Capstone Project

X-HAND IMAGE PROCESSING

| Theo Madikgetla – **MDKTHE015** |
| Cameron Broomfield – **BRMCAM003** |
| Connor Macleod – **MCLCON002** |
| **CSC3003S** | 23/09/2022 |

| Client – **Prof. Patrick Marais** |

DECLARATION

We declare that this project report for **X-Hand:** *Image Processing* is based on our original work that was conducted during the projects given timeline. We further certify that:

1. Our work has not been shared with anyone who was not part of the group as we understand that it risks being plagiarized
2. Where we used materials (data, theoretical analysis, and text) from other sources, we have given credit to the author, in the text of the report and given the details in the reference list.

ABSTRACT

This project report outlines an image processing application that is developed using Python's *OpenCV* library. The objective is to design an application that takes in a batch of hand x-ray images and applies the image processing pipeline and then aligning the processed images with a template image that is considered perfect by the user. Images with the same features will then be clustered together using the *K-means* algorithm. A user will be interacting with this application via a *Graphical User Interface.*

Table of Contents

1. Introduction

This project's intention is to preprocess images of xray hands to be more suitable for later machine learning applications; this involves cleaning up, rotating, and aligning the images. The program has added functionality to perform analysis on batches of X-ray hand images using k-means clustering. The main purpose is to normalize and simplify images so that machine learning can be more efficiently performed over the batches in future projects.

The image processing application is developed to support an 8- stage pipeline. The pipeline consists of histogram normalization, image thresholding, component analysis and extraction, computing contours, convex hull and detecting convexity defects in order to extract the features of a hand and use them for and lastly image alignment. The software was developed as an evolutionary prototype building on each step of the image processing and improving on it as each subsequent step relied on the quality of previous outputs. It is designed to use the Model, View, and Control, MVC, architectural design pattern

A user can interact with the application using a graphical user interface GUI. The application processes hand x-ray images with the intention of them being used for machine learning in another project.

2. Requirements Analysis

The following are the requirements that the client communicated during the design phase. All the requirements were implemented in the final Xhand application.

Functional Requirements

- Select a batch of images
- Normalize the images to get rid of noise
- Threshold images
- Perform a connected component analysis and extract only the hand.
- Compute convexity hull and convexity defects
- Use these points to locate the middle fingertip and point it meets the palm.
- Rotate the images to align the middle finger vertically
- Align and overlay images to target image selected by the user.
- Compute an alignment error, indicating how well the images align with the template image.

Non-functional Requirements

- Application should not take long to process a large number of images.
- Application should be simple to use.
- Applications processing code can be used independently in other projects.

Usability

- The application should not be too complicated, which will allow any user to be able to learn how to use it quickly.
- The user interface should have a nice aesthetic to make it appear more orderly, well designed, and professional. This will have a positive effect on the user, as they will be more invested in learning how to use the application.
- Buttons should be the right size and color so that they are easily visible.
- Application opens and caters most to its most common use case being to process a large batch of images.

Extras

These are requirements which were not part of the core features that the client requested. They are;

- Use k-means to cluster images with similarities using their choice of a few metrics.
- Allow users to search for images in the same cluster and display them.
- After the user has loaded up the batch of images, they must be able to reject ones they do not want to be processed.

