2019), in this thesis VFS will be performed without employing memory expensive AI or ML techniques. In this thesis, an Image Face Swapping (IFS) code will be refactored to work with videos, afterwards the result will be analyzed and obstacles of VFS will be identified, finally the code will be optimized to improve the VFS results.

1 LITERATURE REVIEW

In the next section the approaches and theory relevant to this thesis will be covered. It will cover the research background i.e. technologies and approaches used to comprehend, explore and interpret the problems as well as how to solve them.

1.1 Research Background

VFS involves a wide variety of research topics and technologies. To comprehend what VFS is and how it works let's start from the basis for VFS, which is computer vision and digital image processing, which are used for face and facial landmarks key points detection, assessment of lighting conditions and skin colour (Adouani et al., 2019) and saving them as data. Afterwards moving to the algorithms used for transforming the data into the desired state and then optimizing said results to look as realistic and smooth as possible after VFS (Garrido et al., 2014).

1.1.1 Digital Image Processing (DIP)

Image processing is any mathematical operation perform on an image, for example some operations can be if one wants to zoom in or out an image, transform a black a white image into a colour image or take some information from the image (International Conference on Computer and Communication Technologies et al., 2016). Image processing is a type of multidimensional signal processing that contains different approaches to enhance, modify, extract information, compress and transform images. Images can be treated as spatial or 2D signals, which is why images can be subjected to signal processing techniques like filtering, attenuating, etc. (International Conference on Computer and Communication Technologies et al., 2016). Since vision is the most developed among our senses, image processing is among the fastest growing technologies. Image processing is employed in order to enhance human analysis capabilities of pictographic attributes (International Conference on Computer and Communication Technologies et al., 2016), therefore it can be used from biology to engineering to computer science. Image processing can be summarized as having an input image, then analysing and manipulating said image and finally understanding the output, which can be an altered image or image analysis (Young et al., 1998).

This thesis is focused on digital image processing, which is manipulation of digital images by using computers. DIP can also be used for videos, since videos consist of multiple images

(frames) put together to form video. A digital image is a quantized depiction of an analogue image, specifically a digital image is a two-dimensional function $f(x, y)$ where x and y are the spatial (plane) coordinates, therefore an image can be represented as a matrix. The amplitude of "f" at any pair of coordinates (x, y) is the intensity of an image (Young et al., 1998). If x, y and the amplitude values of "f" are finite and discrete quantities, the image is a digital image, i.e. a digital image is composed of a finite number of elements called pixels, each of which has a particular location and value. The pixels surrounding a certain pixel are its neighbourhood, thus various neighbourhoods arranged in a significant order results in an image. A neighbourhood is defined by its shape the same way a matrix is (Young et al., 1998). Digital images can be defined as either Binary, Grayscale and Colour Images. Binary images have two different possible intensity values for every pixel, they are displayed as black and white images, with the numerical values of 0 for black and 1 or 255 for white, as a result of this, for a binary image only one bit per pixel is needed.

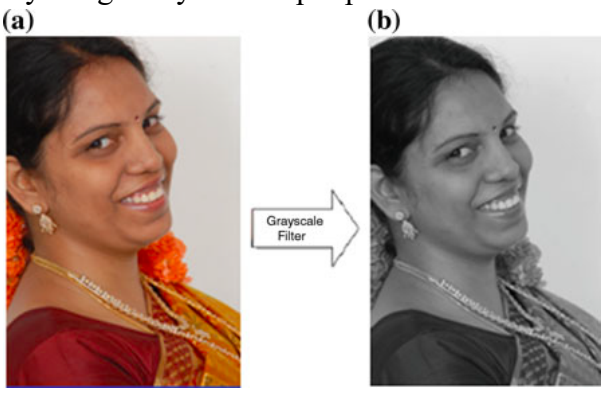


Figure 1 - (a) Original image and (b) grayscale version of the same image (International Conference on Computer and Communication Technologies et al., 2016).

Greyscale images (Fig.1b) are achromatic, or without colour, therefore in a greyscale image, every pixel has an intensity of grey, between the lower strength side of 0 (black) to the stronger side of 255 (white). Every pixel value is a description of its brightness capacity. The ranges of pixel values imply that each pixel can be represented by 8 bits or 1 byte. The 8-bit format is one of the most popular image formats out there. It has 2^8=256 shades of colour in it, ranging between 0-255, where 127 is grey. As explained before an image is a 2D function, which can be represented by a 2D matrix or array, in the case of a grayscale image, its 2D matrix would have values between 0 and 255. This is not the case with colour images.

Under regular illumination conditions the human eye perceives more colour than brightness in images (International Conference on Computer and Communication Technologies et al., 2016). Colour can be conveyed as the mixture of three factors of red, green, blue (RGB). A

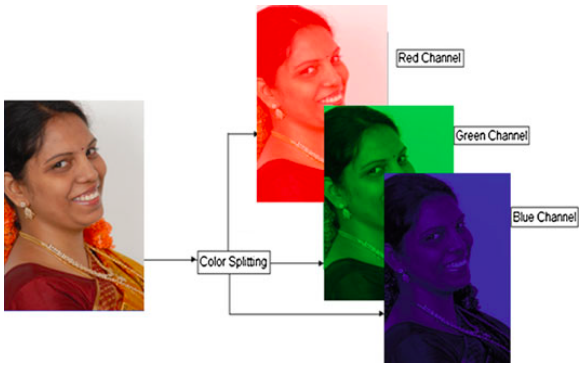


Figure 2 - Colour filter of an image which is split into three channels (International Conference on Computer and Communication Technologies et al., 2016)

colour is determined by the amount of Red, Green, and Blue in each pixel. In one pixel the factors have a range 0-255 each, this makes 256^3=16,777,216 different colour possibilities. A Colour image (Fig.2) with this format has three matrices "behind it", meaning that each pixel has 3 values (Red, Green, Blue).

1.1.2 Computer Vision

The science and technology field that works on allowing computers to see and comprehend the real world is Computer Vision, i.e. words, computer vision is a scientific discipline that encompasses methods to collect, process, evaluate and interpret images from the real world with the goal to produce information that can be grasped by a computer. Just like how humans can use their eyes and brains to comprehend the world around us, computer vision aims to reproduce the same process in order for computers to perceive and comprehend one or multiple images and act as appropriate in a given situation.

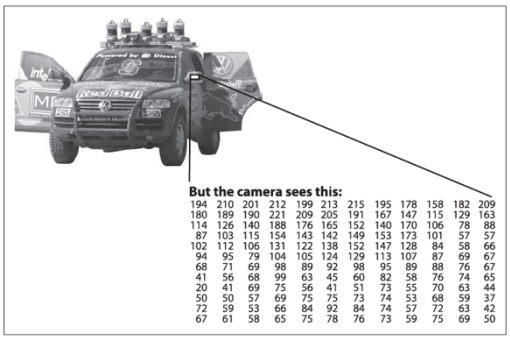


Figure 3 - To a computer, the car's side mirror is just a grid of numbers (Bradski and Kaehler, 2011)

It's difficult to get a computer to see, process and understand an image. The steps of how computer vision is achieved can be summarized as image acquisition, image processing and analysis of the image. In image acquisition, the computer takes the real world environment and turns it into data (Fig. 3), this data is now a digital image (Bradski and Kaehler, 2011). For humans this process can be something as easy as taking a picture with our phone camera or a digital camera. Now that the computer has the data it can move forward and process it. As explained above in the DIP section, a digital image goes through mathematical operations in order to achieve the desired output. Image analysis is the final step, this involves reviewing the output from the previous steps and discuss it contents. Before the face swapping process is initiated, the computer needs to be able to recognize a human face and save it as data in order to manipulate it afterwards, computer vision is vital to achieve this objective. Thankfully, computer vision has become one of the most promising research fields in recent years, mainly due to its high demand in everyday life, from unlocking your phone, tagging people in a picture, to detecting and identifying people through Closed Circuit Television CCTV cameras. This demand has resulted in increased development of computer vision tools and technologies. This is also true for one of the most popular obstacles/puzzles in computer vision, face detection (Boyko et al., 2018).

1.1.3 Face Detection

For humans, face detection comes naturally, its effortless to differentiate between the face of a human being and something that resembles a human face but isn't, for computers it's the opposite. Through face detection the computer or system can determine the coordinates and magnitude of any human face, if there's any, in an input image. Face detection is an intriguing challenge due to the human face being a natural structured object with an intricate detailed pattern disparity (Sung and Poggio, 1998). A working face detection system can contribute helpful insight in how to proceed with detection issues with other similar facial characteristics. Face detection can be a relatively complex task due to a high variability in face patterns. The variability is a result of human diversity in skin colour, facial expressions, aesthetic presentation (ex. Make-up, face tattoos, beard, moustaches, etc.). Further variability can arise from accessories (glasses, scarfs, face masks, etc.). Since human faces fall under the category of 3D objects, shifts in conditions such as lighting can improve or diminish certain facial characteristics (Sung and Poggio, 1998). As a consequence of such complications using pre-established object classification techniques, that have performed fine with

firm and pronounce objects, behave badly with human face detection (Sung and Poggio, 1998).

Face detection is an algorithm that establishes the (x, y) coordinates and size of every human face (if any) in an input image. Face detection is a compelling technique since it's the first step of more advanced computer vision systems, for example face recognition (Annan Li et al., 2012) and human mood detectors (Zenonos et al., 2016) systems. From a scholarly point of view, face detection is a compelling task due to how complex and diverse a human face can be. A strong face detection system can produce helpful information on how to further address similar issues (Sung and Poggio, 1998).

The main concerns connected to face detection are; variations in pose, change the angle of the face in relation to the camera (frontal, 45-degree, profile, upside down) and this results in some facial features, like eyes or nose, to disappear from the image. Different people may or may not have features such as beards, moustaches, and glasses and those features may differ in colour, shape and size. Facial features can change due to a person's facial expression, a person can look different depending if they are happy or sad. Some parts of the face may be blocked by other objects. Image orientation, a face in an image might become unrecognisable to a computer if it's upside down. Non-human factors like lighting and camera sensors/lens quality can change the appearance of a face (Ming-Hsuan Yang et al., 2002).

Face detection methods and approaches can be divided into four categories (Sung and Poggio, 1998) that may overlap with one another (Table 1):

Approach	Representative Works	Reference
Knowledge-based	Multiresolution Rule-based Method	(Yang and Huang, 1994)
Feature Invariant	Facial Features	(Leung et al., 1995) (Yow and Cipolla, 1996)
	Texture	(Dai and Nakano, 1996)

	Skin colour	(Jie Yang and Waibel, 1996) (Mckenna et al., 1998)
	Multiple Features	(Kjeldsen and Kender, 1996)
Template Matching	Predefined Face Templates	(Craw et al., 1992)
	Deformable Templates	(Lanitis et al., 1995)
Appearance-based Method	Eigenface	(Turk and Pentland, 1991)
	Distribution-based	(Sung and Poggio, 1998)
	Neural Network	(Rowley et al., 1998)
	Support Vector Machine (SVM)	(Osuna et al., 1997)
	Naive Bayes Classifier	(Schneiderman and Kanade, 1998)
	Hidden Markov Model (HMM)	(Leung et al., 1995)
	Information-Theoretical Approach	(Lew, 1996)

Table 1 - Categorization of Methods for Face Detection in a Single Image (Sung and Poggio, 1998)

1. Knowledge-based methods, these methods write in code human knowledge of what composes a "common" human face, they also include connection between facial features.
2. Featuring invariant approaches, these approaches aim to find the face using the core facial features that can be found in all face images while disregarding variations like lighting, pose, rotation or viewpoint. This approach includes *image-invariant methods* (Sung and Poggio, 1998), which presumes that there's an universal common connection among all face patterns, even when the conditions of the image vary. To be able to detect faces, a classifier needs to be created from a set of image invariants to

scan the target image for places where they happen, an example of this approach would be a scheme based on observed brightness invariants between different parts of the human face. The importance of this approach is that when a classifier is made, it has to be able to take into account variations among the input patterns. Meaning that the transformation does not affect the class or object that needs to be detected/identified, a rotated image of a face is still a face even if its upside. This ability is not difficult for humans, but it's extremely difficult for a computer classifier.

