CAIRO UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING

Retratista

A Graduation Project Report Submitted
to
Faculty of Engineering, Cairo University
in Partial Fulfillment of the requirements of the degree
of
Bachelor of Science in Computer Engineering.

Presented by

Mohamed Shawky Zaky AbdelAal Sabae
Remonda Talaat Eskarous
Mohamed Ahmed Mohamed Ahmed
Mohamed Ramzy Helmy Ibrahim

Supervised by

Dr. Mayada Hadhoud

26th July, 2021

Abstract

الملخص

Acknowledgement

Contents

Abstract (English) 1

Abstract (Arabic) 2

Acknowledgement 3

Table of Contents 5

List of Figures 6

List of Tables 7

List of Abbreviation 8

List of Symbols 9

Contacts 10

1 Introduction 12

2 Market Feasibility Study 13

3 Literature Survey 14

4 System Design and Architecture 15

4.1 Overview and Assumptions . . . 15
4.2 System Architecture . . . 17
4.2.1 Block Diagram . . . 18
4.3 Module 1 : Speech Recognition . . . 19
4.3.1 Functional Description . . . 19
4.3.2 Modular Decomposition . . . 19
4.3.3 Design Constraints . . . 19
4.3.4 Other Description . . . 19
4.4 Module 2 : Text Processing . . . 19
4.4.1 Functional Description . . . 19
4.4.2 Modular Decomposition . . . 19
4.4.3 Design Constraints . . . 19
4.4.4 Other Description . . . 19
4.5 Module 3 : Face Code Generation . . . 19
4.5.1 Functional Description . . . 19
4.5.2 Modular Decomposition . . . 20
4.5.3 Design Constraints . . . 22
4.5.4 Other Description . . . 22
4.6 Module 4 : Code-to-Face Translation . . . 22
4.6.1 Functional Description . . . 22

4.6.2 Modular Decomposition 22
4.6.3 Design Constraints 22
4.6.4 Other Description 22
4.7 Module 5 : Face Refinement 22
4.7.1 Functional Description 22
4.7.2 Modular Decomposition 22
4.7.3 Design Constraints 22
4.7.4 Other Description 22
4.8 Module 6 : Multiple Head Poses Generation 22
4.8.1 Functional Description 22
4.8.2 Modular Decomposition 22
4.8.3 Design Constraints 22
4.8.4 Other Description 22

5 System Testing and Verification 23

6 Conclusions and Future Work 24

A Development Platforms and Tools 26

B Use Cases 26

C User Guide 26

D Code Documentation 26

E Feasibility Study 26

List of Figures

4.1 Block diagram of complete system architecture 18
4.2 Block diagram of application design 18
4.3 Detailed block diagram of the three core modules workflow 19
4.4 Illustration of feature directions in latent space 20

List of Tables

List of Abbreviation

List of Symbols

Contacts

1 Introduction

2 Market Feasibility Study

3 Literature Survey

4 System Design and Architecture

In this chapter, we discuss our working pipeline and system architecture in details. Generally, our system takes a speech note, textual description or numerical attributes as an input. It processes the input description and outputs the initial human face portrait that corresponds to the given description. Afterwards, the user is allowed to manually control some facial attributes and morphological features and to rotate the face and render it in multiple poses. In the first section, we give an overview about the system. Then, we discuss the system architecture in the second section. In the subsequent sections, each module implementation is discussed in details.

4.1 Overview and Assumptions

As mentioned above, our system basically enables the user to describe a human face in words or using numerical values and turns it into a full human face portrait that can be manipulated and rendered in multiple poses. The system relies heavily on generative models and text processing, both are iteratively designed to obtain the required results. The overall flow can be described as follows :

- The input speech notes are translated to text.
- The textual description (extracted from speech input or manually entered) is processed to extract the numerical values of the required facial features.
- The numerical values are used generate a face embedding vector that encodes the facial attributes in low dimensional space ($512D$).
- A generative model is specifically designed to translate from the low dimensional embedding into the full face portrait ($1024X1024$).
- The generated face portrait can be further refined by navigating the face embedding space and re-generating the face portrait.
- Once the user settles on the final face portrait, the system can render that face in multiple poses to provide further identification.