3. Use Cases

Use case diagram

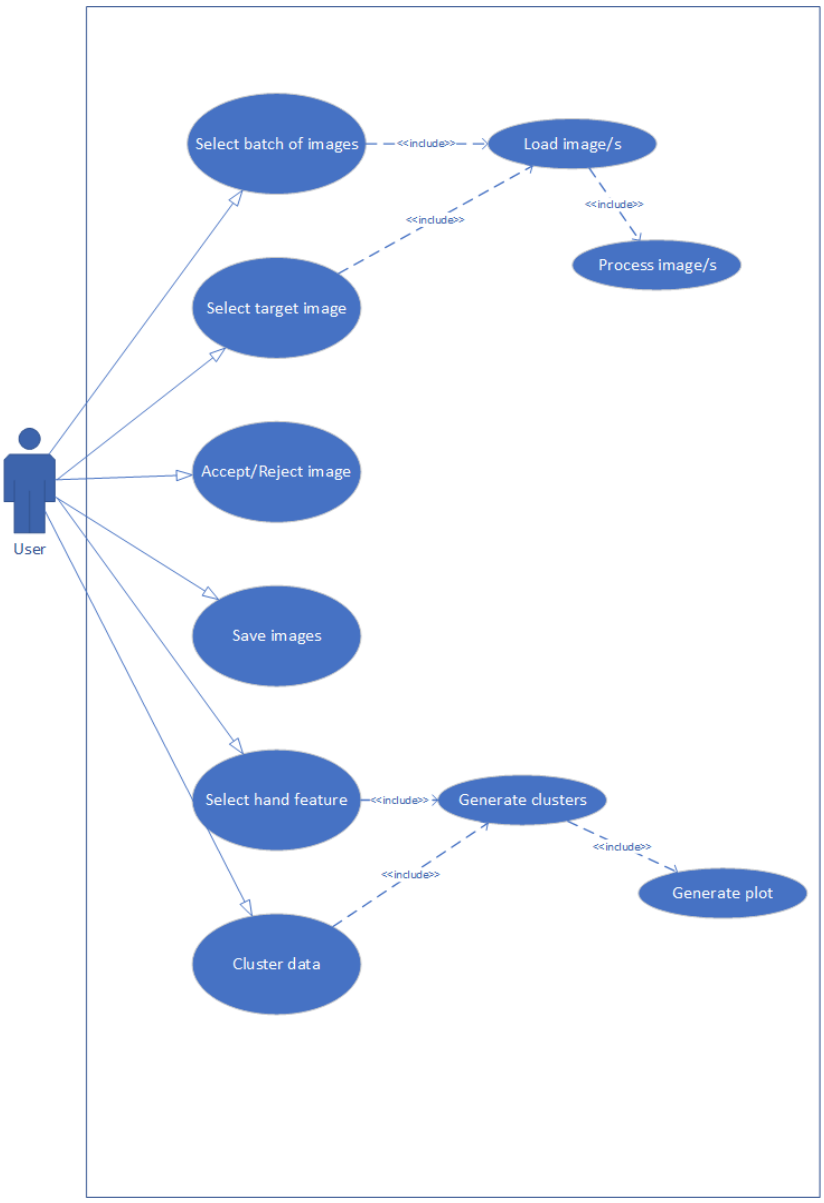


Figure 1: Use case diagram for the application

Narrative

Use Case	**Select a batch of images**
Use Case ID	101
Actors	Application User
Trigger	User selects an arbitrary number of images
Pre-Condition	User clicks on “Select batch of images” button
Post-Condition	Images loaded in the left pane, showing the first image that was loaded.

Basic Path	1. User clicks on “Select batch of images” 2. Application opens file dialog box. 3. User locates the batch of images they want to select from. 4. User selects an arbitrary number of images. 5. Application loads the images on the left pane and shows a scrollbar to indicate there are multiple images loaded.
Alternative Path	1. User clicks on “Select batch of images” 2. Application opens file dialog box. 3. User locates their batch of images to select from 4. User selects only a single image 5. Application loads the image on the left pane and does not show scrollbar or scrollbar inactive, as there is only one image.

Use Case	**Select Target Image**
Use Case ID	102
Actors	Application User
Trigger	Load base image that will be used for alignment
Pre-Condition	User clicks on “Select Target Image” button
Post-Condition	Image displayed in the “Overlay and alignment” tab.
Basic Path	1. Click on “Select Target Image” button. 2. Application shows file dialog box. 3. User locates images on their local machine and selects it. 4. The application loads the image in the “Overlay and alignment” tab.
Alternative Path	

Use Case	**Accept/Reject image**
Use Case ID	103
Actors	Application User

Trigger	Mark an image as accepted/rejected by the user
Pre-Condition	User toggles the Accept/Reject switch
Post-Condition	Rejected images are not saved to disk/*
Basic Path	1. Toggle the Accepted/Reject switch 2. Image is marked as accepted/rejected 3. Rejected images are not saved
Alternative Path	

Use Case	**Save images**
Use Case ID	104
Actors	Application User
Trigger	Save the batch of processed images
Pre-Condition	User clicks on “Save images” button
Post-Condition	Images are saved to disk
Basic Path	1. Click on “Save images” button. 2. Application shows file dialog box. 3. User locates save directory on their local machine and selects it. 4. The application saves the images in the directory.
Alternative Path	

Use Case	**Select hand feature**
Use Case ID	105
Actors	Application User
Trigger	User wants to select a different hand feature by which to cluster the images
Pre-Condition	User selects different hand feature from drop down menu
Post-Condition	Application displays a plot of the hand feature data in clusters

Basic Path	1. User clicks on selects different hand feature from drop down menu 2. User is prompted to enter the number of clusters they want. 3. Data is clustered and a plot is generated showing the clusters in different colours.
Alternative Path	

Use Case	**Cluster Data**
Use Case ID	106
Actors	Application
Trigger	User wants to view the results
Pre-Condition	User clicks on “Cluster data” button
Post-Condition	Application displays a plot of the data in clusters
Basic Path	1. User clicks on “Cluster data” button 2. User is prompted to enter the number of clusters they want. 3. Data is clustered and a plot is generated showing the clusters in different colours.
Alternative Path	

4. Design

Overview

The Xhand image processing application uses the Model-View-Controller, MVC, architectural pattern, and it separates the input, processing, and output of an application. It is divided into three components called the model, the view, and the controller. The controller receives all the requests for the application and then instructs the model to prepare any information required by the view. The view uses that data prepared by the controller to bring the final output.

The MVC was chosen for this application because it supports parallel development, and it allows easy modification of code without affecting or having to browse through the entire code of the application.

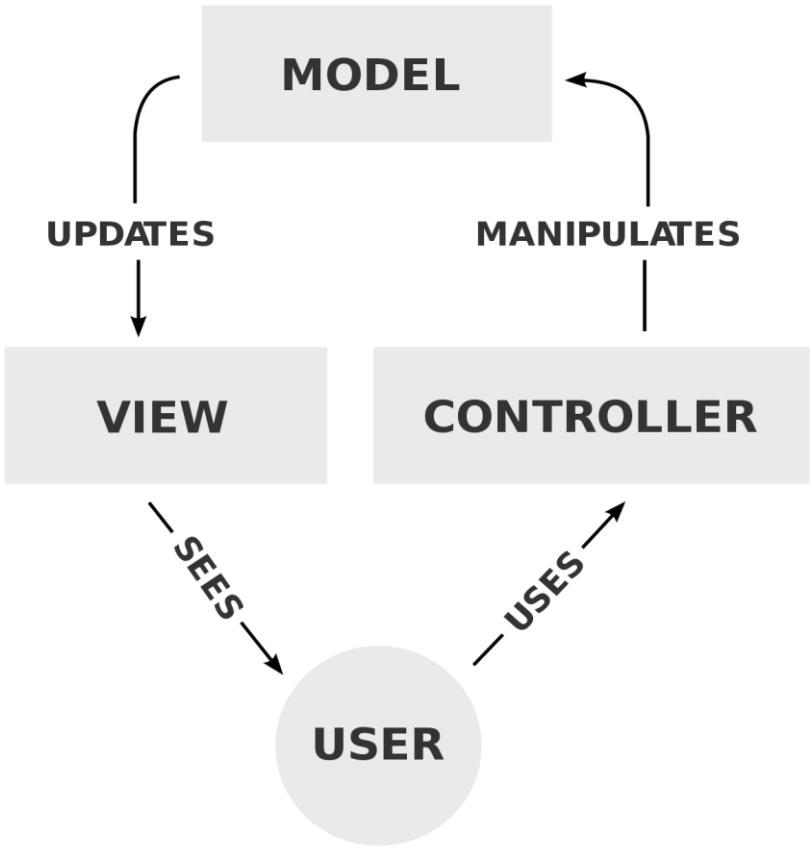


Figure 2: Diagram showing MVC architecture

Overall Architecture

Controller
All stages of the image processing pipeline fall under the controller component of the MVC, as it controls how images are manipulated.

