**System Design Document**

**For**

**Machine Learning Algorithm for Rhinoplasty (M-LAR)**

Team members:

Chris Graziano

Victoria Jordan

Anton Kiselev

Jacob Preseau

Chintan Thakrar

|  |  |
| --- | --- |
| Version/Author | Date |
| 1.0/Team 1 | 9/29/20 |
| 1.1/Team 1 | 10/1/2020 |
| 1.2/ Anton Kiselev | 10/27/2020 |
| 1.3/ Anton Kiselev | 10/28/2020 |
| 1.4/ Anton Kiselev | 11/13/2020 |
| 1.5/ Anton Kiselev | 11/16/2020 |
| 1.6/ Jacob Preseau | 11/24/2020 |
| 1.7/ Victoria Jordan | 11/26/2020 |
| 1.8/ Anton Kiselev | 11/29/2020 |

TABLE OF CONTENT

[1](#_heading=h.30j0zll) INTRODUCTION 3

[1.1](#_heading=h.1fob9te) Purpose and Scope 3

[1.2](#_heading=h.3znysh7) Project Executive Summary 3

[1.2.1](#_heading=h.tyjcwt) System Overview 3

[1.2.2](#_heading=h.46r0co2) Design Constraints 3

[1.2.3](#_heading=h.26in1rg) Future Contingencies 3-4

[1.3](#_heading=h.2lwamvv) Document Organization 4

[1.4](#_heading=h.1ksv4uv) Project References 4

[1.5](#_heading=h.111kx3o) Glossary 4

[2](#_heading=h.2jxsxqh) SYSTEM ARCHITECTURE 4

[2.1](#_heading=h.z337ya) System Hardware Architecture 5

[2.2](#_heading=h.3l18frh) System Software Architecture 5-6

[2.3](#_heading=h.206ipza) Internal Communications Architecture 6

[3](#_heading=h.4k668n3) HUMAN-MACHINE INTERFACE 6

[3.1](#_heading=h.3whwml4) Inputs 6

[3.2](#_heading=h.2zbgiuw) Outputs 6-7

[4](#_heading=h.1egqt2p) DETAILED DESIGN 7

[4.1](#_heading=h.3as4poj) Hardware Detailed Design 7

[4.2](#_heading=h.1pxezwc) Software Detailed Design 7-9

[4.3](#_heading=h.3ygebqi) Internal Communications Detailed Design 9-10

[5](#_heading=h.2dlolyb) EXTERNAL INTERFACES 10-11

[5.1](#_heading=h.sqyw64) Interface Architecture 11

[5.2](#_heading=h.3fwokq0) Interface Detailed Design 12-17

[6](#_heading=h.3cqmetx) SYSTEM INTEGRITY CONTROLS 17

7 APPENDIX 18

SYSTEM DESIGN DOCUMENT

# INTRODUCTION

## Purpose and Scope

This document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, and external interfaces for the Machine Learning Algorithm for Rhinoplasty (M-LAR) system.

## Project Executive Summary

This section provides a description of the M-LAR system from a management perspective and an overview of the framework within which the conceptual system design was prepared.

### **System Overview**

The system will provide a doctor an additional way to incorporate customer feedback and more easily provide a rhinoplasty service. The system will take a 2D frontal photo of the user, and extract facial points. Utilizing a facial ratio algorithm, the system will find 40 faces from our database closest to that of the user. Of these faces, the system will display up to 5 nose clusters allowing the user to navigate to each cluster and choose the desired nose. After the user selects a desired face from the nose cluster, the system will calculate required changes, in this case distances between the facial points of the users’ photo and the chosen face from the database, so that the doctor can perform the rhinoplasty with the correct measurements.

### **Design Constraints**

The current version of the project was made with the assumption that it will work only with 2D photos, such as a face looking straight forward into a camera with aligned sides.

The person in the photo should remove their glasses if they wear them, as they disturb the gathering of the face coordinates.

Scaling of the distances on the photo may have error due to frontal view only constraints.

### **Future Contingencies**

The current version of the Python application displays windows via the Tkinter library [4]. This could be changed if another library that could better suit our future needs such as supporting 3D images of the users is discovered.