3. Temple matching methods, a template can be pre-established patterns that are saved and then used to "explain" what a face is or what are the separate features that form a face to the computer. In order to detect a face, the correlation between the template and the input image are calculated. Templates can be further divided into; *correlation templates (Sung and Poggio, 1998)*, this method uses predetermined face templates to discover the faces by matching the face template with the image. Since the human face is too diverse, an indefinite number of templates are needed to detect the basic human face features (eyes, eyebrows, nose, mouth, etc.). And *deformable templates*, this method differs with correlation templates in that it can deal with nonrigid facial features. Templates can be rigid (fixed), like eyes, eyebrows, nose and mouth, or can be nonrigid like the variations in the face between the eyes, eyebrows, nose and mouth. In other words, most humans possess eyes, eyebrows, nose and a mouth but every human possesses different proportions between these features. With deformable templates there is a global template frame to which configured curves and surfaces are fixed to, this fixture is elastic to permit for small deviations between facial features.

4. Appearance-based methods, in opposition to temple matching this method focus on using sets of images to teach a model all the variabilities of what a face looks like, subsequently the models are used for face detection.

1.1.4 DLib

Amongst the tools developed to work with face detection, DLib is a cross-platform open source software library containing machine learning algorithms (Korshunova et al., 2017), image processing, data mining (Junior et al., 2019) and many other tasks (Sharma et al., 2016) and is used by the public and private sector to solve problems in a wide range of domains (Fig. 4) (King, n.d.). Even though DLib is principally a C++ toolkit, it has easy-to-use python bindings ("dlib C++ Library," n.d.).

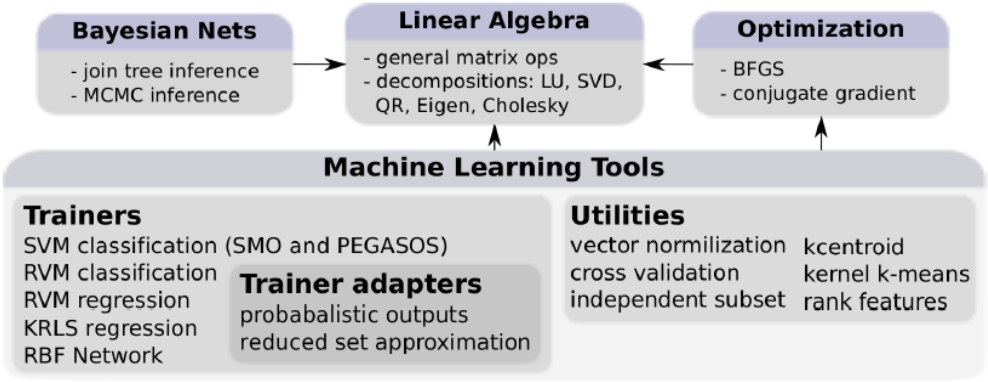


Figure 4 - Elements of DLib-ml. Arrows show dependencies between components (King, n.d.)

DLib has major features that explain why it is one of the most popular libraries used for computer vision. Contrary to other open source libraries, DLib offers thorough documentation for each of its classes and functions. When its debugging mode is used it will pick up most of the bugs caused by incorrectly calling functions or incorrectly using objects. According to the creator of DLib, Davis E. King, the documentation is astonishing because it provides many example programs and if any user finds anything that isn't in the documentation, is unclear, or is out of date you can contact him, and he'll fix it. Between the vast ML Algorithms, DLib possesses structural Support Vector Machines (SVM) tools aimed at object detection (Wan et al., 2017). This algorithm is used in combination with Histogram of Oriented Gradients (HOG) algorithms to create a powerful and robust frontal face detector (Xu and Kakadiaris, 2017).

1.1.5 The Bias/Variance Trade-off (BVT)

Before venturing into what a Support Vector Machine is, Let's become familiar with a fairly common pitfall of machine learning, the Bias/Variance Trade-off (BVT) (Annan Li et al., 2012). Briefly, the BVT shows that as the complexity of a model grows the bias decreases and variance increases, resulting in a U-shaped test error curve. A fundamental part of every single learning model is, how successful is it at making generalizations from training data? The success is evaluated by taking data from the same source and accurately classifying it.

Following this logic, one would assume that if the model used for training data is almost perfectly adapted to the data set, this will in turn, increase the model accuracy. Nevertheless, this is not the reality, an extremely adapted or fitted model will perform poorly with future data sets, because it will be unable to take into account the random noises found in future data sets, this phenomenon is called **overfitting**. At the same time, if the model does not fit the training set properly it will miss most of the trends as well as noises, this is called **underfitting**.

When a model does not have the flexibility to accurately fit the training data set, this model is unable to properly capture the true relationship between the parameters that made up the training set. This inability to capture the true relationship is called **Bias** in ML**.** A model that cannot properly capture the true relationship between the parameters has a relatively large amount of bias. On the other hand, if a model fits the relationship between parameters in the training set extremely well, this model then has little bias due to its great flexibility. In this case, if the model perfectly captures the relationship within the training set, when it needs to be ran through the testing set, the model won't be able to properly capture the trend and won't fit over the new set, in ML the difference in fits between data sets, testing and training is called **Variance.**

The dilemma becomes clear, a model can show low bias (its flexible and adapts to the relationship curve between the parameters) but it will also show high variability (it results in different sums of squares for different data sets), as a result it's hard to know how well this model will perform with future data sets. It may do extremely well, or it may be completely wrong. On the other hand, if a model has relatively high bias (it cannot capture the curve of the relationship between the parameters) but it has relatively low variance (the sum of squares are similar for different data sets), this model more often than not, will have consistently good predictions. This dilemma, or trade-off, is referred to as the Bias/Variance trade-off, this trade-off appears under a vast variety of conditions, as it displays the fundamental nature of generalizing any data that entail a mix of common and random elements (Briscoe and Feldman, 2011). This dilemma can be better visualized by plotting how complex a model is (model complexity) against its accuracy (Fig.5), one can observe that accuracy rises to a certain degree, after it hits a threshold the accuracy of the model starts decreasing the more complex the model is.

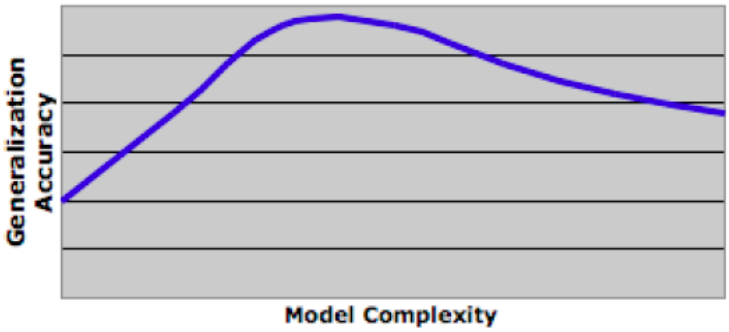


Figure 5 - A schematic illustration of generalization accuracy as a function of model complexity, illustrating the bias-variance trade-off. (Note: model complexity here is schematic and differs from the measure used in subsequent graphs.) (Briscoe and Feldman, 20111)

1.1.6 Regression Trees

An essential part of face swapping is face alignment, which has improved greatly due to its shape regression framework. This process is a chain of facial landmarks coordinates, repeated and updated many times over with a cascade of regression trees (Fig.6), taking the current shape estimation and a new shape increment are calculated in each tree until a final shape estimation is reached (Lee et al., n.d.).

Figure 6 - A selected prediction result on the 300-W dataset using cGPRT. The shape estimate is initialized and iteratively updated through a cascade of regression trees: (a) initial shape estimate, (b)–(f) shape estimates at different stages of cGPRT (Lee et al., n.d.).

The concept of what a regression tree is and why it is such an accurate, robust and efficient framework is better understood if it's broken down to its basic parts. In a decision tree each stage asks a "question" and then based on the answer, the data is classified into categories or numbers, usually the "question" can be answered with a true or false statement. The top of the tree is called the root node and the subsequent nodes are the internal nodes, at the bottom there're leaf nodes.

Regression trees and classifications are ML approaches for building prediction models from data. The model is acquired by recurrently separating data and fitting a straightforward prediction model within each separation. Consequently, a decision tree can be used to visually represent the separation. With regression trees, variables are dependent, can take continuous or ordered discrete values and prediction error can be calculated by using the difference between predicted and observed values squared (Loh, 2011).

1.1.7 Support Vector Machines (SVM)

Given a data set, when a threshold is used and it gives the largest margin to make classifications, a maximal margin classifier (MMC) is employed. MMC can be extremely useful, but what if our data set is not consistent, MMC are super sensitive to outliers in the training data set, thus they have their limitations. In order to make a threshold that is not so sensitive to outliers, misclassifications must be allowed. When misclassifications are allowed the distance between the observations and the threshold is called a soft margin. To improve upon the soft margin cross-validation is used to determine how many missed classifications and observations to allow inside of the soft margin to get the best classification. When a soft margin is utilized to determine the location of a threshold, then a soft margin classifier AKA a Support Vector Classifier is applied to classify observations, the name Support Vector classifier comes from the fact that the observations on the edge and within the soft margin are called support vectors.

Support Vector Machines show a remarkable performance when it comes to inadequate, sparse and tumultuous data (Cortes and Vapnik, 1995). If a SVM is applied for classification, they are capable of separating labelled training data sets with a hyperplane that is as distant from the data as possible. They can also work with cases where nonlinear separation is required, as they use kernels that systematically recognize a non-linear charting to a feature space, then the SVM calculates the new hyperplane to create a non-linear border (Furey et al., 2000). For SVM to properly work, the fact that some features space will be large needs to be highlighted, since the hyperplane requires the ability to generalize data points well and not every hyperplane will be able to generalize well. The concept of optimal hyperplanes addresses this issue, an **optimal hyperplane** is a linear decision function with maximal margin between the vectors of the two classes (Fig.7) (Cortes and Vapnik, 1995).

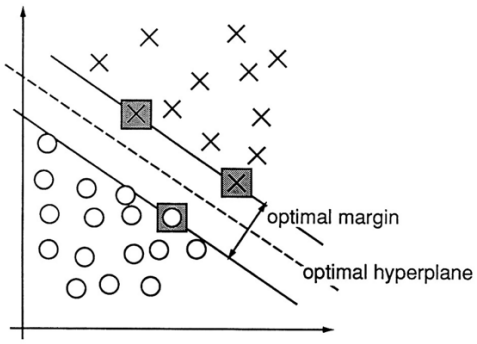


Figure 7 - An example of a separable problem in a 2-dimensional space. The support vectors, marked with grey squares, define the margin of largest separation between the two classes. (Cortes and Vapnik, 1995).