View
The code responsible for the user interface is inside the view component of the MVC, as it creates the graphical user interface that allows the user to interact with the application.

Model
This where all the data of the application is stored. The view component requests the model to give information so that it presents the output presentation to the user.

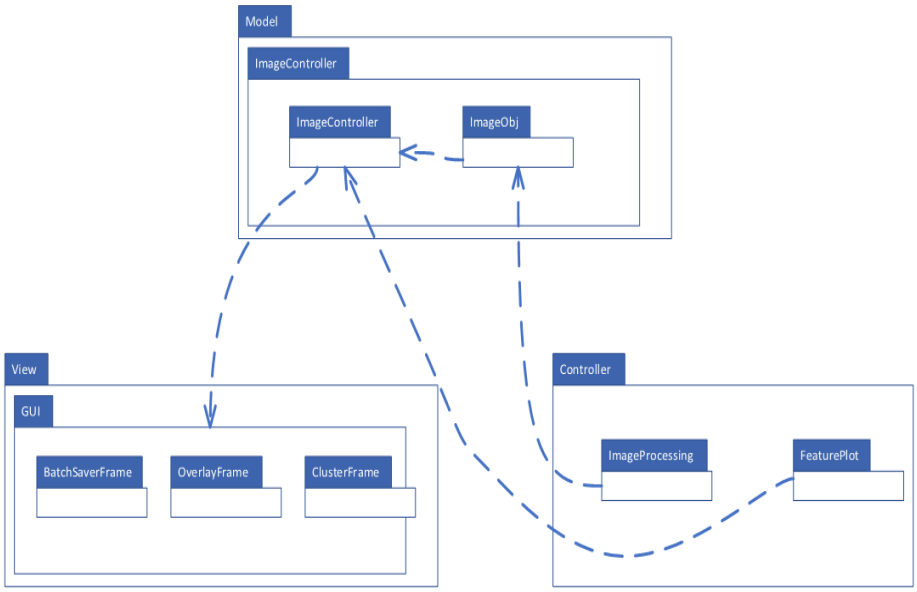


Figure 3:Shows how different classes interact using MVC

5. Implementation

- *Data Structures*
    - *Dictionary*

      It is used for storing an image with its numerical data. The image name is the key, the value is the numerical data belonging to it. The purpose of using a dictionary is to implement the search feature after the images have been clustered together by the k-means. When the user chooses to display images in a certain cluster, the dictionary will be traversed and will return all the image names belonging to that cluster.

    - *Lists*

      They are used to store all the pixel points that are extracted from the hand during certain stages of the processing pipeline. They are also used for storing the color name strings which are used when plotting the k-means data.
- *User Interface*