This project, like any other machine learning project, is dependent on the initial data inputs used for training, categorization and prediction. At the current stage, it is assumed that the current selected dataset is sufficient in terms of quality and quantity to successfully complete the study.

## Document Organization

This document is designed to describe the structure of the Machine Learning Algorithm for Rhinoplasty (M-LAR) system. The following sections will provide information on what the product does, limitations, interactions, interfaces, hardware and software designs, and security.

## Project References

* + 1. System Requirements Specification

## Glossary

* CV2 - CV2 is a subcomponent of OpenCV library. OpenCV-Python is a library of Python bindings designed to solve computer vision problems. [2]
* Dlib - Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software. Dlib is responsible for assigning the facial coordinates using a given image, essential for scanning the user’s face [1].
* “Facial key points”, “facial coordinates” and “facial landmarks” are used as interchangeable terms, meaning specified facial points extracted by Dlib.
* "Nose points" are a subset of facial coordinates, referring just to the 9 points used to identify the nose.
* KNN - "*k*-nearest neighbors", a machine learning algorithm that, when given a test point represented by a point on the coordinate plane, will find the *k* points in the data set, also in the coordinate plane, that are nearest to the test point [3].
* M-LAR - Machine Learning Algorithm for Rhinoplasty (M-LAR) system
* "Patient", “Client” and "User" are used as interchangeable terms.
* Shape\_predictor - definition here

# SYSTEM ARCHITECTURE

This section describes an overview of the hardware and software architecture for the M-LAR system and subsystems.

## System Hardware Architecture

For the current version, the system does not have any hardware components.

## System Software Architecture

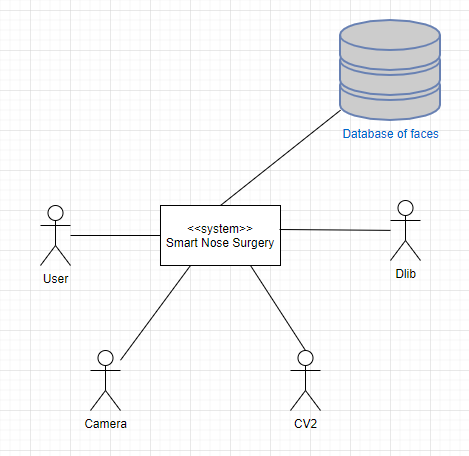


Figure 1. Context view of the system.

Figure 1 displays a context overview of the M-LAR system. The system consists of five (5) crucial components: Database of faces, Dlib, CV2, Camera, and User. The database of faces contains all of the images we have available for the facial ratio algorithm to pull 40 faces from. The database includes 2D frontal images of both men and women. Dlib assigns the facial coordinates to a given image. CV2 is used for image detection and capturing the users 2D frontal photo by accessing the users’ computer camera. The user provides access to the camera and a canvas to pinpoint facial coordinates as well as selecting a nose candidate from the nose clustering.

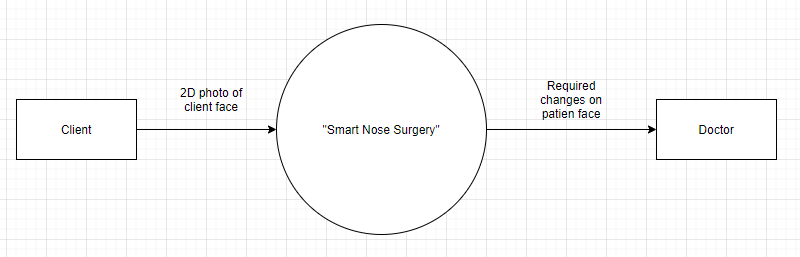


Figure 2: Data-flow diagram. Level 0

As shown in Figure 2, the system contains 3 main components: user, main process and doctor. The user provides a 2D frontal photo for the main process to analyze their facial coordinates and deliver options for the user to select from the performed nose clustering. The doctor receives data on what changes they need to perform on the user to meeting the measurements of the users desired nose preference.

## Internal Communications Architecture

As shown in Figure 4, the system communicates via passing results of previous processes.

