Impact Breccia Composition and Glass Clast Analysis Tool

Group Members:
Nicolas Jacobs: njacobs6@uwo.ca
Yuhan Zhang: yzha3296@uwo.ca
Yifei Zhang: yzha3744@uwo.ca

Supervisor:
Dr. Gordon Osinski
Department of Earth Sciences

CS4470Y: Software Maintenance and Configuration Management

Feb 12, 2024

Progress Report 2

1 Overall Project Description

1.1 Description

Analyzing the texture of a breccia sample formed by meteorite impacts can reveal information about how the breccia was created. When breccias form, the interior may not be uniform and will instead be filled with glass clasts (clumps of darker material). Figure 1 shows a breccia sample featuring several glass clasts. Analyzing the size, quantity, and shape of each glass clast is a very time-consuming process when performed manually, which is why geophysicists use software (e.g. ImageJ) to semi-automate this process. Our project aims to create an application that automates the process of analyzing glass clasts in recovered impact breccia samples using OpenCV, an open-source computer vision library for image processing and object detection.

Figure 1: Breccia sample featuring glass clasts.

ImageJ is an open-source scientific imaging application that is used across the scientific community. ImageJ features a wide range of image processing tools that the Earth Sciences department uses to isolate the glass clasts of a breccia image. Figure 2 shows Figure 1 after processing it through ImageJ.

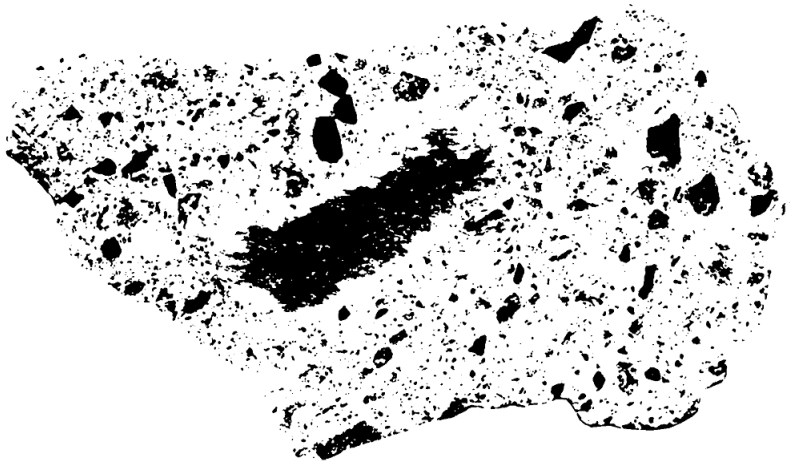


Figure 2: After sending an image into ImageJ for processing.

The main goal of our project is to improve the process of how the Earth Sciences department utilizes ImageJ to analyze breccia sample images. Dr. Osinski has identified that the main problem with the process of isolating glass clasts is the time spent manually adjusting the image after it has been processed by ImageJ. The images produced by ImageJ still require manual touchups around each

isolated glass clast. Clast edges must be adjusted, holes must be filled in, and invalid/incorrectly identified clasts must be erased; the result after applying these adjustments on Figure 2 is shown in Figure 3. These adjustments are very hard to achieve using the native ImageJ software, and could only be done either using primitive functions like paintbrush or by installing additional plugins. The lack of handy functionalities to concur these processing requirements brought our attention to OpenCV, which has a more comprehensive collection of image processing and computer vision-related tools, such as the functions for morphological transformations[1].

Figure 3: After manual adjustments of ImageJ output.

Before a sample can be processed by ImageJ, it must undergo contrast adjustments, Gaussian blur filtering, background removal, etc. After these adjustments have been made, the RGB channels are split, resulting in three monochrome images. The resulting images that best showcases the glass clasts is then chosen for further adjustments. This pre-processing pipeline is the combination of many image filters, and our project aims to automate the process of applying each pre-processing filter by developing a reliable black box approach to image pre-processing. Once image pre-processing is complete, main processing begins. In main-processing threshold values are adjusted and despeckling is applied to isolate each glass clast. Post-processing is performed once the user is satisfied with the results of the main processing. In post-processing, the user must manually adjust the edges of each clast to remove unwanted segments. Post-processing is the most time-consuming aspect of image processing, and the core goal of our project is to reduce the time spent in manual image post-processing.

The development of algorithms to achieve a better post-processing result is a major component of our project. Our algorithms would take the main-processing output, and attempt to automatically produce a final result that is as close to the result of manual adjustment as possible, many useful techniques[2] can be utilized to achieve the automation process. Our project will also provide a clean interface for researchers to use when analyzing images. In addition to improving the image processing pipeline, our project will also create tables, graphs, and charts to automatically present data gathered from the finalized image.