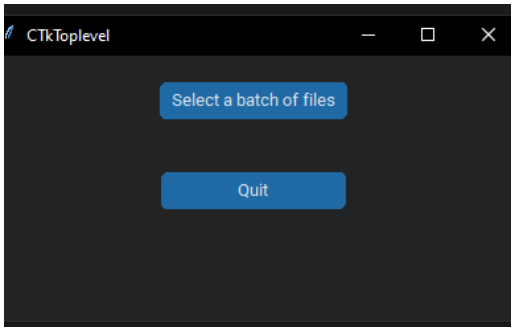


Figure 1: First window prompting user for batch selection

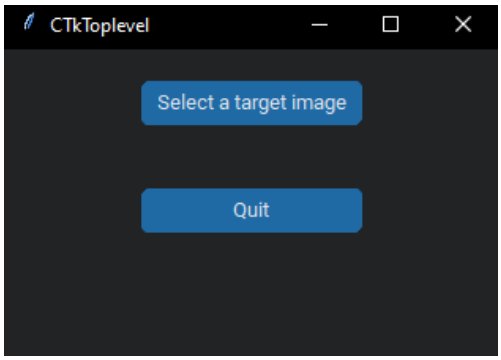


Figure 2: Second window prompting user to select target image

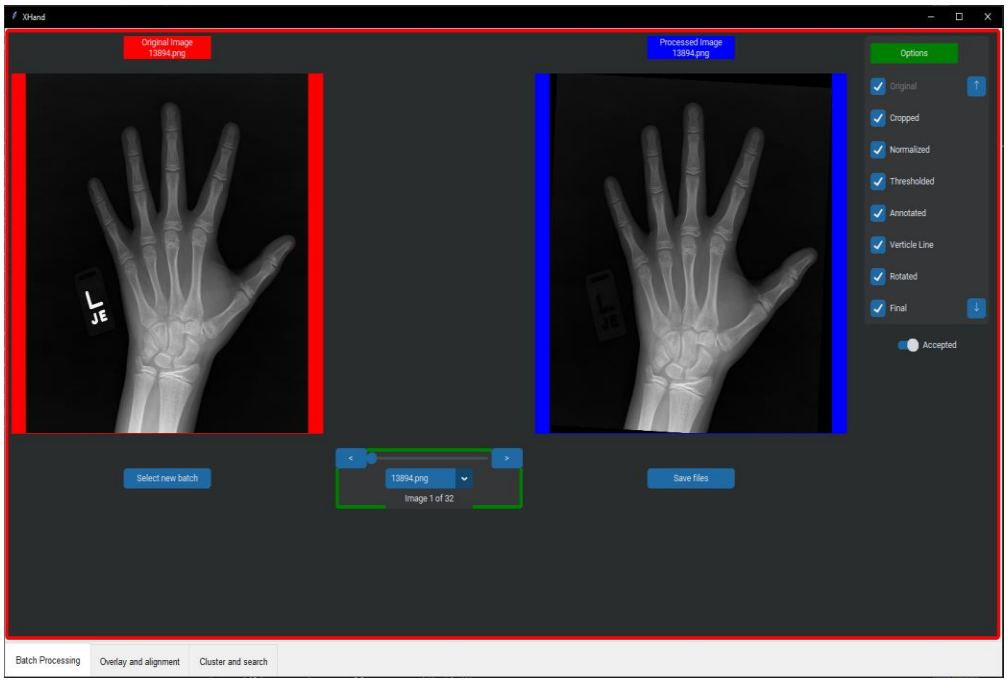


Figure 3: Batch processing window with preview and options

Figure 4: Alignment and overlay window, compared against target image

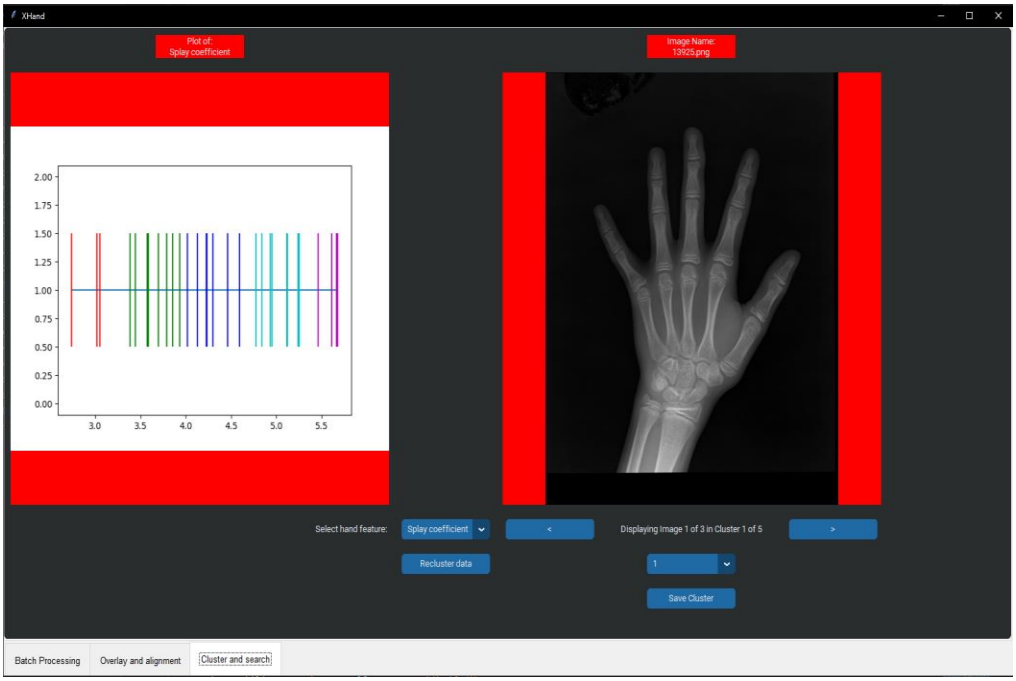


Figure 5: Image analysis and clustering window

Classes:

GUI:

The user interface is for displaying the hand images at the desired stage of the processing pipeline. The user will input a file path where the images are stored and will be able to select desired inputs which alter the hand images to their liking. It also provides functionality to overlay the images with the base image, which is considered the ideal hand image, selected by the user. At the end of the processing pipeline, the user will then be able to visualize the results from k-means clustering. The following are functions in the *GUI* class:

- ***change_children_state (****parent, new_state****)*****:**
  This function enables/disables all widgets in a container.

- ***smart_dimensions (****image_shape, label_width, label_height****)*****:**
  It resizes the image to fit within the image label but retains the original proportions.

- ***convert_image (****image****)*****:**
  Converts an image from cv Mat (numpy array) to GUI compatible form.

ImageSelectorFrame:

This is a reusable class that controls the iterating through images within a batch. It has a slider bar for sliding through the whole batch. It has left/right arrow buttons to select the previous/next image respectively. It also has an option menu where the user may select an image by its file name.

- ***__init__ (****self, container****)*****:**
  Constructor for ImageSelectorFrame object. Creates a container with a slider and arrow keys to iterate through image batches. Widgets are disabled by default in the frame.

- ***slider_func (****self, new_index****)*****:**

  Updates the index of the slider.

- ***button_func (****self, value****)*****:**

  Function called when buttons are used to iterate backwards or forward over the batch by one.

- ***optionmenu_func (****self, selection****)*****:**

  Selects a new image by the file name that the user selected

- ***update (****self, source****)*****:**

  Takes argument ‘source’- the widget that initiates the update. All other widgets will be updated to match the new selection.

- ***activate (****self, paths****)*****:**

  Function to activate the widgets within this frame and initializes variables.

- ***deactivate (****self****)*****:**

  Function to deactivate widgets within this frame

OptionFrame:

This class is the frame containing all the options for the selection features for the user to control what is displayed in the GUI.

- ***__init__ (****self, container****)*****:**

  Class constructor.

- ***check_box_func(****self****)*****:**

  Controls the cumulative selection/deselection of the option checkboxes.