The previous flow provides a very versatile framework to generate face portrait and adjust it to your liking. However, there is an extremely large number of facial attributes and morphological features to describe a human face. Consequently, we have to choose a descriptive subset of these attributes to consider in the face description. We consider 32 facial attributes for face description, which are listed as follows :

- Overall face :
  - Gender : Male / Female.
  - Age : Young / Old.
  - Thickness : Chubby / Slim.

  - Shape : Oval / Circular.
  - Skin Color : Black / White.
  - Cheeks : Normal / Rosy.
- Eyes :
  - Color : Black / Blue / Green / Brown.
  - Width : Wide / Narrow.
  - Eyebrows : Light / Bushy.
  - Bags Under Eyes : On / Off.
- Nose :
  - Size : Big / Small.
  - Pointy : On / Off.
- Ears :
  - Size : Big / Small.
- Jaw :
  - Mouth Size : Big / Small.
  - Lips Size : Big / Small.
  - Cheekbones : Low / High.
  - Double Chin : On / Off.
- Hair :
  - Color : Black / Blonde / Brown / Red / Gray.
  - Length : Tall / Short.
  - Style : Straight / Curly / Receding Hairline / Bald / with Bangs.
- Facial Hair :
  - Beard / None.
- Race :
  - White / Black / Asian.
- Accessories :
  - Glasses : Sight / Sun.
  - Makeup : On / Off.
  - Lipstick : On / Off.

4.2 System Architecture

Now, let's discuss our system architecture. The system consists of 6 modules, 3 core modules of the project and 3 auxiliary modules. These modules are deployed in a *web application* to provide an easy-to-use interface for face generation and manipulation. Figure 4.1 shows the complete block diagram of the system architecture. Meanwhile, figure 4.2 shows the application design and how the modules are deployed in a web application. The *core* modules are listed as follows :

- **Text Processing :** processes the input textual description and extracts the corresponding numerical values of facial attributes. This problem is similar to *multi-label text classification*, however the outputs are normalized scores of facial attributes, which are designed carefully to match the *face code generation* process.
- **Face Generation :**
  - **Code Generation :** converts the numerical attributes values to be low dimensional face embedding. This is the most *important* and *innovative* module of our system, because it glues the desired attributes scored with the latent space of the generative model (used to generate the face), resulting in more accurate quality outputs.
  - **Code-to-Face Translation :** translates the low dimensional face embedding into the actual face portrait. For this purpose, we use `StyleGAN2`, which is a *state-of-art latent-based generative model*, whose latent space can be manipulated easily to fit our needs.

Meanwhile, the *auxiliary* modules are listed as follows :

- **Speech Recognition :** translates the input speech to textual description.
- **Face Refinement :** uses the same generative model to manually refine the generated face portrait through navigating the latent space.
- **Multiple Head Poses Generation :** rotates the generated face portrait and renders it into multiple poses.

We discuss each module in more details in the subsequent sections. Also, these modules are organized into a web application for easier usage, as shown in Figure 4.2. The application is divided into :

- **Web (Frontend) :** which contains the user interface and, also, the *speech recognizer*. The speech recognizer is moved to the frontend to reduce the network communication overhead between the web application and the server, as transmitting text is easier than transmitting speech. Moreover, the speech recognizer doesn't require high computational power, so it can be embedded in the web application.
- **Server (Backend) :** which is separated into two servers. First server contains the *text processor* and the *generative model* and serves the requests of face generation and refinement. Second server contains the *pose generator* and serves the requests of face rotation.

The two servers can communicate with each other to exchange the generated face portraits through `TCP sockets`. Meanwhile, the web application communicates and sends requests to the servers through `HTTP REST API`.