# HUMAN-MACHINE INTERFACE

This section provides the detailed design of the system and subsystem inputs and outputs relative to the user.

## Inputs

The main input for the current version is a 2D frontal photo of a patient face in jpg format, in which the patient holds his/her head upright without tilts and turns, and looks directly towards the camera. The inputs for the external processes are coordinates of facial features. With the current status and scope of the project, it is assumed that the input will always be a face; this implies that exceptions are currently not being handled. Additionally, the user is required to select a face as the "target face", which will be used to calculate the required changes to the user's nose as the difference between the user's nose points and the target face's nose points.

## Outputs

The main output of the current version of the system is a list of the required coordinate changes on the patient's nose. The coordinates for a certain face are calculated as a set of numbers between 0 and 1 of the point's location scaled to the length and height of the face in question, which are determined by finding the outermost points among its facial coordinates. For example, a point (0.48, 0.59) would indicate that the point is 0.48 face widths to the right of the leftmost point and 0.59 face heights below the topmost point. The origin (0, 0) represents the top-left corner of the face, and (1, 1) represents the bottom-right corner. With these coordinates calculated for both the user's face and a "target face" selected by the user, the system then outputs the difference between the two faces' nose points.

# DETAILED DESIGN

This section provides the information needed for a system development team to build and integrate the hardware components, code and integrate the software modules, and interconnect the hardware and software segments into a functional product.

## Hardware Detailed Design

As mentioned in section 2.1, the current version of the system does not contain any hardware components and requirements.