1.1.8 Histogram of Oriented Gradients (HOG)

The HOG technique is used for detecting semi-rigid objects within an image, it depends on the concept that the angle or directions of the edges and the intensity of the gradients holds really important information. With this information a program would be able to estimate the structure and aspect of an object within an image (Dalal and Triggs, 2005). This method, summarized in (Fig.8), allows a group of histograms to be calculated quickly from the image, the histograms are the magnitudes of the gradients in accordance with the direction of the gradients in a block of the image (Dalal and Triggs, 2005).

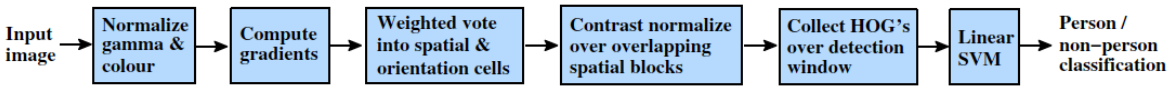


Figure 8 - An overview of our feature extraction and object detection chain. The detector window is tiled with a grid of overlapping blocks in which Histogram of Oriented Gradient feature vectors are extracted. The combined vectors are fed to a linear SVM for object/non-object classification. The detection window is scanned across the image at all positions and scales, and conventional non-maximum suppression is run on the output pyramid to detect object instances, but this paper concentrates on the feature extraction process. (Dalal and Triggs, 2005)

It makes it possible to generate a feature for each key point, and its neighbouring area, a division of small blocks occurs and for each pixel inside the block a local histogram of gradient directions is calculated. The final result from all of the histogram is the descriptor. Additionally, for each block the gradient feature vectors are calculated, these vectors are compounded to obtain the feature vector for a single image (frame), then, once all vectors from different images are chained to form one long vector, this vector is the HOG feature vector, later this vector will be applied as an input for the SVM classifier (Dalal and Triggs,

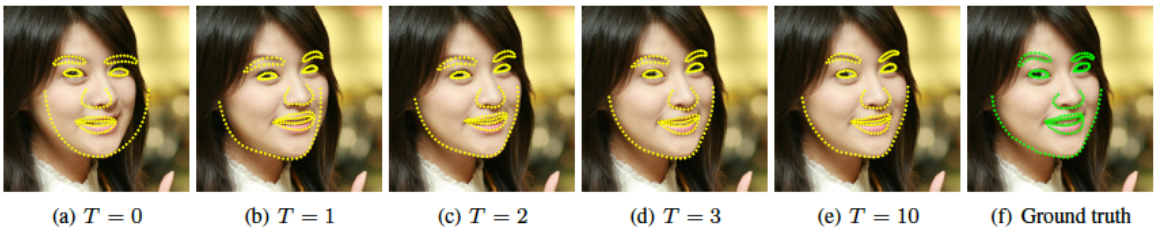
2005). In a previous face detection research, while comparing different frontal face detection approaches in video sequences (Adouani et al., 2019), they learnt that the HOG with SVM is the approach with the highest accuracy and it displayed the best performance. It finds 14.45% more faces than the next detector, the HAAR like cascade and 32.31% more faces than in the Linear Binary Pattern cascade (LBP). Furthermore, it also reached the highest detection rate at 92.68%. The HOG method also decreases false positive by at least an order of magnitude when compared to the HAAR detector (Dalal and Triggs, 2005).

1.1.9 Face Alignment

In digital images, the geometric shape of human faces can be determined thanks to face alignment technology. As previously mentioned, the most efficient way to detect a face is to know which are the facial features the programs needs to look for and consequently detect their coordinates in order to assess if there's one or more faces in the image. These facial features become the regions of interest (ROI) or landmarks of the face. Subsequently, the facial key points can be found within this ROI, they include the centres and edges of the eyes, eyebrow, nose, mouth and jaw (Longpre and Sohmshetty, n.d.). In the 2014 Kazemi and Sullivan paper (Kazemi and Sullivan, 2014) a new algorithm was introduced that was able to do face alignment in milliseconds with higher or competitive accuracy when compared to cutting edge methods on common data sets of the time. The speed improvement comes from their understanding of previous face alignment algorithms, finding their vital points and assembling them in a simplified design and then into a cascade with superior regression functions power, that are learnt via gradient boosting. In other words, they show that face alignment can be done quickly using a cascade of regression functions. Each regression function is capable of assessing the shape from an initial assessment and the level of a thin set of pixels listed in relation to the initial assessment.

They addressed the elements that commonly exist among the most successful algorithms. One element is cataloguing the pixel intensities relative to the current assessment of the shape. The facial features that need to be extracted from an image can differ greatly from one another due to deformation and nuisance factors, like changes in lighting conditions. If one is to use these features, further shape estimations become highly complicated. The predicament involves the need for reliable features to be able to precisely predict the shape, while at the same time the algorithm needs precise shape estimates in order to extract reliable

features. The Kazemi and Sullivan research uses a repetitive approach to deal with the predicament, the cascade approach. Instead of taking the shape parameters from the overall system of the image, the parameters are taken from the local system in accordance with the current estimate of the shape, subsequently, an updated vector for the shape parameters is predicted from it. This approach is repeated until the predicted shape and the pixel intensities catalogue intersect (Fig. 9).



(a) $T = 0$ (b) $T = 1$ (c) $T = 2$ (d) $T = 3$ (e) $T = 10$ (f) Ground truth

Figure 9 - Landmark estimates at different levels of the cascade initialized with the mean shape centred as the output of a basic Viola & Jones face detector. After the first level of the cascade, the error is already greatly reduced. (Kazemi and Sullivan, 2014)

To regress the position of the facial landmarks from a small subset of intensity values in an image, a group of regression trees can be used. This algorithm can manage partial or uncertain labels, it already showed to be faster at decreasing the error when compared to other works.

The DLib library includes an implementation of this trained model for facial alignment created by Davis King and has been released into the public domain (King, 2020). Inside of the DLib library the pre-trained facial landmark detection model is used to find the location of 68 (x, y)coordinates, each of these 68 points maps onto a facial feature like the eyebrows, eyes, nose, mouth and jaw (Korshunov and Marcel, 2018) which is trained on in the ibug 300-W dataset and it's intended use is a alongside DLib's HOG face detection. It assumes there will be bounding boxes from the face detector in order to do face alignment.

1.1.10 Euclidean Distance (ED)

A common obstacle in image recognition is how to establish the distance between images(Liwei Wang et al., 2005). In short the ED is the measurement of the distance between images and it's line distance between two points and for that couple of points it demonstrates the square differences between their coordinates (Hazim et al., 2016). When working with continuous data, Euclidean distance will be used as a way to measure dissimilarity, since the higher the value of the Euclidean distance the more unlike the objects are. The Euclidean

distance can be used in as many dimensions as needed, to calculate the ED between two data points, the difference in each attribute value is calculated then it's square, then sum for all values and take the square root of the total summation. After calculating the ED for multiple data points, a distance matrix can be created to observe how dissimilar the data points are, the points that are the most similar are the ones that are the closest to each other. For most object recognition algorithms, the first step is to represent images as points in an Euclidean space, making Euclidean distance a good candidate to determine the image metrics.

1.1.11 Homography (Computer Vision)

In computer vision, corresponding points are points that depict the same physical points in two images. A transformation that matches the physical points of one image to the corresponding points in another image is called a Homography. The homography matrix (**H**) is defined as a 3x3 matrix (Fig.10) and **H** can only be defined up to scale ("OpenCV: Basic concepts of the homography explained with code," n.d.). Image alignment is possible if all

$$s\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Figure 10 - The homography matrix is a 3x3 matrix ("OpenCV: Basic concepts of the homography explained with code," n.d.)

corresponding points are in the same plane in the real world, if not they won't be aligned by the homography. If there is more than one plane this means there is more than one homography, one per plane. Due to its simplicity in computer vision tasks like obstacle detection, planes are used because the transformation among a real world plane and its corresponding image plane is a homography, these connections are also true for different perspective views of a plane in different images (Vincent and Laganiere, 2001).

1.1.12 Ordinary Procrustes Analysis (OPA)

Procrustes analysis is utilized when working with a set of shapes and there's an interest in analysing their allocation and distribution. When the non-shape elements like size, position and orientation are eliminated to extract the shape variation, this is called procrustes superimposition (Klingenberg, 2015). There are two types of procrustes fit, let's focus on the first type, Ordinary Procrustes Superimposition (OPS), where the movable configuration is superimposed onto another target configuration. Under OPS (Fig.11), in a gradual process between the movable and the target configuration the variations in size, position and orientation are eliminated, by using the sum of squared distance between the landmarks points an optimal fit is discovered. The dissimilarity of two shapes can be analysed after superimposing the two shapes by translating, rotating and scaling them (Chikuse, 1999).

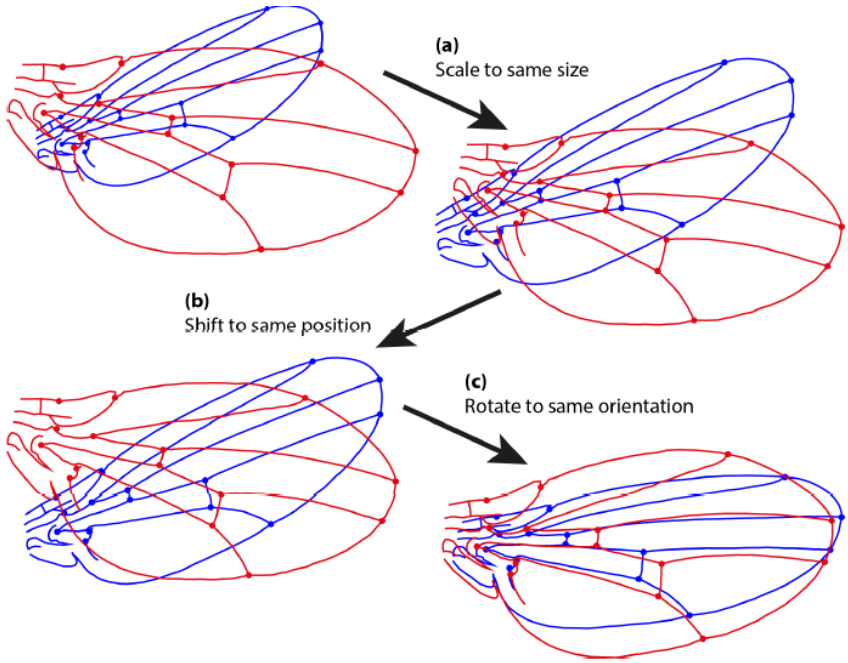


Figure 11 - Procrustes superimposition. The figure shows the three transformation steps of an ordinary Procrustes fit for two configurations of landmarks. (a) Scaling of both configurations to the same size; (b) Transposition to the same position of the centre o of gravity; (c) Rotation to the orientation that provides the minimum sum of squared distances between corresponding landmarks. (Klingenberg, 2015)

2 THE AIMS OF THE THESIS

Refactor an Image Face Swapping code, provided by supervisors, with the purpose of creating face swap code that works with videos, or a Video Face Swap, without utilizing RAM and CPU expensive Deep or Machine Learning techniques.