1.2 Context Diagram

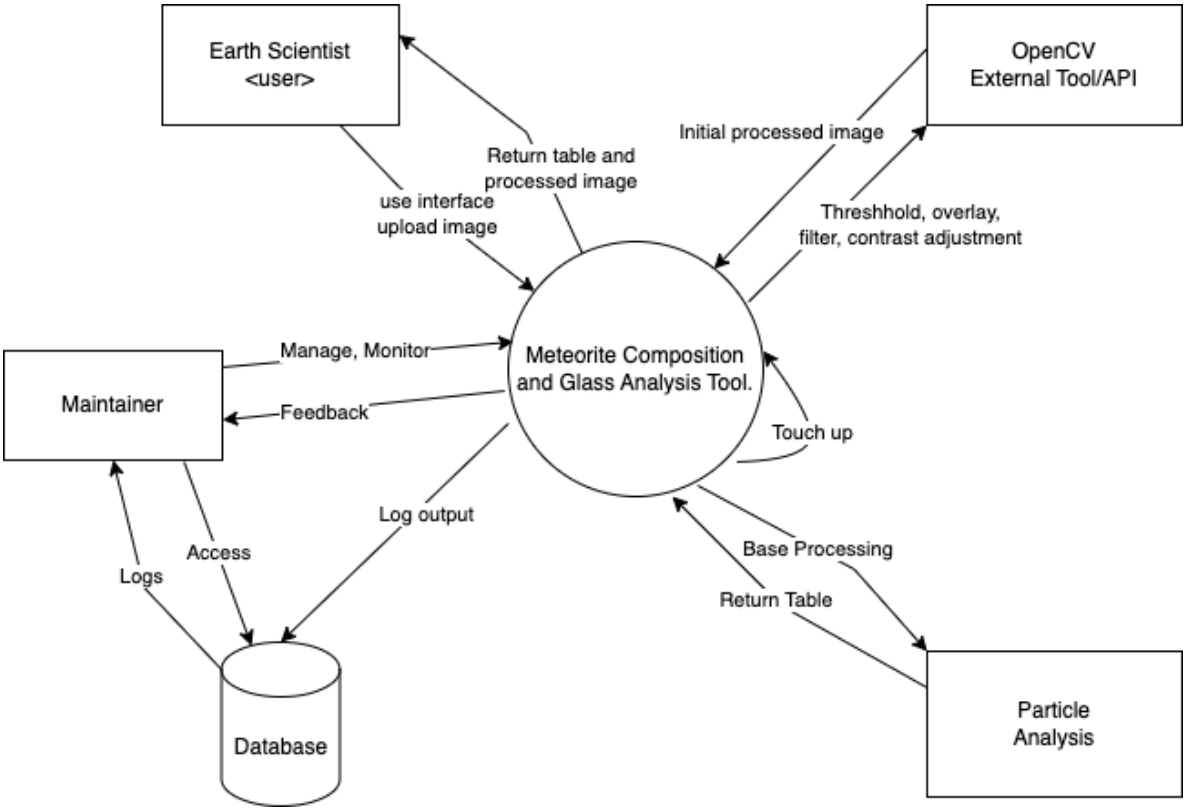


Figure 4: Impact Breccia Composition and Glass Clast Analysis Tool Context Diagram

2 Project Goals and Objectives

2.1 Goal

To develop a system that streamlines the process of identifying glass clasts in impact breccias through image processing and object detection using OpenCV, and delivering the result image and data associated with the sample.

2.2 Objectives

- O1: Collaborate with Dr. Osinski and his students to refine the system based on their practical use cases and requirements.
- O2: Implement functions for pre-processing adjustment of filtering: blur, contrast, background subtraction, and RGB split of input image using OpenCV.
- O3: Implement functions for main-processing adjustment: threshold and despeckling of rock sample images using OpenCV.
- O4: Develop algorithms for post-processing of rock sample images (remove outliers, image overlay, touch up, set scale) to automate the process using OpenCV.
- O5: Integrate the developed functions and algorithms into a desktop application that is available on Windows.
- O6: Implement a clean interface for users to navigate between processes and adjust the parameters of certain algorithms.
- O7: Design a pipeline for manual and automated image processing with intermediate file saving.
- O8: Implement error-checking mechanisms within the application to identify and handle potential issues during the image processing pipeline.
- O9: Conduct thorough testing and validation of the automated image adjustments to ensure their effectiveness across a diverse set of breccia samples.
- O10: Optimize the algorithm's performance to achieve real-time or near-real-time processing speeds for.
- O11: Provide comprehensive documentation for users, including tutorials and guidelines for effectively utilizing the desktop application and its features.
- O12: Distribute the system to users outside of the Western Earth Sciences department.