## Software Detailed Design

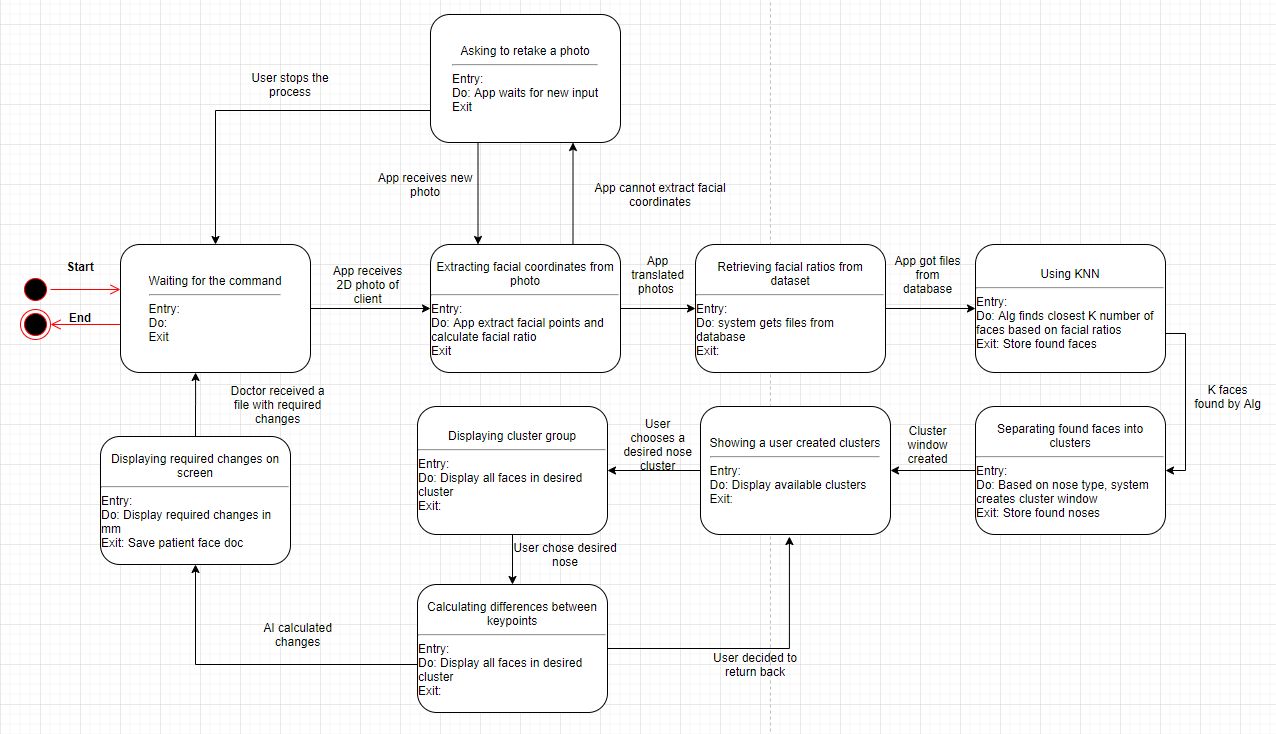


Figure 3: State Chart Diagram

Figure 3 describes what states and conditions will be cycled through the system. After the system is turned on, it will wait for the user input, in this case, a 2D photo. After receiving a 2D photo by using the CV2 library, the system will extract facial coordinates by using Dlib. If the system is unable to extract coordinates it will ask the user to re-upload or retake the photo. After extracting facial coordinates, the system will calculate the facial ratio for the user's face and extract facial ratios for faces from the dataset. After that system will apply the KNN algorithm to find the K number of the closest faces. The K value means how many faces the user wants to look at and will be specified by the user after uploading the photo. Based on the nose cluster, the system will divide the results of the KNN algorithm into available clusters which vary for the user's face. While displaying available clusters, the user will be able to choose the cluster which he or she likes. After choosing any available cluster, the system will display all faces from KNN results in this cluster. Users will be able to choose the desired face. After choosing the desired face, the system will start calculating distances between nose points by applying the Euclidean distance formula, which will be meaning the required changes on the user's face. After calculations, the system will display user photos with required changes. At the final step, the doctor will be able to save the client photo and the system will wait for a new command.

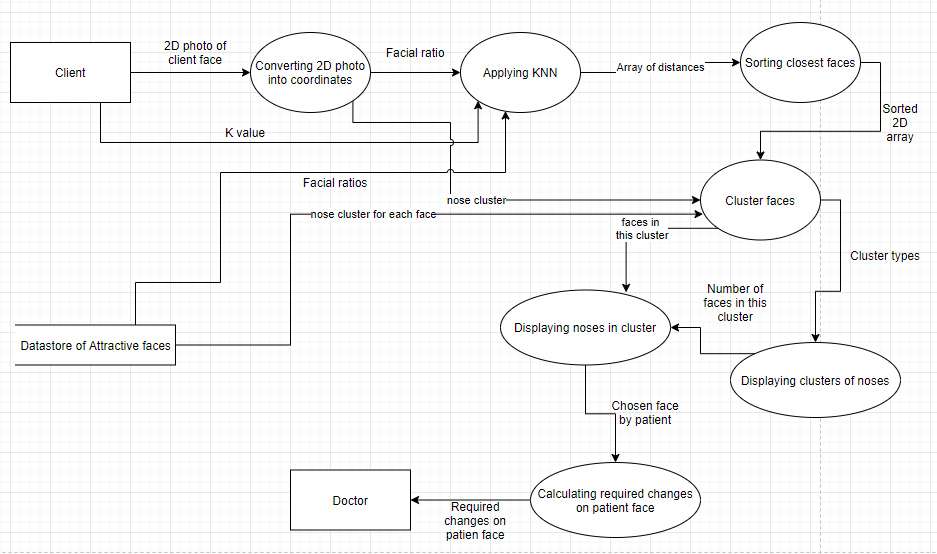


Figure 4: Data-flow diagram. Level 1

As shown in figure 4, the system will process 2D face into facial coordinates using the Python Dlib and OpenCV modules. The coordinates are computed with respect to the origin, which is the top-left corner of the screen. It is inferred that the final matrix that is to be fed to the ML model will be independent of the resolution of the input image, because it will consist of proportions or ratios between coordinates. The use of ratios is reliable as they are automatically adjusted to provide a consistent matrix with respect to each face. After that, the system will pass the coordinates, stored in 2D array, to the KNN algorithm, which will use the coordinates to calculate the Euclidean distance relative to the user's face ratio. After finding the K value of the closest neighbors, the system will divide the result into nose clusters, based on nasal index [5]. Users will be able to choose which cluster they want to explore, and which face they like. After choosing the desired face, the system will use nose points and calculate distances between them. After calculations are done, the system will display the client's face with required changes.