1. Identify obstacles the VFS code can overcome in order to create faster and more realistic looking outputs.
2. Optimize the code to overcome some of the obstacles identified after the results analysis were completed.

3 EXPERIMENTAL PART

3.1 MATERIALS AND METHODS

3.1.1 Materials

In this section the materials employed in the thesis as well as the working conditions set-up will be addressed.

Computer

The computer used for the thesis work is a MacBook Air (13-inch, 2017), which has a processor Dual-Core Intel Core i5 processor with a processor base frequency of 1.8 GHz with two independent cores that each has a 256k level 2 cache with an integrated memory controller. A 8 GB 1600 MHz DDR3 of memory and an Intel HD Graphics 6000 1536 MB video card.

Operating system

Since a MacbookAir was employed for the totality of the thesis, the code was written in the native operative system of MacOS. The versions of the OS used were macOS Mojave 10.14 to current (May 17, 2020) Catalina 10.15. When the OS X was released by Apple, it featured a fully operating Unix system, one of the most used command-line OS in the world. Other OS also had similar feautres like Linux or Ubuntu, but the difference being the OS X was a certified Unix OS ("The Register of UNIX® Certified Products," n.d.). This is important because the Unix shell allows the user to run a program without employing a specific IDE in nearly any language and many tech companies utilize Unix based systems.

Anaconda

Is an open-source distribution of R and Python programming languages that facilitates management of packages and development stages for data science and machine learning. With the aim to fully deliver every need for Python data science, tasks it includes the package management system *conda* that is used for keeping versions of the packages updated.

The importance of the *conda* package manager is the contrast to the usual *pip* package manager, when installing a package with *pip* it does so without verifying if there's a conflict with packages installed previously. Conda notifies if the installation cannot be done, it figures out how to install suitable dependencies and it examines if there's any version limitations in the current environment (including everything that is already installed). There

are multiple ways to install a package with anaconda. It can be done through the Anaconda repository, Anaconda Cloud, the user's own private repository or using the *conda install* command. There are available binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. Conda can keep track of anything installed in the environment by pip and itself. The Anaconda distribution also has a graphical user interface (GUI) for desktops called Anaconda Navigator, it allows Anaconda to manage packages, applications, environments and more without utilizing command-line commands.

Spyder

Spyder is a default application found in the Anaconda Navigator, it's an extremely powerful scientific integrated development environment (IDE) for Python, specially built by and for data analysts, engineers and scientists. It possesses powerful editing, testing, debugging and feedback tools as well as a numeric computing environment. The components of the Spyder IDE are what makes it so powerful. The editor works efficiently with a function/class browser, code review tools, code autocomplete, etc. the user can employ as many IPython consoles as necessary with the adaptability of a GUI interface.

Programming Language

Just like how humans use languages to communicate, if a human wants to work with a computer it would need an interface to translate actions (clicks) or words (command lines) into the desired outputs. A programming language is a set of instructions that the computer can understand. Each programming language has its own syntax that are adapted to their respective domains.

Python

The base code was written in python and given that this is the programming language I manage the best and that it also has excellent computer vision, machine learning and data science bindings, packages and libraries that function as the foundation of this thesis, the adapted and updated version of the code is also written in python.

Libraries

To explain what programming libraries are let’s use the example of real world libraries. One goes to the library to gain access to previously existing knowledge, this way you don't need to start from zero. Software libraries use this logic to offer already existing logic to solve problems with functions built inside the libraries.

OpenCV

This library was built to solve computer vision problems, starting from reading an image in the path folder and displaying it, to image transformation and operations like grayscaling, thresholding, blurrin, erosion, etc. this library is required for the multiple modifications that need to be applied to the images.

Numpy

Numpy is a library for research computing in Python it adds support for vectors and matrices. Consisting of higher mathematical functions to handle vectors and matrices its primary function is its N-dimensional array data structure. As opposed to Python's data structure the numpy arrays are uniformly typed, meaning that all elements of a single array must belong to the same type.

DLib

As explained in the literature review in more detail, DLib is a multi-purpose software library initially written in C++ but with excellent Python bindings. This library is mainly used in this thesis for its components with data structure, machine learning, image processing and face detection.

DLib Face Detector

DLib contains a face detector that finds the frontal faces of humans in images and creates a bounding box that is overlapped onto the face. The detector uses a Support Vector Machine (SVM) linear classifier and a Histogram of Oriented Gradients (HOG) to detect human faces.

DLib Model

DLib's facial detector has pre-trained models built by Davis King to estimate poses. The models are used to estimate poses. The model specifically built to find human faces was based in the famous Kazemi-Sullivan paper (Kazemi and Sullivan, 2014), which was trained with the ibug 300-W face landmark dataset ("i·bug - resources - Facial point annotations," n.d.). Dlib's facial detector has a pre-trained model that estimates the shape of the face through mapping the facial features by location of 68 (x, y) coordinates. The 68 coordinates index the location of the mouth, right eyebrow, left eyebrow, right eye, left eye, nose, jaw (shape_predictor_68_face_landmarks.dat) (Fig.12).

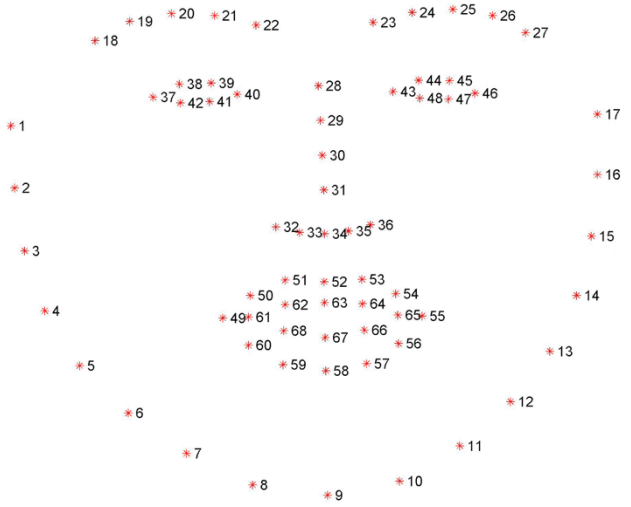


Figure 12 - The 68 points mark-up used for our annotations ("i·bug - re-sources - Facial point annotations," n.d.)

SciPy

SciPy is a free and open source library. It is composed of algorithm and mathematical tools. Scipy has modules for Fourier Transform, optimization, special functions, linear algebra, signal and image processing. Scipy helps with the Euclidean distance function among two 1-D arrays.

Time

Time is a Python module that can manage tasks related to "time" and represent it in code in different ways like objects, numbers and string. It can also be used to measure how efficient the code is and to name outputs based on the date and time they were created.

Date time

In python, dates are not a data type, a module built to manage dates and times as data types need to be imported. By importing the module and executing datetime.datetime.now() command a new object will be created, containing the current year, month, day, hour, minute, second and microsecond of when the command was executed.

Line Profiler

A module to profile the time each line in a python script takes to execute. Using a built in magic command in Spyder *(%prun*), the external package *line_profiler* is called (works just like the import command). After the run command completes a txt file with the number of

calls of a line/function, the time it took for each line/function and the total time it took the code ro run from start to finish will be printed on the txt file.

Inputs for the faces

To determine the source face, an image of president Donal Trump was used. The image is a JPEG color image, JPEG is a method of compression for digital images. For the target face a video file was used to to detect the target face. The main video “face2” is an MP4 file with 250 frames, the MP4 format is a digital multimedia container format, it's called contained because it can compress a variety of data in its files. Due to its high compression power it permits its files to be smaller in size without decreasing the quality of the file when compared to other video formats, this is why MP4 is the most common video format for web content.

3.1.2 Methods

In this section the methods used for video face swapping will be explained. For the program to work properly the following elements need to be in the folder path where the code is; An image with a face in it, this image will become the source image, for the thesis a color image of president Donald Trump was used (“trump3.jpeg”) as the source image. A video file, its frames will be used as a target image, the video can be a pre-recorded video, if no video file is available the live feed of the web camera can be used as well. The pre-trained shape predictor model for 68 face landmarks (shape_predictor_68_face_landmarks.dat) also needs to be in the folder.

Upon establishing the initial variables the face swapping process starts by calling the *faceswap* function and loading the source image and the target image into it.

The process breaks down into five steps and each step includes one or more methods:

1. Face Detection
2. Facial Landmarks Detection
3. Image Transformations
4. Color Correction
5. Image Blending

Inside the *faceswap* function, six other functions are called before returning the face swapping as an output, new variables are created to make the calculation easier to follow along.

3.1.2.1 Face Detection and Facial Landmarks Detection

The *Face Detection* and *Facial Landmarks Detection* steps happen in the *get_landmarks* function. As explained before, DLib uses an implementation of the Kazemi-Sullivan algorithm (Kazemi and Sullivan, 2014) to identify the facial landmarks from a face bounding box. The application of this implementation is actually quite simple. Within the get landmarks function, an empty array is created to store the 68 (x, y) landmarks coordinates from the source image and target image at the end of this function, resulting in two landmark matrices. Then using the face detector to locate and create a list of the rectangles that correspond to the face bounding boxes in an image. The bounding boxes are used in the feature extractor as an input for the Kazemi-Sullivan algorithm. This algorithm requires a the pre-trained shape predictor model for 68 face landmarks.

3.1.2.2 Image Transformation

The *Image Transformation* step is where the majority of the calculations happen. The source image needs to be aligned and mapped onto the target image. Now that there's two landmark matrices, each row within a matrix contains coordinates to an specific facial feature, the two faces can be aligned using procrustes analysis in the *get_M* function. Each coordinate is a point that needs to be rotated, translated and scaled to fit the landmark key points of the source image over the landmark points of the target image.

First, the input matrices need to be turned into floats, then the centroid has to be subtracted from each point, subsequently each point needs to be divided by its standard deviation. This is done in order to remove the scaling component, next rotation needs to be calculated through Singular Value Decomposition (SVD) (Cao, n.d.). SVD is a data dimensionality reduction technique that breaks down a matrix into three matrices (U, S, V) to reduce noise and redundant information. **U** is the left singular vector and it contains important non redundant information about observations, **S** is the diagonal matrix and it contains the entire information about the decomposition process and **V** is the right singular vector and it contains important non redundant information about features. The result is an affine transformation matrix (Yan et al., 2007)(Yao et al., 2001) that can be used as the input for

the warping function. In an affine transformation matrix a combination of linear transformations, such as scaling and rotation with translations occurs. During affine transformation points, straight lines, planes, parallel lines, and the ratio of distance between the points that lay on the same straight line are preserved. What affine transformations do not preserve are the angles between the lines and the distance between the points, because they might not be in the same 2D plane they originally were from, therefore the angle and distance may differ due to the change in planes.

Before getting to the warping function *warp_img*, the *get_mask* function is required in order to get the two masks needed for both images and their landmark matrices, as well as to establish what parts of both images are going to be used in the last step. An area with value 1 (White) matches the sections where the source image should be displayed, and areas on black (Value 0) match the areas where the target image should be displayed. The area where the values fall between 0 and 1 are where the source and target image will be mixed. This is done by drawing two convex polygons (in white), one encircling the eyes and the other the mouth and nose. The edges of this mask are blurred towards the outside by 11 pixels, this is done to prevent the mask border from showing. This is done for both images in the *warp_img* function, the affine matrix that was obtained in the procrustes analysis function to align both images.