4.2.1 Block Diagram

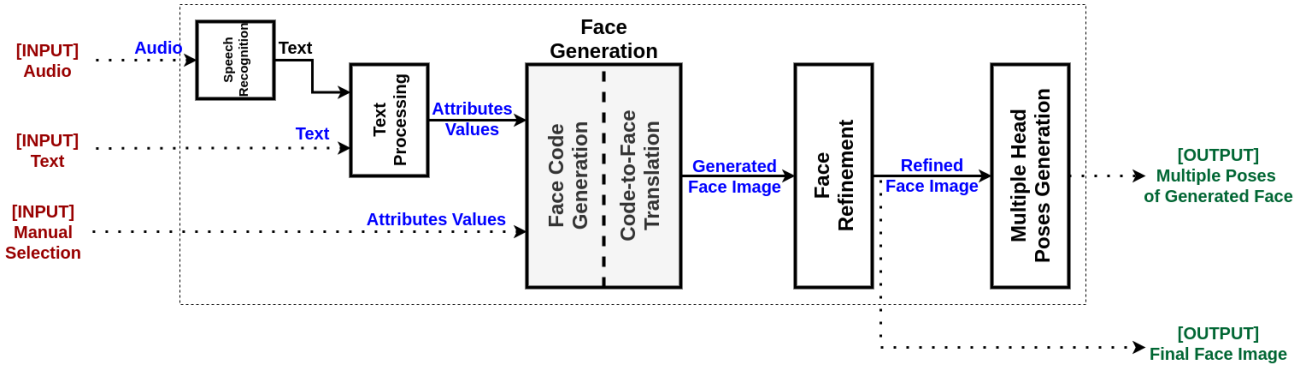

Figure 4.1: Block diagram of complete system architecture

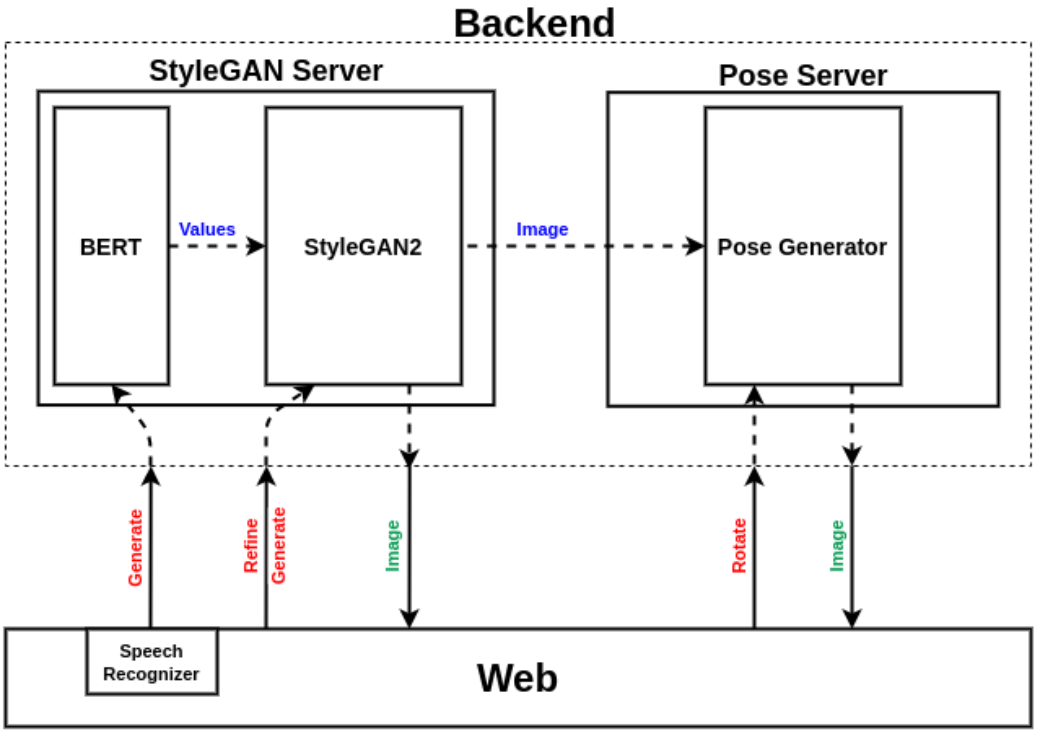

Figure 4.2: Block diagram of application design

4.3 Module 1 : Speech Recognition

4.3.1 Functional Description

4.3.2 Modular Decomposition

4.3.3 Design Constraints

4.3.4 Other Description

4.4 Module 2 : Text Processing

4.4.1 Functional Description

4.4.2 Modular Decomposition

4.4.3 Design Constraints

4.4.4 Other Description

4.5 Module 3 : Face Code Generation

Here, we discuss the face code generation from numerical values of facial attributes. This is the most important and innovative module in our system and the first stage of *face generation*. It's worth noting that we use both of the terms *"feature"* and *"attribute"* to refer to a facial attribute, like hair color or nose size.