## Internal Communications Detailed Design

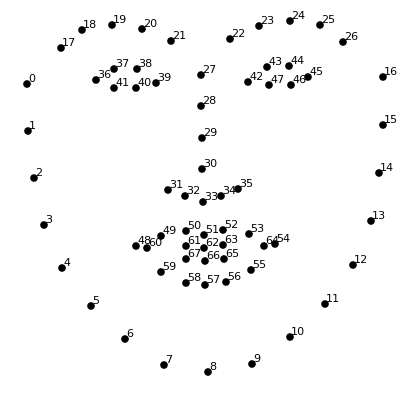


Figure 5. Facial coordinates. Picture extracted from <https://towardsdatascience.com/detecting-face-features-with-python-30385aee4a8e>.

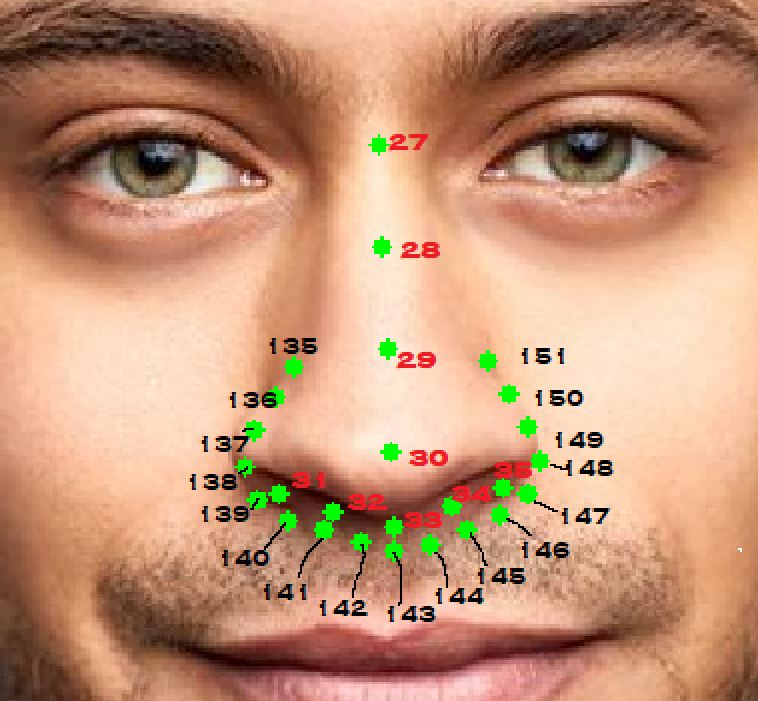


Figure 6. Nose points by 68-shapeppredictor and 194-shape-predictor.

The main instance of internal communication occurs when the system reads and converts the dataset of face images into a respective set of coordinate points, shown in Figure 5 to send as input to the KNN algorithm. The dataset contains a set of images, currently stored as JPG files. The face recognition component converts these images into sets of coordinates, calculates facial ratio for each face and stores it in a separate CSV file. Facial ratio represented by width and height ratio. For the width ratio are used horizontal components of the coordinate, while in height ratio are used vertical components of coordinates. The width ratio is computed by dividing distance from point 0 to point 16 by distance from point 42 to point 16. The height ratio is computed by dividing distance from point 27 to point 8 by distance from point 8 to point 64. Each nose is classified into nose clusters based on nasal index. Nasal index is calculated by formula[5] and for calculations the system needs width and height of the nose. For nasal index calculations were used two shape predictors: 68 shape predictor and 194 shape predictors. The width of the nose was calculated by subtracting the horizontal component of point 138 from the horizontal component of point 148. The height of the nose was calculated by subtracting the vertical component of point 27 from the vertical component of point 33.

# EXTERNAL INTERFACES

The implemented software model uses multiple predesigned, developed, and trained models to assist with the development of the system. This decision was considered to complete the specified requirements within the allocated times. Modules include but are not limited to:

1. Dlib
2. OpenCV

These modules help the system get the facial coordinates from an aligned image and then change the image to display those points, respectively. It is assumed that the final system is expected to be used by the patient, who takes a photo of their face and then picks from some presented noses.