3.1.2.3 Colour Correction

In the *Colour Correction* step the color difference between the two images is corrected. Because both images are not only from different people but were also taken under different conditions (skin color, light, facial hair, etc.), the source image conditions need to match the target image conditions, otherwise the result will create a disconnect in the borders of the superimposed area. This is done in the *correct_color* function by diving source image and a gaussian blur of the source image and multiplying the result by the gaussian blur of the target image, this is a mathematical color balancing method called scaling monitor R, G and B (Viggiano, 2004), but rather than having a single scale factor for all the image, localised scale factor is assigned to each pixel. This method covers the differences in light conditions between the two images to a certain degree. A good example of this is how in the source picture (Donald Trump) the light is coming from cameras flashes or another light source in front of Donal Trump, while in the video that is the target image (my face), the light comes from the window in the back and a light on top of my face, the color correction fixes the

light condition in the source image so it also looks like the light is coming from behind and above so it matches the target image. For this method a suitable kernel size is key, if the size is too small the facial features from the target image will show up, even when they are underneath the source image features, too big and discoloration happens.

3.1.2.4 Image Blending

In the final step, *Image Blending*, the two masks are combined by element-wise maximum in the *mask_img* function. This guarantees that the features of the target face are covered with the features of the source image. Then it applies alpha blending to overlay the foreground image (source image) mask, on top of the background image mask (target image).

For analysis a line profiler already installed in Spyder was utilized to get the time it takes the code to run and how many times each function is called.

3.2 RESULTS

The results are going to be divided in stages since the refactored code had to be tested with every new iteration of the code.

3.2.1 First result

During the first stage of experiments, the newly refactored code was not returning any outputs, only errors. Meaning that the code couldn't run from start to end. After debugging line by line, the error was found to be that when the video frame didn't have a face (no person is present in the frame or the face is obstructed) the *get_landmarks* function returned a None type data when it was supposed to return a matrix with the (x, y) coordinates of the facial landmarks key points. The None type output cannot be used in further functions since a None type data cannot be used for calculations. To fix this error a *try-except loop* was used inside the *while loop* in order to try run the *faceswap* function and if there was any problem to run the *exception* part of the code and print the exception to know what the problem was. Additionally, a *pass* command was used inside the *get_landmarks* functions so it returned an empty matrix instead of a None type data.

3.2.2 Second result

In the second stage of experiments, the target image was a live feed from my laptop webcam and the output was shown using an *imshow* command to open up a new window to see the live results of the code. Using the webcam live feed as the target image was replaced with a pre-recorded MP4 video due to an error with Macs where the *imshow* window cannot be closed using keyboard commands, the code just freezes. For this particular error I tried many of the suggested solutions found in programming forums like Stack Overflow or Github but nothing worked, for the sake of simplicity and since I needed to record the outputs, the target image was changed to a pre-recorded video.

3.2.3 Third result

In the third stage of experiments, satisfactory results were obtained. Video Face Swap was performed in all frames of an MP4 video. The problem now was that the code took between

7-11 minutes (Fig.13) to produce a 25 seconds face swap video from an original 8 sec video with 250 frames. The code was analysed with the aim to discover why the code took so long.

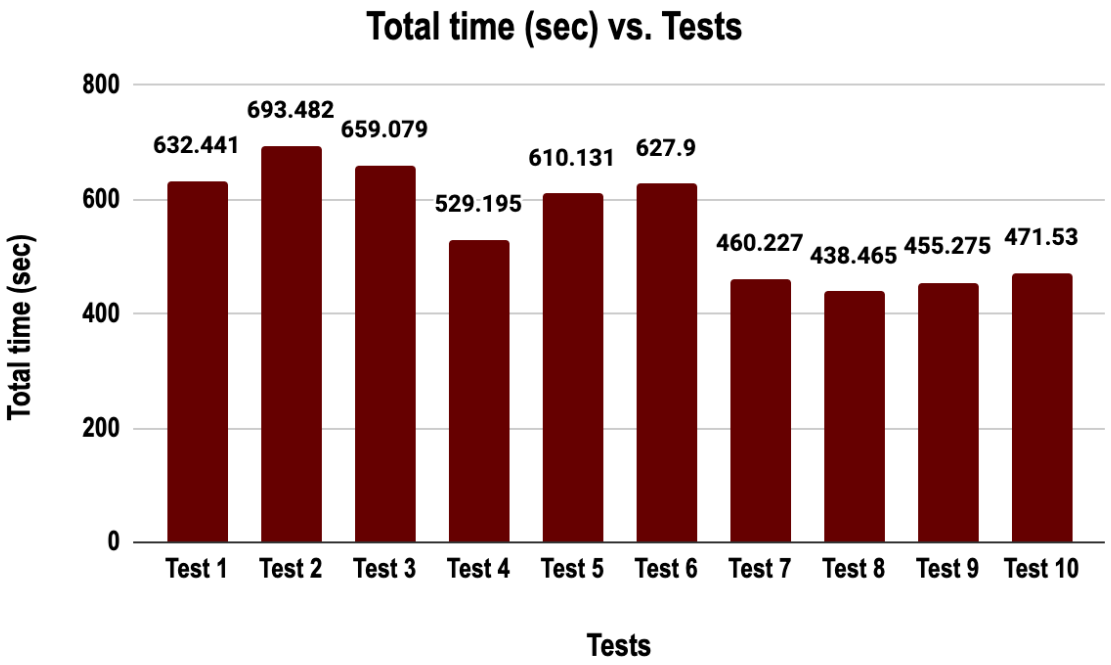


Figure 13 - Total time the Python script took calculating VFS for all 250 frames, the code was tested 10 times to avoid errors in the measurements.

After running a line profiler in Spyder to get the execution time for each function (Fig.14) and how many times each function was called (Fig.15), it was discovered that the landmarks for the source image were getting calculated all over again for each new video frame, this is counterproductive since only the video frames have different and new landmarks on each frame, while the Donald Trump image landmarks remains the same throughout the process. The code was changed to ensure that the source image landmarks were calculated once at the beginning of the code and that these values are saved and used alongside each new target image (frame) landmarks.

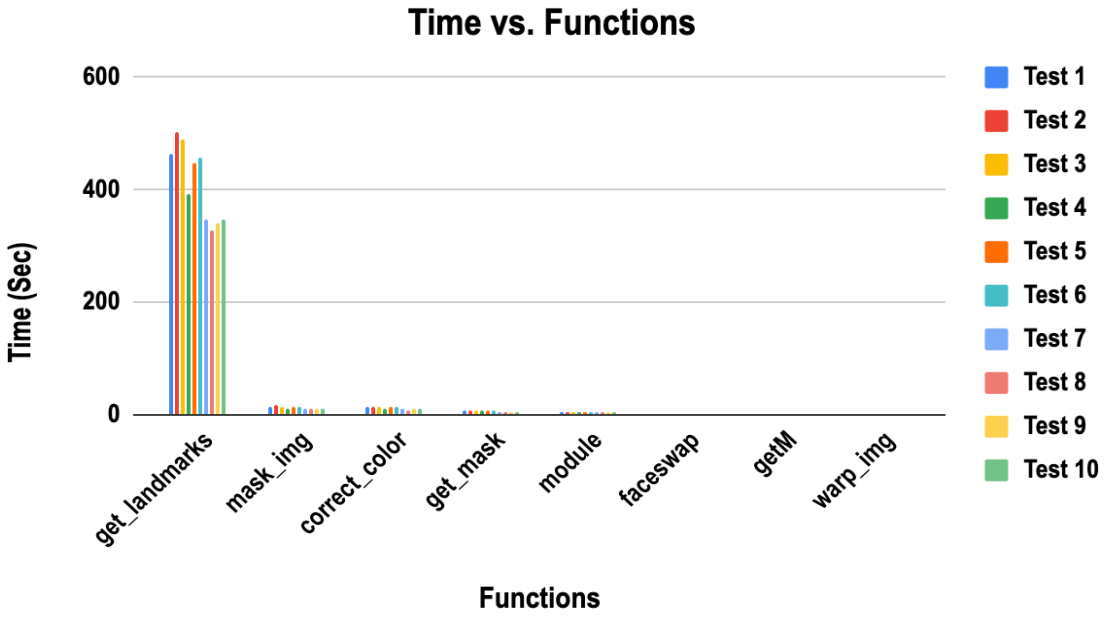


Figure 14 - The total execution time of each function, VFS was calculated for all 250 frames, the code was tested 10 times to avoid errors in the measurements.

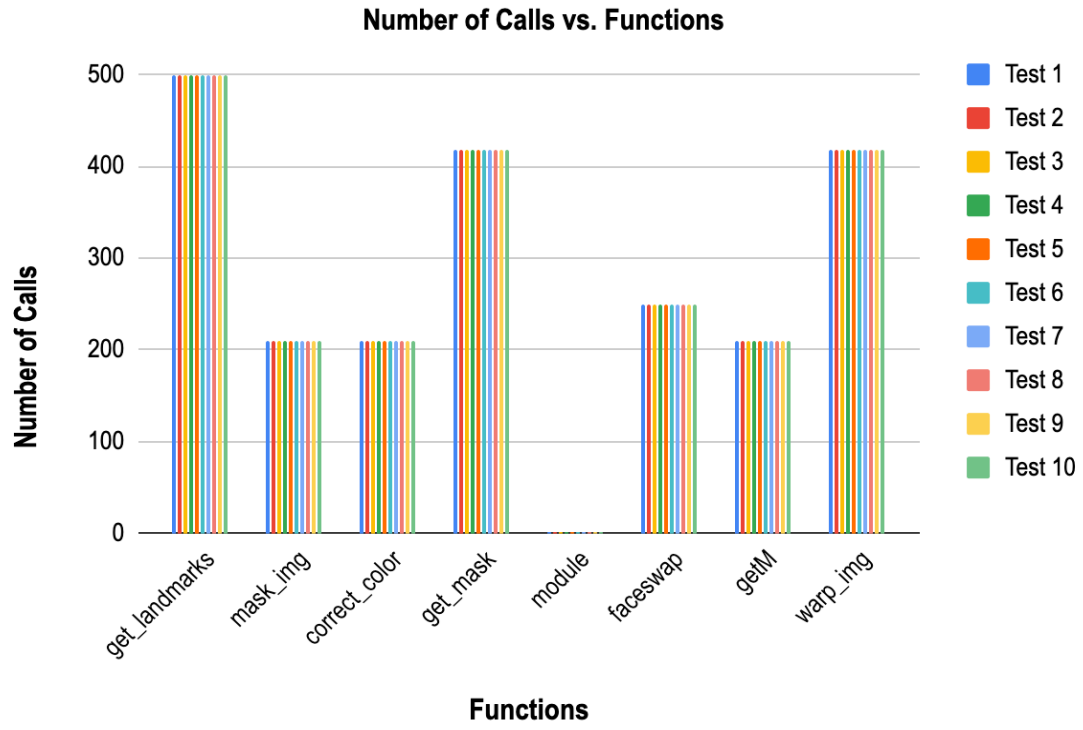


Figure 15 - The number of times a function was called from start to end in the Python code, VFS was calculated for all 250 frames, the code was tested 10 times to avoid errors in the measurements.