2.3 Significance

The process of analyzing the glass clasts in impact breccia samples is important and time-consuming. Current approaches use ImageJ to semi-automate the analysis process, however, several hours of manual adjustments are still required afterward to produce a usable image. Our project would significantly reduce the time spent by geophysicists editing images and would create results that are more accurate than current software. Dr. Osinski is interested in releasing our system to the public, which would allow our work to benefit the broader scientific community.

3 Roles

Project Manager	**Nicolas Jacobs**
Liaison	**Nicolas Jacobs**
Programmer	**Nicolas Jacobs** **Yuhan Zhang** **Yifei Zhang**
Algorithm Design	**Yifei Zhang** **Yuhan Zhang** *Nicolas Jacobs*
Documenter	**Nicolas Jacobs** *Yifei Zhang* *Yuhan Zhang*
Quality Analysts	**Yuhan Zhang** *Yifei Zhang* *Nicolas Jacobs*
UI\UX design	**Nicolas Jacobs** *Yuhan Zhang* *Yifei Zhang*

NOTE: **Lead** *secondary*

4 System Design

4.1 Component Information

Our system is being designed to be an improvement on the application ImageJ. Components that Dr. Osinski and other Earth Scientists use in ImageJ are being recreated in our system, with improvements and extra features being included. Each component is being newly created for our system with the help of the OpenCV library.

Features of ImageJ that are being recreated for our system:

- **Gaussian blur:** The user can apply a Gaussian blur to the image.
- **Contrast adjustment:** The user can adjust the contrast of their image.
- **Subtract background:** The user can remove the background from their image.
- **Split RGB channels:** The user can split their image into the R, G, or B colour channel.
- **Threshold:** The user can adjust the threshold of their image to remove pixels below/above a specific value of greyscale.
- **Despeckle:** The user can remove small sections of pixels scattered around the image that shouldn't be included in the final output.
- **Outlier removal:** The user can remove pixel values beyond a specific greyscale range.
- **Overlay of original image:** The user can toggle an overlay of the original image on top of their current image.
- **Eraser:** The user can manually erase pixels from the processed image.
- **Pencil:** The user can manually add pixels to the processed image.
- **Fill bucket:** The user can select an area surrounded by pixels and the interior area will be filled in with pixels.
- **Erase bucket:** The user can select an area to erase and all pixels touching this section will be erased from the image.
- **Scale/resizing:** The user can adjust the size of the image file.

- **Exporting:** The user can export the image into a jpg/png/tif file.
- **Zooming:** The user can zoom in and out on the image being processed.
- **Pan:** The user can pan the image, which is primarily used when zoomed in.

Features of our system not included in ImageJ:

- **Auto hole filler:** The system can automatically fill in holes that it finds in the image.
- **Drag and drop:** The user can drag and drop an image into the application to begin processing it.

Most of the components being taken from ImageJ can be recreated with the OpenCV library. All components of our system have existed in some form before our proposal, as each is a common image adjustment filter. We plan to implement each component with the OpenCV library when possible, but we will create components from scratch when otherwise required.

The open source software ImageJ is being referenced throughout development as a baseline for our system's functionality. ImageJ is what Dr. Osinski and his colleagues currently use to adjust impact breccia samples with, so we are comparing our system to ImageJ to ensure that we meet or outperform ImageJ's capabilities and ease of use.

Our system's GUI is being developed from scratch using the JFrame library. JFrame presents a clean interfaces for buttons, text inputs, image display, etc. which is why we have chosen it as the library to create our GUI with.