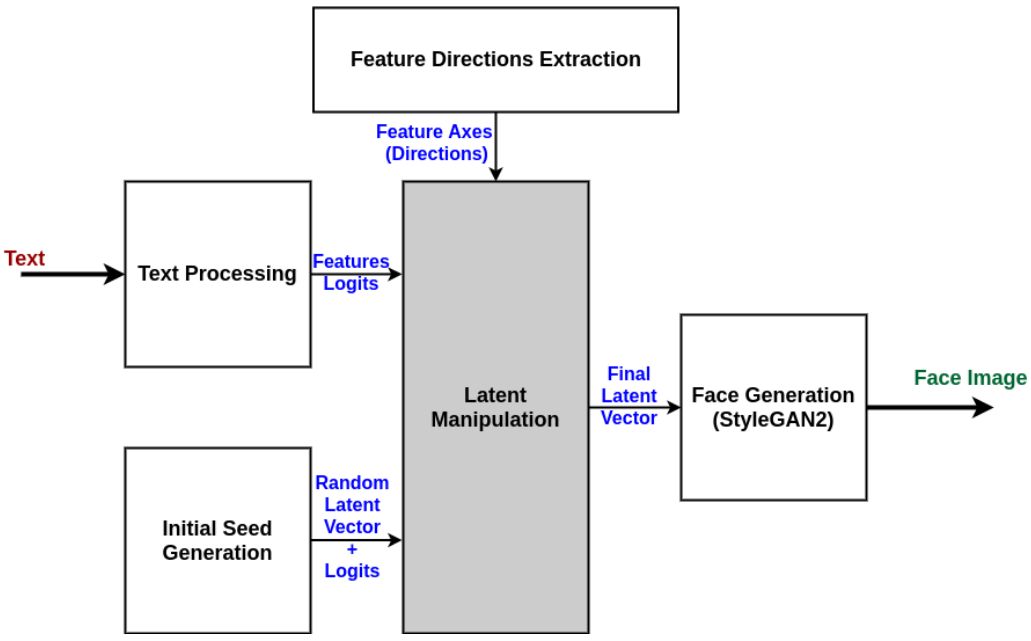

Figure 4.3: Detailed block diagram of the three core modules workflow

4.5.1 Functional Description

Figure 4.3 shows a block diagram of the interaction between the 3 core modules. We can see that the code generation module is the main driver of our face generation process. Generally, it converts the numerical attributes values (a.k.a. *logits*) into a face embedding vector

that matches the design of the latent space of the face generator (*StyleGAN2*). Basically, it starts from an initial vector and uses the *required feature values* and *extracted feature directions* to transform this vector into the final latent vector, which is passed to the generative model.

- **Input :**
  - Numerical values of facial features (logits).
- **Output :**
  - Low dimensional face embedding vector (latent vector).

4.5.2 Modular Decomposition

As figure 4.3 tells, the code generation module can be torn down into 3 sub-modules, which are **latent manipulation**, **initial seed generation** and **feature directions extraction**. Each sub-module is discussed in details to show how they integrate to each other to achieve the desired goal.

Feature Directions Generation Since, we use `StyleGAN2` [1] as our generative model, we have a full $512D$ latent space that is used to encode the whole face attributes. The changes in this latent space maps to the generated face image and similar features occupies the same area in the latent space. Consequently, we have to come up with a way to extract the axes (*hyperplanes*) in this latent space to define each of our 32 facial features. These feature directions are, then, used to manipulate the latent vector, in order to map to the required face image.