- ***btn_func (****self, change****)*****:**

  Controls the stepwise selection/deselection of the option checkboxes.

- ***activate(****self****)*****:**

  Function to activate the widgets within this frame.

- ***get_selection_state (****self****)*****:**

  Returns the selection state of the object.

ClusterFrame:

This class is the frame that is intended to display k-means clustering outputs and relevant information.

- ***__init__ (****self, container****)*****:**

  Class Constructor.

- ***update(****self****)*****:**

  Updates the plot and image displayed on the screen.

- ***cluster_func(****self, selection****)*****:**

  Prompts the user to select the number of clusters they want then clusters the data and generates a plot displaying clusters in different colours.

- ***select_cluster_func****(self, selection):*

  Displays the images within a particular cluster, as selected by the user.

- ***btn_func****(self, delta):*

  Used for iterating through the images within the selected cluster.

- ***save_cluster(****self****)*****:**

  Prompts the user for a save path and then saves the images within the selected cluster.

OverlayFrame:

The frame is responsible for overlaying the selected hand images over any desired image as the base image in the batch to compare differences.

- ***__init__ (****self, container****)*****:**

  Class constructor which creates a 3x7 grid.

- ***select_target (****self****)*****:**

  Selects the desired target image and displays it in the left panel.

- ***refresh(****self****)*****:**

  Refreshes the frame to update the displayed outputs.

- ***update_image (****self, new_index****)*****:**

  Updates the image displayed to the new selected image based on the given index.

BatchSaveFrame:
This frame is responsible for saving the desired processed images from a batch to a user supplied path.

- ***__init__ (****self, container****)*****:**

  Class constructor which creates a 4x7 grid.

- ***select_files (****self****)*****:**

  Opens file selection window for the user and loads selected images into memory.

- ***save_files (****self****)*****:**

  Prompts the user to specify file path for saving images.

- ***set_switch (****self****)*****:**

  Toggle Rejected/Accepted switch on and off without triggering the command function.

- ***refresh (****self****)*****:**

  Refreshed frame to ensure all displayed information is up to date.

- ***update_image (****self, new_index****)*****:**

  Updates the image displayed on the frame based on the new index value provided.

- ***reject_switch_func (****self****)*****:**

  Allows the user to reject the images based on their preview.

Root:
This class represents the main window of the application. It contains a notebook widget which contains the frames of the application.

- ***__init__ (****self, container****)*****:**

Set the size of the window.

Notebook:
This class is a notebook widget which holds the three major frames of the application. The notebook widget allows the user to select which frame they use by clicking on the tabs at the bottom of the screen.

- ***__init__ (****self, container****):***

  Place frames into their respective tabs.

ImageController:
Wrapper class around the data structure containing ImageObj objects.

- ***__init__ (****self, container****):***

  Class constructor.

- ***load_images (****self, file_paths****):***

  Creates image objects from the list of file paths provided.

- ***length(****self****):***

  Returns the length of the images array.

- ***get_imageObj (****self, index****):***

  Returns the image object at the index provided.

- ***get_file_name (****self, index****):***

  Returns the filename of the image at the index provided.

- ***get_original_image (****self, index****):***

  Returns the original unprocessed image at the index.
- ***get_processed_image (****self, index, stage****):***

  Returns the processed image object at the index - returns the image processed up to the specified stage.

- ***get_final_processed_image (****self, index****):***

  Returns the final processed image object at the index.

- ***get_rejected (****self, index****):***

  Returns the Accepted/Rejected status of the image at the index.

- ***toggle_rejected (****self, index****):***

Toggle the state of the rejected status of the image object at the index.

- ***save_image (****self, save_path****):***

  Saves the images currently in the images array.

ImageObj:
Image object class stores information and the state of the displayed image.

- ***__init__ (****self, container****):***

  Class constructor

- ***get_file_name (****self****):***

  Returns a string of the object's filename

- ***get_original_image (****self****):***

  Returns the raw input image.

- ***get_processed_image (****self, stage****):***

  Processes and returns the image up to a specified stage.

- ***get_final_processed_image (****self****):***

  Returns the final processed image of this object.

- ***save_image (****self, save_path****):***

  Saves an image, writing the final image to the specified path.

- ***get_rejected (****self****):***

  Returns the Accepted/Rejected status of the image.

- ***toggle_rejected (****self****):***

  Toggles the image Accepted/Rejected state of this object.

- ***get_hand_points (****self****):***

  Passes the image to be processed again in its final state to find features of the hand.

- ***get_splay_coefficient(****self****):***

  Calculates the area of triangles drawn between each finger as a representation of how splayed out the hand is.

- ***get_hand_size(****self****):***

  Returns the size of the hand as the distance between the pinky tip and thumb tip.

- ***get_contours(****self****):***

  Returns the contours of the outline of the hand in the final processed image.

- ***get_contour_alignment_error(****self, target_image, option****):***

  Returns the alignment error between the contour of the current image and the specified target image using sum of squared differences. The option parameter specifies if the sum, mean or standard deviation is returned.

- ***get_fingertip_alignment_error(****self, target_image, option****):***

  Returns the alignment error between the fingertips of the current image and the specified target image using sum of squared differences. The option parameter specifies if the sum, mean or standard deviation is returned.

ImageProcessing:
This class is concerned with the main alteration and processing of images, handles all manipulation of images for the program and functions as a pipeline over 6 stages.

- ***center_crop (****img, crop****):***

  Takes an image and a crop percentage value then returns a cropped image which is still focused on the center point.

- ***prepare_image (****imageX, threshold****):***

  When provided with an image and desired threshold value this function converts the image to grayscale, blurs and equalizes it before thresholding the image.