4.2 Functional decomposition

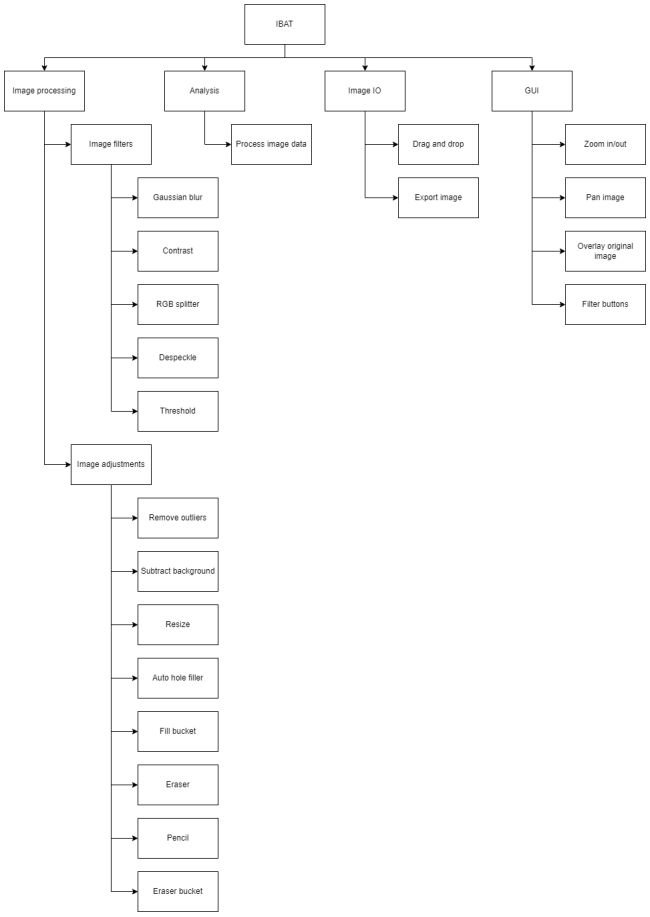


Figure 5: Functional decomposition of the program.

- **IBAT:** The application being built.
- **Image processing:** Modifications being made to the images input to the program.
- **Image filters:** Image processing algorithms that specifically manipulate the sample as a whole.
- **Gaussian blur:** Applies a blur to the image, adding fuzziness which helps with further filters.
- **Contrast:** Boost the contrast of the image, which reduces pixel variety.
- **RGB splitter:** Split the image into the R, G, and B channel. The user picks whichever channel best showcases the breccia sample and selects it for further processing.
- **Despeckle:** Removes small regions of pixels that are left over from other filter algorithms.
- **Threshold:** Adjusts the max and min pixel greyscale values of allowed pixels, removing all other values.
- **Image adjustments:** Algorithms and manual adjustments that happen to the mostly processed image. These are primarily touchups.
- **Remove outliers:** Removes outlier regions of pixels beyond a certain range.
- **Subtract background:** Removes the background of the image, leaving only the sample itself.
- **Resize:** Adjust the size of the image. Note: this is not the same as zooming in or out.
- **Auto hole filler:** Automatically detect any regions of the image that are holes inside of a larger clump of pixels and fill them in.

- **Fill bucket:** Allows the user to manually fill in holes.
- **Eraser:** Erase the region of pixels selected by the user.
- **Pencil:** Draw new pixels to the screen in the area selected by the user.
- **Eraser bucket:** Erase all pixels touching the area selected by the user.
- **Analysis:** After image processing is complete, the program can apply analysis on the sample and print results about the sample.
- **Process image data:** Each specific form of sample analysis is still under consideration, as these are a "wish list" feature.
- **Image IO:** The program can receive an image as input from the user and can export the processed image into a variety of image formats.
- **Drag and drop:** The user can drag and drop their image into the program to begin processing it.
- **Export image:** The application can export the processed image into a jpg/png/tif format.
- **GUI:** The program has a GUI that allows the user to view and interact with the image, as well as press buttons which allow them to perform image processing.
- **Zoom in/out:** The user can use their scrollwheel to zoom in and out on the image.
- **Pan image:** The user can pan the screen to view different parts of the image. This is best used when the image is zoomed in on.
- **Overlay original image:** The user can press tab to toggle an overlay which allows them to view the original unprocessed image ontop of their filtered image.
- **Filter buttons:** Each filter that the program can perform for image processing can be accessed from a button on the main screen.