## Interface Architecture

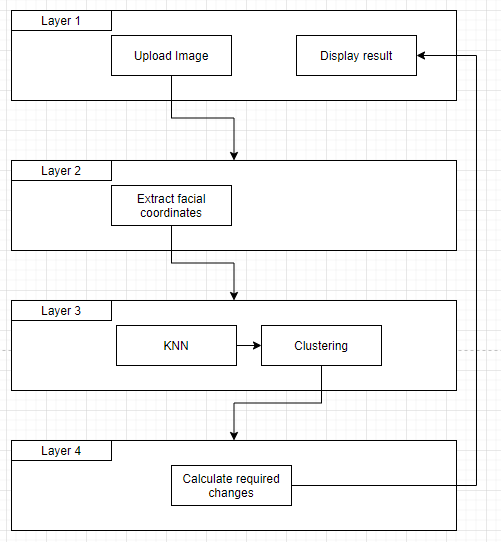
~~~~

Figure 7. Interface architecture.

Figure 6 describes the design of the system’s interface architecture. The system contains multiple layers which will display current tasks such as taking a photo, displaying nose clusters, displaying each cluster and displaying the final window with required changes.

## Interface Detailed Design

This section provides a detailed interface layout of the system.

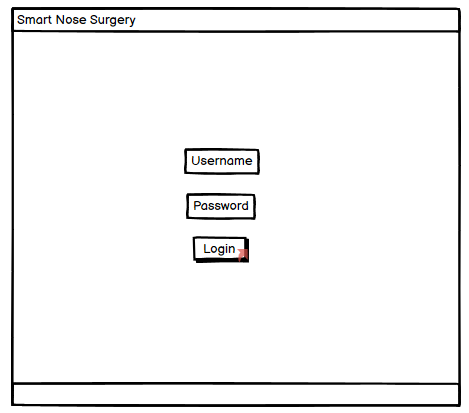


Figure 8. Login screen.

The “Smart Nose Surgery” app provides a basic login system, contains username and password input as shown on Figure 8. If login is unsuccessful the system will ask the user to re-enter username and password. If the user enters three unsuccessful entries, the system will be blocked for a time of one minute.

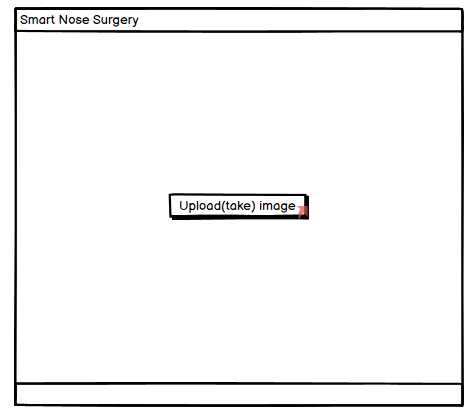


Figure 9. Upload image window.

After successful login, the user will be asked to upload a photo (.jpg format accepted) or take a new one with a web camera.

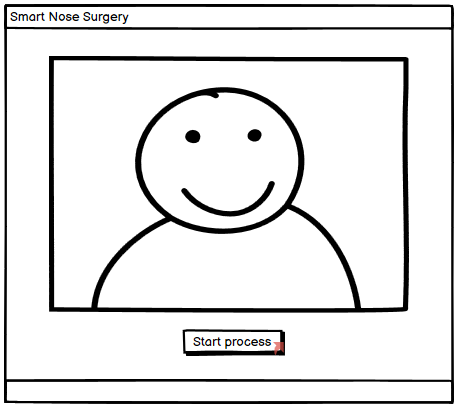


Figure 10. Window with taken photo

After user uploads or takes a photo, the system will display it for final confirmation before future processes.

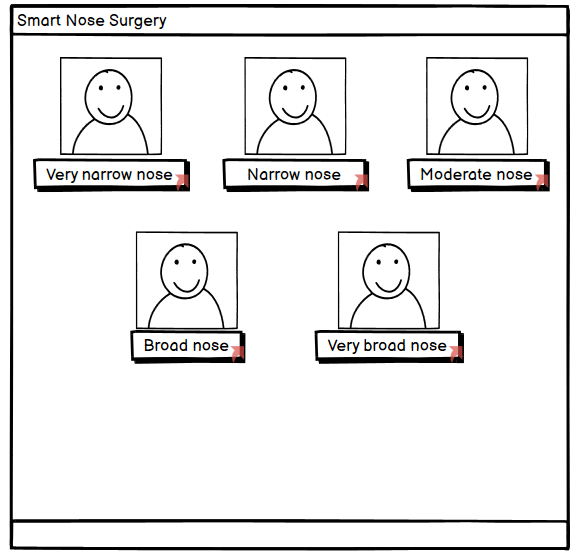


Figure 11. Window with nose clusters.

After clicking the “Start Process” button, the system will apply KNN algorithm, to find K the nearest similar faces, or in other words neighbors. Results of the KNN algorithm will be displayed in the frame based on the nose cluster. All faces have their own nose cluster, based on nasal index. In our system we have 5 clusters: very narrow nose, narrow nose, moderate nose, broad nose and very broad nose. It is possible for some user faces, the system will not display all 5 clusters, but will display at least one of them.

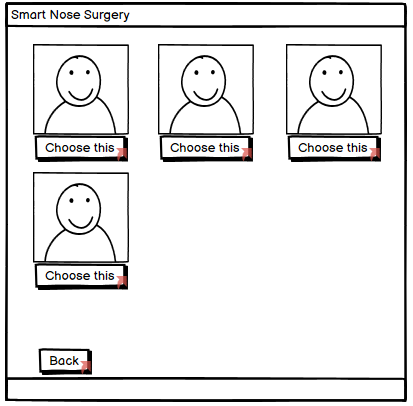


Figure 12. Cluster window.