3.2.4 Fourth result

After the code was optimize, the output was an identical video to the one obtained in stage 3, however, the difference was that now it took between 3-6 minutes to calculate (Fig.16) face swap in all 250 video frames.

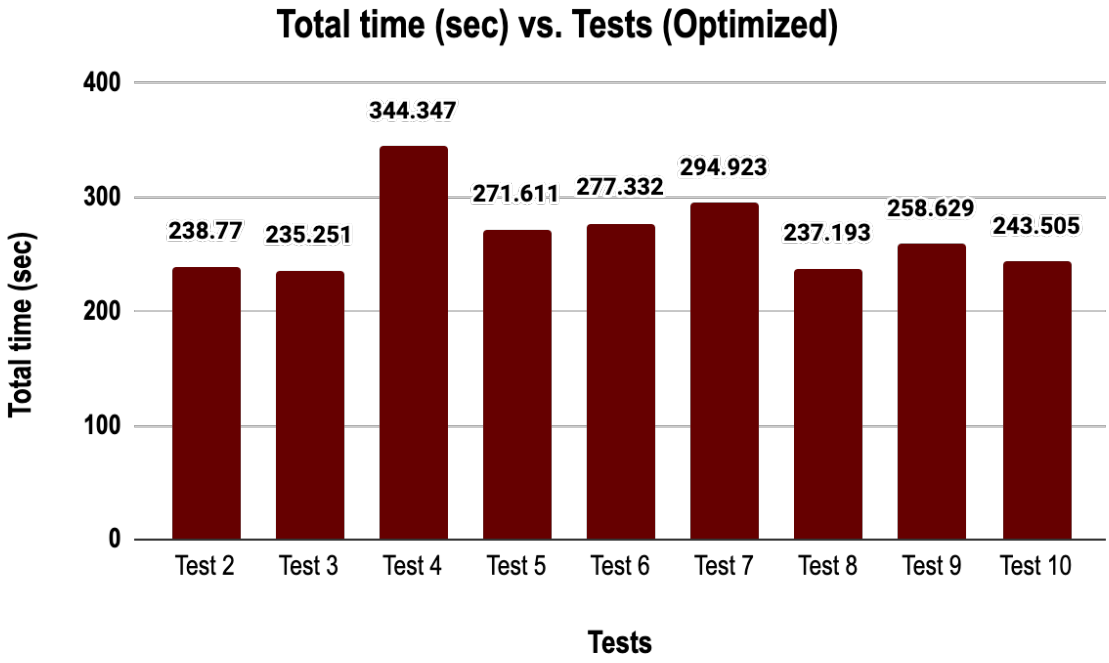


Figure 16 - Total time the Python script took calculating VFS for all 250 frames when the source image landmarks were calculated once at the begging of the code, the code was tested 10 times to avoid errors in the measurements.

3.2.5 Fifth result

The next optimization was to reduce the time it took to create an output video. The strategy employed was to perform VFS every other frame instead of performing it in all frames in the video. First, this was tested by calculating VFS every 2 and 3 frames. When VFS was calculated every 2 frames, the code took 1-3 minutes to process half of the frames (Fig.17). While for every 3 frames, the code took between 1-2 minutes to calculate for a third of the frames. The frames where VFS was not calculated were discarded and not used to create the output video (Fig.18). This resulted in creating a 8 secs video as an output for VFS every 3 frames, which matches the length of the original input video for the target image.

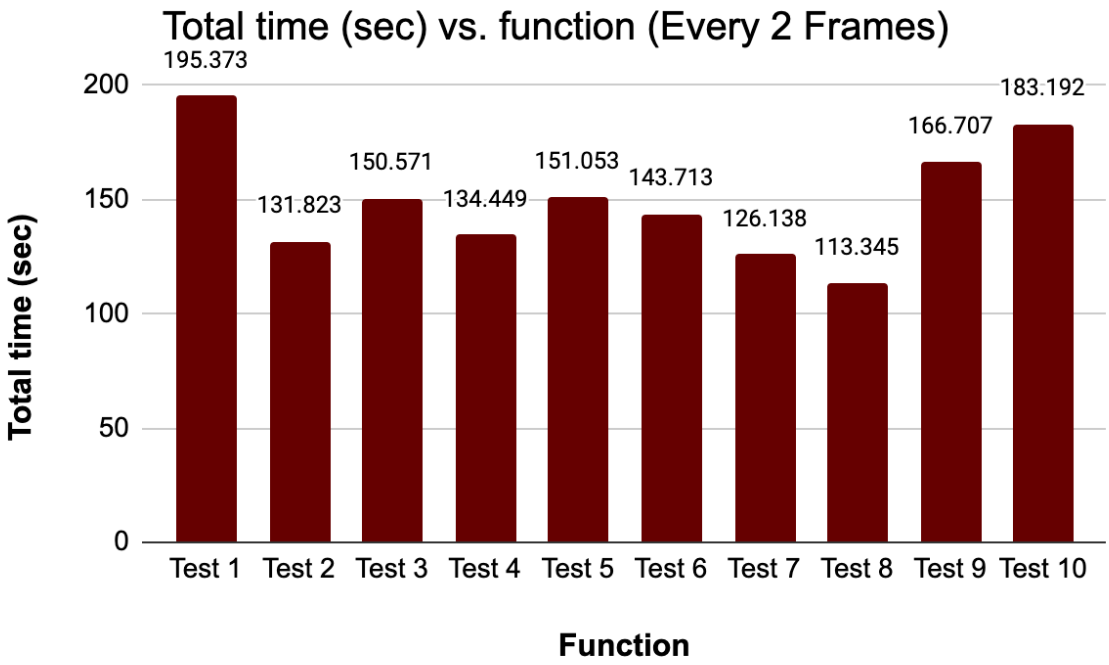


Figure 17 - Total time the Python script took calculating VFS for half the frames, the code was test-ed 10 times to avoid errors in the measurements.

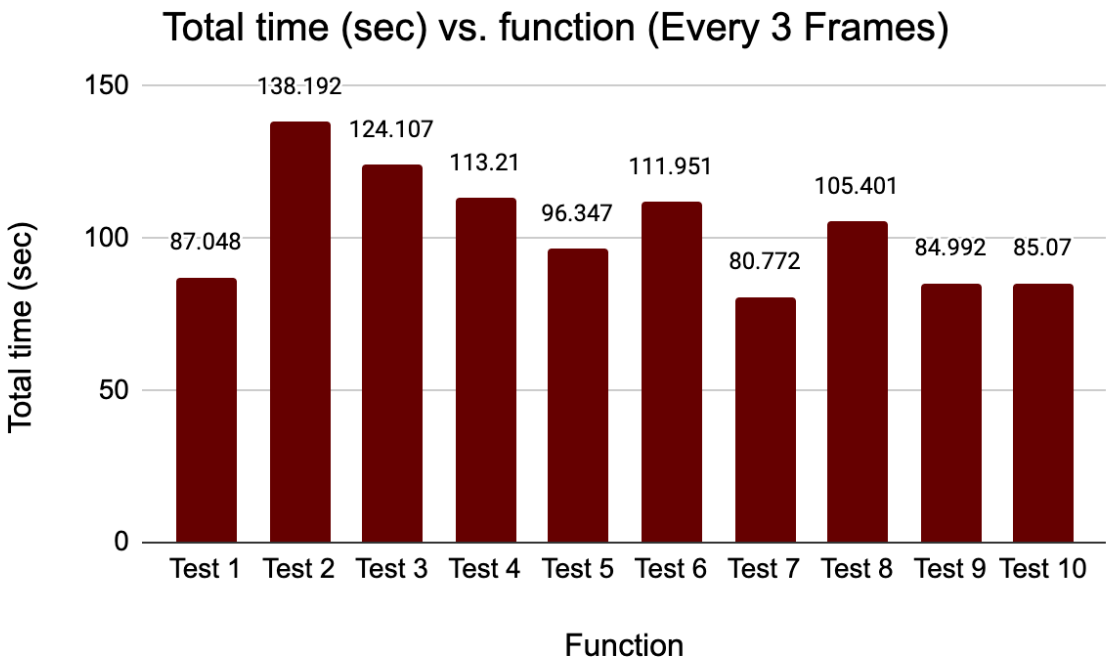


Figure 18 - Total time the Python script took calculating VFS for a third of the frames, the code was tested 10 times to avoid errors in the measurements.

3.3 DISCUSSION

This section will be focused on discussing the background research, methodology and results involved with this thesis. While discussing the results the thesis questions are gonna be answered, obstacles and problems identified, and future research and work is proposed.

Thesis Questions

The following are the theis questions that are going to be answered. In order to do this, the research background and results will be analyzed.

1. Is it possible to refactor the Image Face Swapping code to work on videos? What are the main differences between the codes?
2. How does this VFS compare against other face swapping (Image and videos) approaches?

Answering Q1: Video Face Swapping code

As mentioned before, the original code for Image Face Swapping, can be broken down to 5 steps; Face Detection, Facial Landmarks Detection, Image Transformation, Colour Correction and Image Blending. These steps have been explained in more detail in the methods section of the thesis. The whole process can be summarized as getting a mask from a source image that covers the Region Of Interest (ROI) of the face (eyebrows, eyes, nose, mouth and jaw), then that mask is aligned, scaled and translated onto the mask face region of the target image, finally color correction is performed such that the color condition of the source face matches the color of the target face and then two mask regions are blended together and a face swap is obtained.

When a video is used as a target image, the core idea is to grab each video frame and treat it as a single image, then perform a similar process to the one used in Image Face Swapping. Then the process is repeated for every frame in the video, or as the results showed, if the input video is a MP4 video the VFS calculations can be done every 3 frames and still create realistic outputs while at the same time reducing the calculation time and length of the output video.

The approach in the VFS changes a bit different in comparison to the original Image Face Swapping code. The IFS iterates once, since it only calculates the facial landmarks once for each image, while for the calculation process for VFS code can be extended a lot depending on the quality and length of the input video, if the quality of the video is good, VFS calculations can be done every other frame and the output will still look realistic, if the quality of the video is poor, the output which is made by stitching together the face swapped images, it will be apparent that some frames are missing and the face swap illusion will be lost. Unfortunately, due to technical problems while testing VFS with live feed from the webcam, I do not have concrete numbers when it comes to the processing time using live videos.

As shown in the results another key difference is that for every new video frame a new facial landmark matrix needs to be calculated, this is one of two key causes for delay in the processing time. The second key cause is the warping calculation, when the mask as well as the source image itself has to warp onto the shape of the target image using affine matrix transformation. This calculation has to be done twice for every frame where VFS is done, in other words if in 50 frames VFS is going to be dcalculate the warping function will be executed 100 times. However, it needs to be pointed out that even when the warping function is executed twice, the time it consumes is nothing when compared to the time used in the *get_landmarks* function.

More differences are discovered in the results of the VFS code, since it’s a dynamic process, some distortions occur that take away from the face swapping illusion. Examples are highlighted in the (Fig.19), where one can see some parts of the source image mask are out-of-bounds front the target image mask region resulting in an eye or eyebrow to be floating outside the face boundary when the face turns to the sides. Another example is how the blurring has to be calculated carefully, if the kernel size is too small the facial features of the target image will show through the source image mask but if it's too big the facial features of the source image will stray outside of the face boundary.