- ***normalize_image (****cropped_input_image****):***

  Converts the image to grayscale the blurs the image slightly before normalizing it.

- ***threshold_image (****normalized_image, threshold****):***

  Thresholds the provided image based on a value provided.

- ***component_analysis (****image, raw_image****):***

  performs a connected component analysis on the *thresholded image (image)* to extract the hand for later feature detection. Returns a *component_image* of just the largest component and the size of said component.

- ***remove_clutter (****thresholded_imageX, raw_imageX****):***

  performs a connected component analysis on the *thresholded image (image)* to extract the background elements to be removed on the *raw_imageX*. Returns the now altered *raw_image* without background

- ***midpoint (****p1, p2****):***

  Given two points finds and returns the rounded to an integer value of the midpoint between them.

- ***contour_analysis (****image, draw_image, dist_threshold****):***

  Finds the contours of the image and marks key points returns on *draw_image* for display purposes later, marks convexity defects as well and then returns lists of palm points, fingertip, and fingertip end points.

- ***drawLineFinger (****drawImage, size, palm_points, fingertips, fingertips_end****):***

  Mark up the middle finger if possible by indicating the middle of the fingertip and point where it meets the palm and drawing a line between these points.

- ***rotate (****image, finger_base, finger_tip****):***

  Rotates the finger based on the angle of the middle finger.

- ***translate_image(****image, finger_base, finger_tip****):***

  Translates the middle finger to center on a point in the middle of the palm

- ***transform_image(****image, finger_base, finger_tip****):***

  Single function which passes the image to the transform and rotate functions then returns a final image.

- ***align (****image, template****):***

  Resizes images to match and overlays them with color mapping performed, returns an overlay of both images.

- ***display (****title, image****):***

  Displays the supplied image with the provided title.

- ***draw_points (****imageX, points****):***

  Draws red circles on the image provided based on coordinates from a points array.

- ***get_hand_points (****imageX****):***

  Finds and returns all significant fingertips and palm points for use in feature analysis

- ***process_image (****input, stage*=**6**, *do_remove_clutter*=**True****):***

  Takes path inputs and calls all other functions to the desired stage of output to display each step of the process individually. Returns the image at that stage of the pipeline.

FeaturePlot:
The class contains code necessary to perform *k-means* clustering to analyze the images.

- ***__init__ (****self, container****)*:**

  Class constructor.

- ***compute_kmeans (****self, data, clusters=2, max_iterations=10, epsilon=1.0****):***

  Performs k-means clustering on the supplied data with desired parameters.

- ***map_images(****self****):***
  Maps the image object to its attribute value in the cluster

- ***get_plot (****self****):***
  Returns a plot of the image data clustered into n-clusters.

- ***get_clustered_objects(self):***
  Returns the image objects in their clusters

Libraries Used:

OpenCV:

The most significant library used is Open-Source Computer Vision Library aka OpenCV (http://opencv.org) it is an open-source BSD-licensed library focused on computer vision and image manipulation algorithms. It is heavily used for feature detection and processing of the images

customtkinter:

CustomTkinter is a python UI-library that is an extension of Tkinter - a common Python GUI interface. It is used extensively in creating our GUI. CustomTkinter provides a lot of customizability to the old-looking Tkinter.

matplotlib:

matplotlib is a robust library for visualizing data and has toolkits for integrating plots into GUIs, including Tkinter.

6 Program Validation & Verification

GUI functionality tests

Functionality	**Test cases**		
	Normal Functioning	***Extreme boundary cases***	***Invalid Data (program should not crash)***
Selecting a batch of images	Multiple images selected -	Only one image selected ->	No images selected -> passed

	> passed	passed	
Saving a batch of images	All images accepted -> passed	Some images rejected -> passed	All images rejected -> passed
Screen resolution	1920x1080 -> passed	Less than 1920x1080 -> failed	Changing screen resolution -> passed
Selecting number of clusters	Valid number of clusters (0 < clusters < number of images) -> passed	Clusters = 1 or clusters = number of images -> passed	Clusters = 0 or clusters > number of images -> passed
Selecting bad images	n/a	(Batch 2) -> failed	Non x-ray hand images -> passed
Different operating systems	Windows -> passed`	Linux Ubuntu -> passed	n/a

Image processing tests

Function	**Test cases**	
	Expected Output	**Actual Output**
normalize_image	Convert image to grayscale then	Performed as expected on all images in batch 1. Some in batch 2 lack

	normalize.	sufficient contrast resulting in pure white images
threshold_image	Images are thresholded to the desired value	all images are thresholded as expected. Confirmed by visual inspection
component_analysis	Should return only the hand component of the image (the largest component)	components checked at the end can see that only a hand is extracted.
remove_clutter	Remove background without loss of hand detail.	visual inspected over batches to ensure background isolation.
drawLineFinger	draws a midline on the middle finger from the point it meets the palm to its tip	performs as expected on all images where normalizing provides enough contrast.
rotate	Rotates the image to align it vertically based on a midline of the middle finger. `	Visually appears to perform as desired in all cases.
translate_image	Moves the image to focus on a new center point in the palm	Visually appears to perform as desired in all cases.
overlay_images	displays 2 images ontop of eachother	

Featureplot tests

Functionality	Test cases	
	Expected Output	**Actual Output**
map_images	Returns a vector of images names in a cluster that was specified by the user.	Returns the expected output, which is a vector with image names that belong to a cluster.
compute_kmeans	Returns labels based on cluster value, centroids, and stacked_data	Returns expected output based
get_plot	Returns plot as an image	Returns the expected image. Visually tested.
get_clustered_objects	Returns the image objects in their clusters	Returns image objects as expected