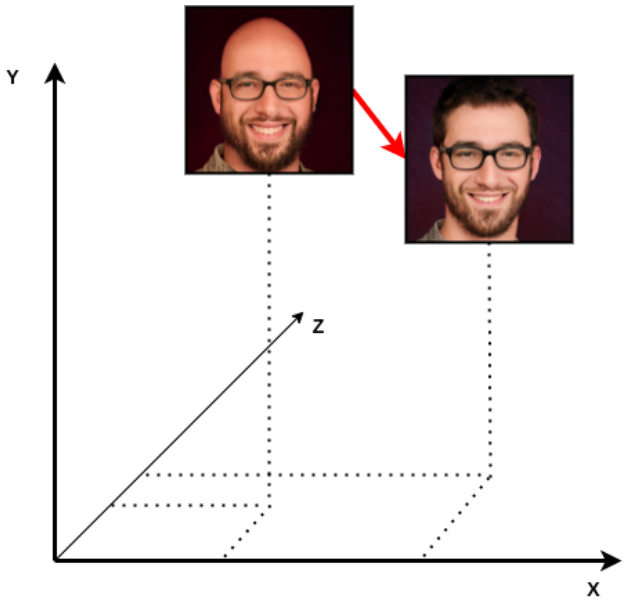

Figure 4.4: Illustration of feature directions in latent space

Figure 4.4 further illustrates the idea of feature directions in the latent space. Here, we plot two face images in a $3D$ latent space. We can see that the difference between the

two images in the existence and the absence of the hair, thus the red arrow represents the *baldness* feature direction in that $3D$ latent space (moving along this particular vector causes hair density to change).

Our method of extracting the feature directions (*hyperplanes*) consists of 3 steps :

1. **Code-Image Pairs Generation and Classification :** First, we use `StyleGAN2` to generate a large number of synthetic faces from random latent vectors. After so, we cluster the synthetic images (along with their latent vectors) according to each feature. The clustering can be based on discrete categories (like *hair color* or *race*) or continuous values (like *hair length* or *nose size*). We randomize the synthetic images in each clustering process to have better generalization and to cope with potential generation noise. For classification and regression, we use one of three possible methods, which are **manual labelling**, **classical image processing techniques** and **neural networks**. Thus, the output of this process is different groups of synthetic images sharing common facial features, along with their latent vectors.

2. **Feature Directions Fitting :** Now, we have a set of latent vectors (`X`) and their corresponding feature values (`Y`). It's required to find a set of feature directions that satisfies the mapping between feature vectors and values. This problem can be formulated as :

$$Y = A_f \cdot X \tag{1}$$

   Where A_f is the axis (direction) of feature f.
   We can obtain the solution to this equation in a closed form. However, due to the noise in both generation and classification, along with the non-linear nature of the problem, we opt to use *ML* methods, specifically **Logistic Regression** and **SVM**. Meanwhile, we cannot see any difference between both methods, as they yield almost the same results. Finally, the generated feature directions are normalized to unit vectors :

$$A_{unit} = \frac{A}{||A||} \tag{2}$$

3. **Directions Orthogonalization :** Facial features entanglement is one of the most difficult challenged of face generation. Some attributes in the human face tend to be extremely entangled by nature. For example, Asians rarely have curly hair, a woman cannot have beard and a man cannot put on makeup. Since `StyleGAN2` is trained and tuned on **FFHQ** dataset [2], which contains real human faces, it is normal to notice some entanglement between some features. Consequently, the feature directions have to be further disentangled by using *orthogonalization*. The orthogonalization process is done iteratively, starting from the most accurate feature directions. We orthogonalize other feature directions on the accurate ones, so that we have completely independent feature directions, where tuning one direction doesn't affect the others. The directions are orthogonalized as follows :

$$A_{proj} = (A \cdot B_{unit})B_{unit} \tag{3}$$

$$A_{orthogonal} = A - A_{proj} \tag{4}$$

To ensure convergence to reasonable set of feature directions, we use a threshold margin to stop the orthogonalization process, which is from 85 to 90 degrees (5 degrees on each side of normal angle).

Initial Seed Generation

Latent Manipulation

4.5.3 Design Constraints

4.5.4 Other Description

4.6 Module 4 : Code-to-Face Translation

4.6.1 Functional Description

4.6.2 Modular Decomposition

4.6.3 Design Constraints

4.6.4 Other Description

4.7 Module 5 : Face Refinement

4.7.1 Functional Description

4.7.2 Modular Decomposition

4.7.3 Design Constraints

4.7.4 Other Description

4.8 Module 6 : Multiple Head Poses Generation

4.8.1 Functional Description

4.8.2 Modular Decomposition

4.8.3 Design Constraints

4.8.4 Other Description

5 System Testing and Verification

6 Conclusions and Future Work

References

[1] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2020.

[2] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

A Development Platforms and Tools

B Use Cases

C User Guide

D Code Documentation

E Feasibility Study