Figure 19 - Distortions in output video, highlighted in red, the source image mask is out of boundaries while the face is turned to the sides, highlighted in green are the blurring kernel size problem.

Conclusion:

It's possible to refactor the IFS code to work with video as long as the necessary requirements are met. The first requirement would be to choose carefully the inputs for both the target and source image, it's better to employ a high resolution image with a high quality video. If the source image has a low resolution the output face might look pixelated due to beign stretched to fit over the target face. Next, make sure that the facial landmarks are calculated accordingly, once at the beginning of the code if the source image input is an image and one time for each new image if the target image is a video. This will greatly impact the overall performance of the program. Subsequently, the code was to work even when there are no faces in the video, or the face is obstructed by turning the face to the sides or blocking it with an object or hand. Finally, in order to find the best rate at which VFS calculations need to be done, multiple code runs need to be performed, this rate will change depending on the length and quality of the video.

Answering Q2: Face Swapping Approaches Comparison

Current face swapping programs can be separated into three categories (Chen et al., 2019); replacement-based, model-based and learning-based. The face swapping used and refactored in this thesis fall under the first category, replacement based. Under this method a face region (mask) in the target image is replaced with the face region of a source image and implementing image processing approaches to mend any distortion or difference between the source

and target image. A disadvantage of this method is that it is unable to retain some facial characteristics from the target face since the idea is to overlay the mask of the source image on top of the mask region of the target face. On the other hand, some advantages are the streamline algorithms and functions as well as the initial set up being minimal after the re-factoring was done. This code doesn't employ more advanced Machine or Deep learning, but it can still deliver realistic results. It relies on common transformations to scale, rotate and translate the two landmark matrices, and while these methods sound quite complex their implementation is straightforward when both the source and target are images.

The second category, model-based face swapping a 2D or 3D model is made in order to stand in for a human face, the specifications and features of the model are adapted from the target image. Then a reconstruction based on the features of the source image is performed onto the model. Some example algorithms (Hong-Xia Wang et al., 2008) of this category require manual set up and a pre-made 3D model, which would require a certain amount of faces for model training, other algorithms (Lin et al., 2012) tried fixing these shortcomings by constructing 3D models based on frontal face images but the resulting 3D model does not truly reflect the original face parameters and calculations take too much time out of the entire process.

In the final category, learning-based face swapping approaches use a target images to train a neural network that already has the information of the source image. This way a convolution neural network model can swap the face of the target image with the face from the source image while retaining the light conditions, facial expression and posture of the target image (Korshunova et al., 2017). Clear advantages of this methods are the realistic output created by the model, on the other hand the the model needs a lot of training and testing while using a lot of computational power, additionally the model will only work on a single target individual at a time.

Conclusion:

While other face swapping programs have no problem using a video as either a source or target, they mainly utilize machine learning and 3D modelling approaches to create realistic

outputs, and while the result might fool the human eye more than the replacement-based model I use in the thesis, they require a great deal of manual set up and computations to run the face swapping network in a single target individual, while the replacement-base methods can be tested multiple times with multiple target individuals and still get satisfactory results.

Future Work

This part will propose future works that could help solve the current obstacles and limitations of the code as well as other potential uses for the code.

Perfil Face Detector

One of the shortcoming of the code was that when the face was turned to the sides it either did not perform VFS or it showed the source mask out of bounds from the target image and it breaks the face swapping illusion for those brief moments. There can be multiple ways to possibly solve this problem. Regardless of which path one chooses to go, a good tool to have in any path forward is to train a facial landmarks keypoint model similar to the 68-points model used in the thesis. That model was trained with frontal faces and it shows problems assigning the coordinates when the face is turned to the sides (Fig.20). In order to create a perfil face detector and to assign the (x, y) coordinates properly to the facial landmarks when the face is turned to the sides, DLib library offers the option to train your own object detection model, therefore it is possible to train a model for perfil faces using DLib's library and Python.

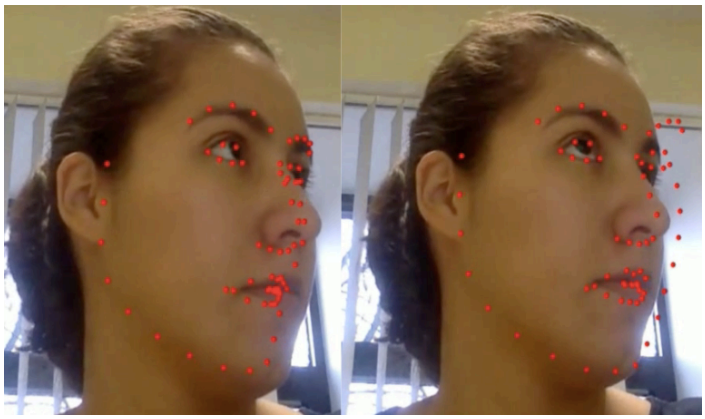


Figure 20 - Current facial landmarks key points for the 68 points model while the face is turned to the sides.

Retaining Target Image Pose

Continuing the problem stated above, when the face is turned to the sides the source image mask is outside of the target face boundary. Considering the facial landmarks keypoint assignment are for (x, y) coordinates the Z coordinates are missing, consequently the code is missing the information to know in which direction (front and back) the target moved and creating a 3D model from the 2D information is difficult. An approach to solve this could be to calculated the distance between the jaw keypoints and the eyes, nose and mouth keypoints, this is better illustrated in (Fig.21) where it shows that when the face is turned to the side the distance between the two sets of keypoints is smaller than when the face is looking toward the camera. With this information a new mask can be created to fit the face when it is turned to the sides.

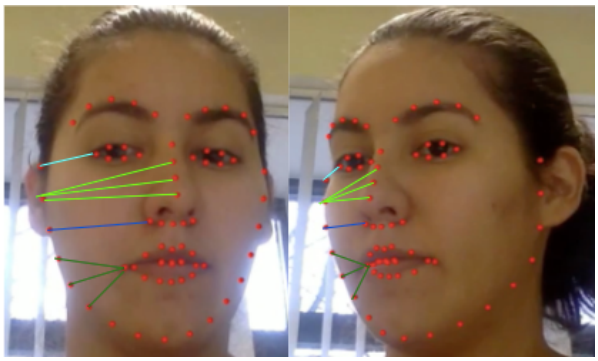


Figure 21 - Difference in distance between the jaw key points and the eyes, nose and mouth key points when the face is turned to the side and when is turned toward the camera.

Face Swapping with Facial Expressions

Since this face swapping approach is replacement-based, when the mask is overlaying the target face it becomes impossible to show any facial expression, like any mouth movements that the target face is making. A possible approach is to create a separate dedicated mask for each facial feature so it moves accordingly to the expression of the target, this might create a face swapping that looks like a picasso painting, but the information obtained in this approach can be helpful to further improve the results in the future.

SUMMARY

In summary the aims of the thesis were achieved, the Image Face Swapping code was successfully refactored to work with videos and without utilizing ML approaches like Convolutional Neural Networks or Generative Adversarial Networks. After performing a satisfactory VFS, the results were analysed in order to find sections to improve within the code. The result and performance information are summarized in the following Table 2:

Name of Code	Total time (sec) (average of 10 tests)	Number of frames	Length of video (Sec)		Number of Calculations		
			input	output	Frames per VFS	Target image landmarks	Source image landmarks
vsf_main.py	557.7725	250	8	25	1	250	250
vsf_main_updated.py	265.8103	250	8	25	1	250	1
vsf_main_experiments.py	149.6364	250	8	13	2	250	1
vsf_main_experiments.py	102.709	250	8	8	3	250	1

Table 2 - Summary of results

The problems and obstacles found in the code were:

1. Source image facial landmarks were calculated 250 times.
2. Calculating VFS for all frames was slowing the process to a degree that the time period for processing 250 frames was 69 times larger than the length of the input video.
3. Some distortions appeared when the face was turned to the sides; some sections of the source image mask were outside the target image face boundary, at a certain angle when the face is turned it looks like the target person has two noses.
4. The blur kernel size was too small and in certain frames the eye movement of the target image was visible.
5. A computer with better memory capabilities will improve the performance of the code.

Upon identifying these obstacles, the code was optimized through calculating the source image face landmarks once at the beginning of the code and saved to be used alongside each

new target image facial landmarks. Next VFS was performed for half and third of the video frames instead of calculating VFS in all frames of the video. Finally the blur kernel size was increased to prevent the target image eye movements to show through the source image mask.

REFERENCES

Adouani, A., Ben Henia, W.M., Lachiri, Z., 2019. Comparison of Haar-like, HOG and LBP approaches for face detection in video sequences, in: 2019 16th International Multi-Conference on Systems, Signals Devices (SSD). Presented at the 2019 16th International Multi-Conference on Systems, Signals Devices (SSD), pp. 266–271. https://doi.org/10.1109/SSD.2019.8893214

Annan Li, Shiguang Shan, Wen Gao, 2012. Coupled Bias–Variance Tradeoff for Cross-Pose Face Recognition. IEEE Trans. Image Process. 21, 305–315. https://doi.org/10.1109/TIP.2011.2160957

Boyko, N., Basystiuk, O., Shakhovska, N., 2018. Performance Evaluation and Comparison of Software for Face Recognition, Based on Dlib and Opencv Library, in: 2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP). Presented at the 2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP), pp. 478–482. https://doi.org/10.1109/DSMP.2018.8478556

Bradski, G.R., Kaehler, A., 2011. Learning OpenCV: computer vision with the OpenCV library, 1. ed., [Nachdr.]. ed, Software that sees. O'Reilly, Beijing.

Briscoe, E., Feldman, J., 2011. Conceptual complexity and the bias/variance tradeoff. Cognition 118, 2–16. https://doi.org/10.1016/j.cognition.2010.10.004

Cao, L., n.d. Singular Value Decomposition Applied To Digital Image Processing 15.

Chen, D., Chen, Q., Wu, J., Yu, X., Jia, T., 2019. Face Swapping: Realistic Image Synthesis Based on Facial Landmarks Alignment. Math. Probl. Eng. 2019, 1–11. https://doi.org/10.1155/2019/8902701

Chikuse, Y., 1999. Procrustes analysis on some special manifolds. Commun. Stat. - Theory Methods 28, 885–903. https://doi.org/10.1080/03610929908832332

Cortes, C., Vapnik, V., 1995. Support-vector networks. Mach. Learn. 20, 273–297. https://doi.org/10.1007/BF00994018

Craw, I., Tock, D., Bennett, A., 1992. Finding face features, in: Sandini, G. (Ed.), Computer Vision — ECCV'92, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 92–96. https://doi.org/10.1007/3-540-55426-2_12

Dai, Y., Nakano, Y., 1996. Face-texture model based on SGLD and its application in face detection in a color scene. Pattern Recognit. 29, 1007–1017. https://doi.org/10.1016/0031-3203(95)00139-5

Dalal, N., Triggs, B., 2005. Histograms of oriented gradients for human detection, in: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). Presented at the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), pp. 886–893 vol. 1. https://doi.org/10.1109/CVPR.2005.177

Dale, K., Sunkavalli, K., Johnson, M.K., Vlasic, D., Matusik, W., Pfister, H., 2011. Video face replacement.

deepfakes, 2020. deepfakes/faceswap.

dlib C++ Library [WWW Document], n.d. URL http://dlib.net/ (accessed 4.8.20).