6. Conclusion

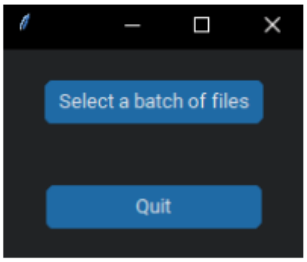
The purpose of this project was to create an application with a graphical user interface, allowing the user to select a batch of raw x-ray hand images, which are then passed through an 8-stage image processing pipeline, to prepare them for the analysis based on the hand features. The hand features were then extracted from the processed images after aligning each image with the ideal target image and were passed to the k-means algorithm to undergo clustering. The results of the k-means algorithm are then plotted, and the user will be able to display all the images in each cluster. The application met all the clients stipulated core requirements

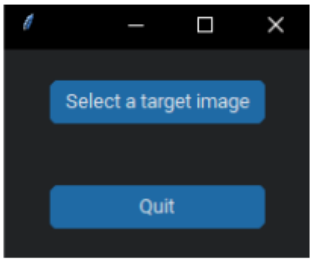
Appendix A - User Manual

XHand User Manual

Start-up:

When the program is started the user will be prompted to select a batch of images and a target image.

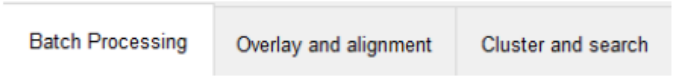




Main window:

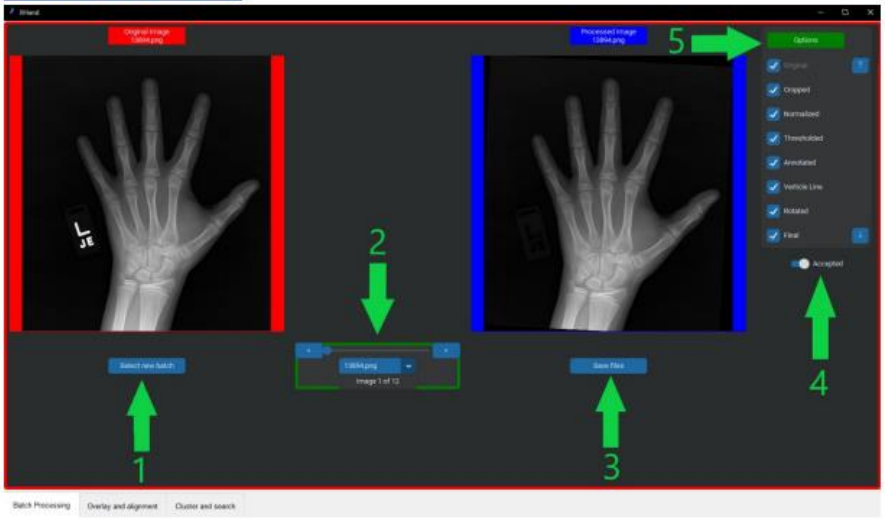
The main application window will then be displayed.

Three different tabs are displayed at the bottom of the window representing the three major use cases of the application:



1) Batch Processing
2) Overlay and alignment
3) Cluster and search

Batch Processing



The original, unprocessed image is displayed on the left panel. The processed image is displayed in the right panel.

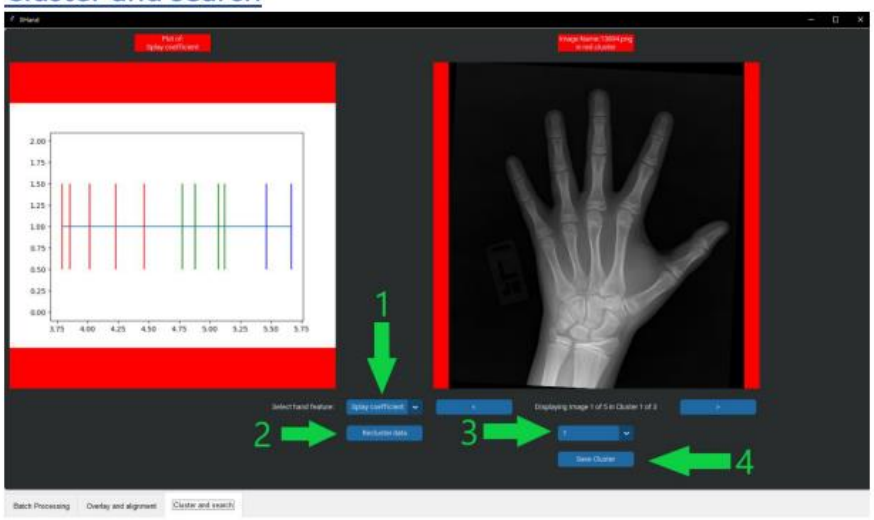
1) Select a new batch of images to be processed.
2) Iterate through your batch of images.
3) Save the batch of processed images.
4) Accept or reject an image if you are unhappy with the outcome (image will not be saved).
5) Select the various stages of processing to be displayed.

Overlay and align

The fully processed image from the batch of images is displayed on the left panel, the fully processed target image is displayed on the right panel. The images are aligned and overlayed and displayed in the middle panel.

1) Iterate through your batch of images.
2) Select a new target image to overlay.

Cluster and search



The plot of clustered data is displayed on the left panel. The images within the selected cluster are displayed on the left panel.

1) Select which attribute you would like to cluster the hand by.
2) Recluster the data with a different number of clusters.
3) Select which cluster you would like to view the hand of.
4) Save the images within the cluster (note: hands reject on the Batch Processing tab will not be save in the cluster).