After the user clicks their desired nose cluster, the system will display faces in this cluster. Number of faces in each cluster depends on the KNN algorithm's result. In this frame the user is able to choose the desired nose and the system will start the calculation process.

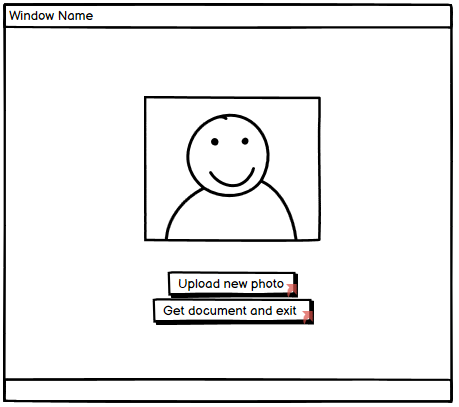


Figure 13. Final window with required changes on patient face.

At the end, the system will display the user's face with marks on it, meaning required changes on the nose for the user. The user will be able to upload a new photo or get a photo with changes and return to the main window.

# SYSTEM INTEGRITY CONTROLS

The Python application is expected to have a security level of login input. The login will consist of a user-name and password. This will add a security layer to the system. The users i.e. the patient and doctor will have different levels of access more specifically the patients will not be able to view the computed difference. The log-in details of each user will be stored on an external SQL server.

The final submission of this project is expected to have an exception handler to make sure that the user submits a picture of the face and nothing else. The system will be able to detect and disregard any picture that does not contain a face that can be used for the processing section.

**7 APPENDIX**

**References:**

[1] Dlib C++ Library. (n.d.). Retrieved October 1, 2020, from <http://dlib.net/>

[2] Introduction to OpenCV-Python Tutorials. (n.d.). Retrieved October 1, 2020, from

<https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html>

[3] Nearest Neighbors. (n.d.). Retrieved November 13, 2020, from https://scikit-learn.org/stable/modules/neighbors.html

[4] Tkinter - Python interface to Tcl/Tk. (n.d.). Retrieved October 1, 2020, from <https://docs.python.org/3/library/tkinter.html>

[5] Uzun, A., & Ozdemir, F. (2014, July 22). Morphometric analysis of nasal shapes and angles in young adults. Retrieved November 14, 2020, from https://www.sciencedirect.com/science/article/pii/S1808869414000809