Furey, T.S., Cristianini, N., Duffy, N., Bednarski, D.W., Schummer, M., Haussler, D., 2000. Support vector machine classification and validation of cancer tissue samples using microarray expression data. Bioinformatics 16, 906–914. https://doi.org/10.1093/bioinformatics/16.10.906

Garrido, P., Valgaerts, L., Rehmsen, O., Thormaehlen, T., Perez, P., Theobalt, C., 2014. Automatic Face Reenactment, in: 2014 IEEE Conference on Computer Vision and Pattern Recognition. Presented at the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Columbus, OH, USA, pp. 4217–4224. https://doi.org/10.1109/CVPR.2014.537

Hazim, N., Sameer, S., Esam, W., Abdul, M., 2016. Face Detection and Recognition Using Viola-Jones with PCA-LDA and Square Euclidean Distance. Int. J. Adv. Comput. Sci. Appl. 7. https://doi.org/10.14569/IJACSA.2016.070550

Hjorth, L., 2012. Studying Mobile Media: Cultural Technologies, Mobile Communication, and the iPhone, 1st ed. Routledge. https://doi.org/10.4324/9780203127711

Hong-Xia Wang, Chunhong Pan, Haifeng Gong, Huai-Yu Wu, 2008. Facial image composition based on active appearance model, in: 2008 IEEE International Conference on Acoustics, Speech and Signal Processing. Presented at the 2008 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 893–896. https://doi.org/10.1109/ICASSP.2008.4517754

i·bug - resources - Facial point annotations [WWW Document], n.d. URL https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/ (accessed 4.17.20).

International Conference on Computer and Communication Technologies, Satapathy, S.C., Raju, K.S., Mandal, J.K., Bhateja, V., 2016. Proceedings of the second International Conference on Computer and Communication Technologies: IC3T 2015. Volume 3 Volume 3.

Jie Yang, Waibel, A., 1996. A real-time face tracker, in: Proceedings Third IEEE Workshop on Applications of Computer Vision. WACV'96. Presented at the Proceedings Third IEEE Workshop on Applications of Computer Vision. WACV'96, pp. 142–147. https://doi.org/10.1109/ACV.1996.572043

Junior, J.C.S.J., Ozcinar, C., Marjanovic, M., Baró, X., Anbarjafari, G., Escalera, S., 2019. On the effect of age perception biases for real age regression. ArXiv190207653 Cs.

Kazemi, V., Sullivan, J., 2014. One millisecond face alignment with an ensemble of regression trees, in: 2014 IEEE Conference on Computer Vision and Pattern Recognition. Presented at the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Columbus, OH, pp. 1867–1874. https://doi.org/10.1109/CVPR.2014.241

Kietzmann, J., Lee, L.W., McCarthy, I.P., Kietzmann, T.C., 2020. Deepfakes: Trick or treat? Bus. Horiz., ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING 63, 135–146. https://doi.org/10.1016/j.bushor.2019.11.006

King, D.E., 2020. davisking/dlib-models.

King, D.E., n.d. Dlib-ml: A Machine Learning Toolkit 4.

Kjeldsen, R., Kender, J., 1996. Finding skin in color images, in: Proceedings of the Second International Conference on Automatic Face and Gesture Recognition. Presented at the Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, pp. 312–317. https://doi.org/10.1109/AFGR.1996.557283

Klingenberg, C.P., 2015. Analyzing Fluctuating Asymmetry with Geometric Morphometrics: Concepts, Methods, and Applications. Symmetry 7, 843–934. https://doi.org/10.3390/sym7020843

Korshunov, P., Marcel, S., 2018. Speaker Inconsistency Detection in Tampered Video. pp. 2375–2379. https://doi.org/10.23919/EUSIPCO.2018.8553270

Korshunova, I., Shi, W., Dambre, J., Theis, L., 2017. Fast Face-swap Using Convolutional Neural Networks. ArXiv161109577 Cs.

Lanitis, A., Taylor, C.J., Cootes, T.F., 1995. Automatic face identification system using flexible appearance models. Image Vis. Comput. 97–12.

Lee, D., Park, H., Yoo, C.D., n.d. Face Alignment using Cascade Gaussian Process Regression Trees.

Leung, T.K., Burl, M.C., Perona, P., 1995. Finding faces in cluttered scenes using random labeled graph matching, in: Proceedings of IEEE International Conference on Computer Vision. Presented at the Proceedings of IEEE International Conference on Computer Vision, pp. 637–644. https://doi.org/10.1109/ICCV.1995.466878

Lew, M.S., 1996. Information theoretic view-based and modular face detection, in: Proceedings of the Second International Conference on Automatic Face and Gesture Recognition. Presented at the Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, pp. 198–203. https://doi.org/10.1109/AFGR.1996.557264

Lin, Y., Wang, S., Lin, Q., Tang, F., 2012. Face Swapping under Large Pose Variations: A 3D Model Based Approach, in: 2012 IEEE International Conference on Multimedia and Expo. Presented at the 2012 IEEE International Conference on Multimedia and Expo, pp. 333–338. https://doi.org/10.1109/ICME.2012.26

Liwei Wang, Yan Zhang, Jufu Feng, 2005. On the Euclidean distance of images. IEEE Trans. Pattern Anal. Mach. Intell. 27, 1334–1339. https://doi.org/10.1109/TPAMI.2005.165

Loh, W.-Y., 2011. Classification and regression trees. WIREs Data Min. Knowl. Discov. 1, 14–23. https://doi.org/10.1002/widm.8

Longpre, S., Sohmshetty, A., n.d. Facial Keypoint Detection 8.

Mahajan, S., Chen, L.-J., Tsai, T.-C., 2017. SwapItUp: A Face Swap Application for Privacy Protection, in: 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA). Presented at the 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), pp. 46–50. https://doi.org/10.1109/AINA.2017.53

Mckenna, S.J., Gong, S., Raja, Y., 1998. MODELLING FACIAL COLOUR AND IDENTITY WITH GAUSSIAN MIXTURES. Pattern Recognit. 31, 1883–1892. https://doi.org/10.1016/S0031-3203(98)00066-1

Ming-Hsuan Yang, Kriegman, D.J., Ahuja, N., 2002. Detecting faces in images: a survey. IEEE Trans. Pattern Anal. Mach. Intell. 24, 34–58. https://doi.org/10.1109/34.982883

OpenCV: Basic concepts of the homography explained with code [WWW Document], n.d. URL https://docs.opencv.org/master/d9/dab/tutorial_homography.html (accessed 4.21.20).

Osuna, E., Freund, R., Girosit, F., 1997. Training support vector machines: an application to face detection, in: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Presented at the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Comput. Soc, San Juan, Puerto Rico, pp. 130–136. https://doi.org/10.1109/CVPR.1997.609310

Rossner, M., Yamada, K.M., 2004. What's in a picture? The temptation of image manipulation. J. Cell Biol. 166, 11–15. https://doi.org/10.1083/jcb.200406019

Rowley, H.A., Baluja, S., Kanade, T., 1998. Neural network-based face detection. IEEE Trans. Pattern Anal. Mach. Intell. 20, 23–38. https://doi.org/10.1109/34.655647

Schneiderman, H., Kanade, T., 1998. Probabilistic modeling of local appearance and spatial relationships for object recognition, in: Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231). Presented at the Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231), pp. 45–51. https://doi.org/10.1109/CVPR.1998.698586

Sharma, S., Shanmugasundaram, K., Ramasamy, S.K., 2016. FAREC — CNN based efficient face recognition technique using Dlib, in: 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT). Presented at the 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), pp. 192–195. https://doi.org/10.1109/ICACCCT.2016.7831628

Story, D., n.d. From Darkroom to Desktop—How Photoshop Came to Light 3.

Sung, K.-K., Poggio, T., 1998. Example-based learning for view-based human face detection. IEEE Trans. Pattern Anal. Mach. Intell. 20, 39–51. https://doi.org/10.1109/34.655648

The Register of UNIX® Certified Products [WWW Document], n.d. URL https://www.opengroup.org/openbrand/register/ (accessed 4.25.20).

Turk, M., Pentland, A., 1991. Eigenfaces for Recognition. J. Cogn. Neurosci. 3, 71–86. https://doi.org/10.1162/jocn.1991.3.1.71

Viggiano, J.A.S., 2004. Comparison of the accuracy of different white-balancing options as quantified by their color constancy, in: Blouke, M.M., Sampat, N., Motta, R.J. (Eds.), . Presented at the Electronic Imaging 2004, San Jose, CA, p. 323. https://doi.org/10.1117/12.524922

Vincent, E., Laganiere, R., 2001. Detecting planar homographies in an image pair, in: ISPA 2001. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis. In Conjunction with 23rd International Conference on Information Technology Interfaces (IEEE Cat. No.01EX480). Presented at the ISPA 2001. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis, Univ. Zagreb, Pula, Croatia, pp. 182–187. https://doi.org/10.1109/ISPA.2001.938625

Wan, J., Escalera, S., Anbarjafari, G., Escalante, H.J., Baro, X., Guyon, I., Madadi, M., Allik, J., Gorbova, J., Lin, C., Xie, Y., 2017. Results and Analysis of ChaLearn LAP Multi-modal Isolated and Continuous Gesture Recognition, and Real Versus Fake Expressed Emotions Challenges, in: 2017 IEEE International Conference on Computer Vision Workshops (ICCVW). Presented at the 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), pp. 3189–3197. https://doi.org/10.1109/ICCVW.2017.377

Xu, X., Kakadiaris, I.A., 2017. Joint Head Pose Estimation and Face Alignment Framework Using Global and Local CNN Features, in: 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017). Presented at the 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), IEEE, Washington, DC, DC, USA, pp. 642–649. https://doi.org/10.1109/FG.2017.81

Yan, S., Xu, D., Tang, X., Member, S., 2007. Face verification with balanced thresholds. IEEE Trans. Image Process. 262–268.

Yang, G., Huang, T.S., 1994. Human face detection in a complex background. Pattern Recognit. 27, 53–63. https://doi.org/10.1016/0031-3203(94)90017-5

Yao, P., Evans, G., Calway, A., 2001. Using affine correspondence to estimate 3-D facial pose, in: Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205). Presented at the Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205), pp. 919–922 vol.3. https://doi.org/10.1109/ICIP.2001.958274

Young, I.T., Gerbrands, J.J., Vliet, L.J. van, TU Delft, F. der T.N., 1998. Fundamentals of image processing: [dictaat behorende bij college et2720in. Delft University of Technology, Delft.

Yow, K.C., Cipolla, R., 1996. Feature-Based Human Face Detection. Image Vis. Comput. 15, 713–735.

Zenonos, A., Khan, A., Kalogridis, G., Vatsikas, S., Lewis, T., Sooriyabandara, M., 2016. HealthyOffice: Mood recognition at work using smartphones and wearable sensors, in: 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops). Presented at the 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), IEEE, Sydney, Australia, pp. 1–6. https://doi.org/10.1109/PERCOMW.2016.7457166

NON-EXCLUSIVE LICENCE TO REPRODUCE THESIS AND MAKE THESIS PUBLIC

I, Barbara Wilson Soto,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Video Face Swapping,

(*title of thesis*)

supervised by PhD Gholamreza Anbarjafari, M.A. Doğuş Karabulu

(*supervisor's name*)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Barbara Wilson Soto

20/05/2020