Appendix A – Image Processing Pipeline Results

The idea behind the development of the image processing pipeline is that raw x-ray images need to be in a desired clean state for properly extracting valuable data needed for analysis. How well each stage performs is based on its previous stage, and because of this, proper emphasis was placed on writing code that maximizes the functionality of each stage. The sequence of the stages is shown in the following diagram:

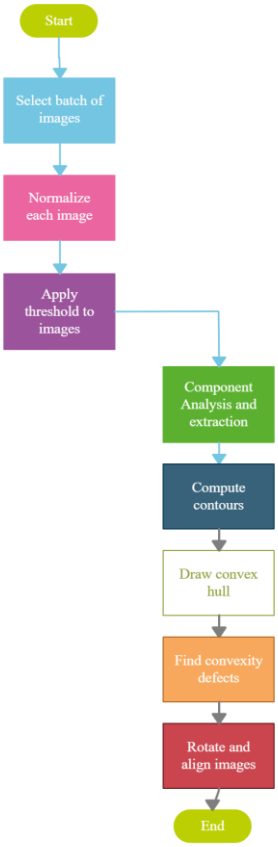


Figure 6: Image processing pipeline flowchart

1) *Histogram normalization*: this method normalizes the distribution of signal intensity values within an image. If an image has low contrast, it will make it easier to threshold. Before an image can be normalized, it must be firstly converted to grayscale.

Figure 4:Image result after Normalization

2) *Image thresholding*: It is a simple form of image segmentation. It is a way to create a binary image from a grayscale or full-color image. It helps with separating foreground pixels from background pixels. The foreground pixels which are of interest are that of the x-ray hand.

Figure 5: Image after thresholding

3) *Component Analysis and Extraction*: Is an algorithmic application of graph theory which is used to determine the connectivity of components in a binary image. This will help us extract only the hand component and not the other components which are not needed.

4) *Computing Contours*: Contours are a curve joining all the continuous points, along the boundary, having the same color or intensity. They are useful for shape analysis, which in our case is the shape of the hand. For better accuracy, they are used on binary images.

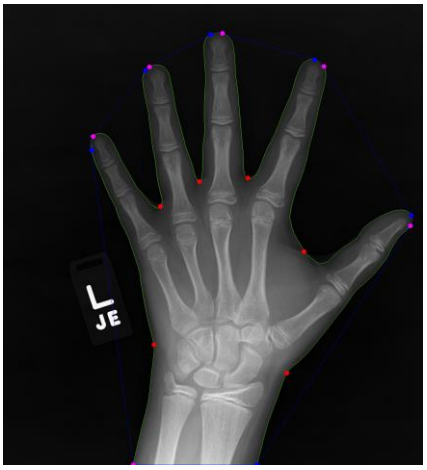


Figure 6:Image after Contour Analysis

5) *Convex hull and Convexity defects*: The convex hull helps with finding the convexity defects of the x-ray hand in that any deviation of the object from this hull can be considered as a convexity defect. Convexity defects in this case are the pixel points of the x-ray hand fingertips, which is valuable information used later in the application.

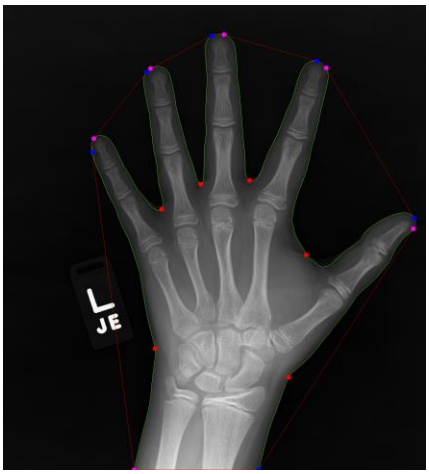


Figure 7:Image after going through the convex hull & convexity defects

6) *Drawing line on the middle finger*: drawing a line on the middle finger of each image becomes beneficial later when the images undergo rotation and alignment.

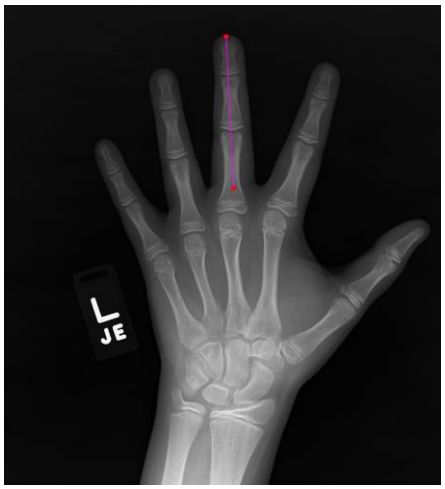


Figure 8: Hand image with drawn line on middle finger

7) *Rotating vertically*: The images are rotated so that their middle finger points upwards. The angle used to rotate the is mainly based on the original position the middle finger is pointing to.

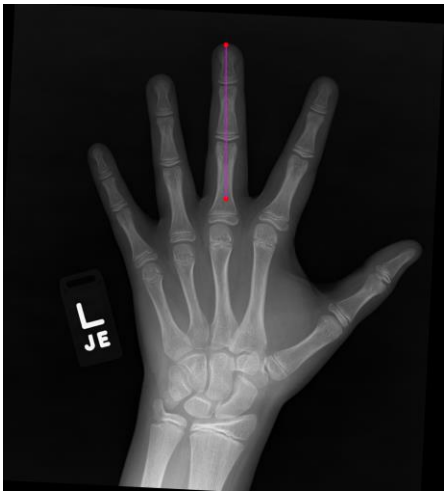


Figure 9:Rotated Image

8) *Alignment*: This is the final stage of the pipeline, and the images are aligned and overlayed with the ideal hand which is selected by the user. An alignment error is the computed which will tell how well the images aligned together. An alignment error of zero indicates that an image aligned perfectly with the ideal hand image. Different colormaps are used, with all the images having the same colormap, and the ideal hand having its own colormap. This will help the user with analyzing the overlay.

Figure 10:Ideal image aligned with an image from the batch of images