4.3 Sequence diagram

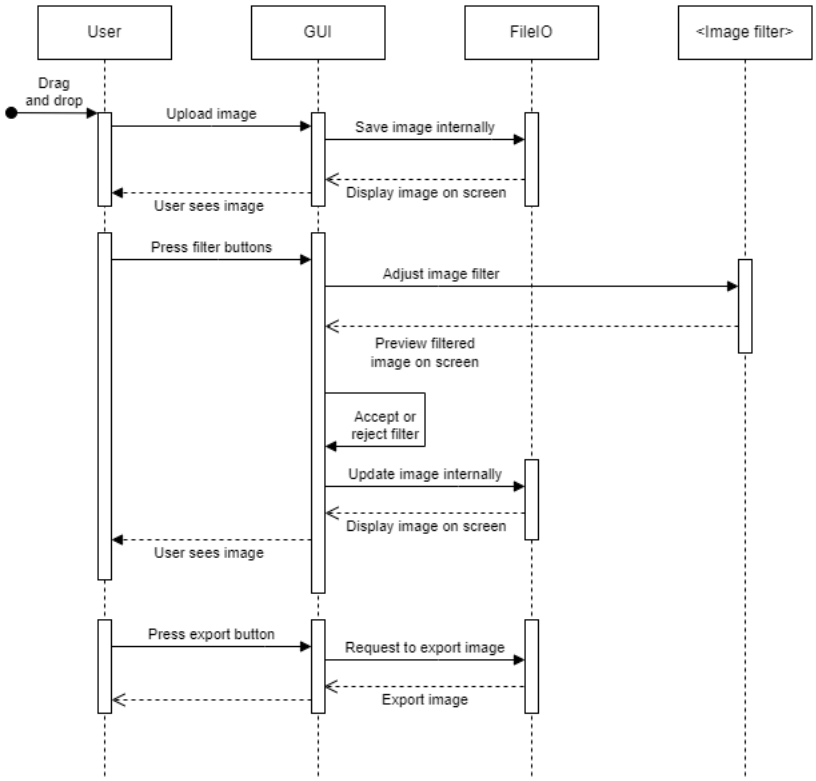


Figure 6: Sequence diagram of the program.

- **User:** The user is able to interact with the system by dragging and dropping image samples into the GUI. They can press different buttons on the GUI to apply filters to their image.
- **GUI:** The GUI allows the user to view their image and to press different buttons to apply different filters to their image. Whenever a filter is applied to an image, the change is previewed to the user, where they can apply the filter, adjust its settings, or cancel. When the user wants to apply the filter to the image, they confirm their choice to the GUI and the internal image is updated.
- **FileIO:** The system is able to import and export images. There is an internally saved copy of the user's image, with any applied filters saved to it. Whenever the user requests, they can export their image in a variety of file types.
- **<Image filter>:** This represents any image filter being applied to the user's image. All of them interact with the user and GUI the same way, so they've been grouped together here to reduce redundant information. An image filter applies a specific filter to the current internal image. The filter will preview its changes to the user before being saved.

4.4 Component and connector diagram

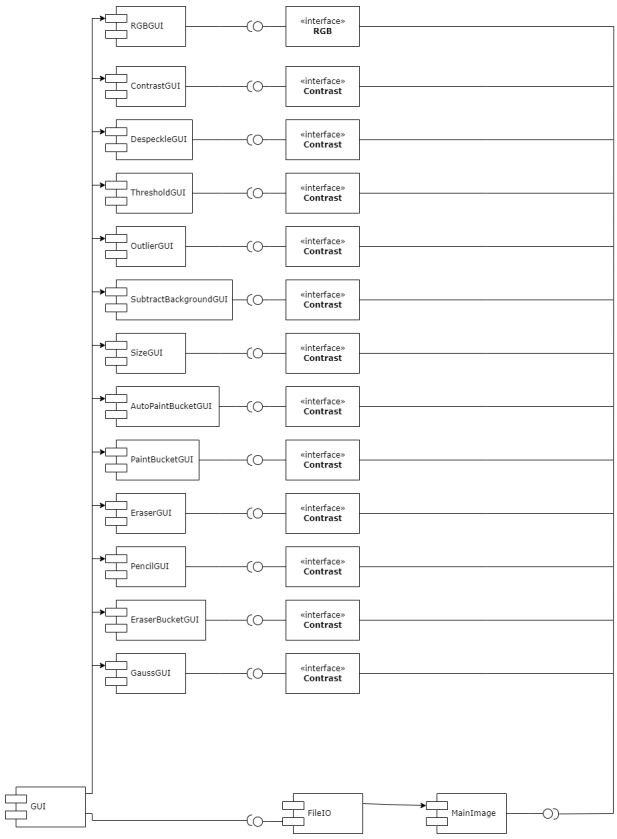


Figure 7: Component and connector diagram of the program.

The user interacts with the GUI, which they can use to drag and drop an image into the main area to begin processing. Until the user has imported an image into the program, no component has any functionality.

Once an image has been imported into the system, the user can press a button to begin processing the image. Each button presents a new interface to the user, which is controlled by the sub-GUI class (RGBGUI, ContrastGUI, GaussGUI, etc.) Each of these GUI classes interfaces with its own respective class of algorithms and functions that controls the functionality of each filter.

The program is organized this way so that the main GUI class only has to call the creator of each sub-GUI, and each respective creator can handle its own implementation.

4.5 Interface

There are two sections to the application. The main panel is where the image is rendered. Here, the user is able to zoom in and out and pan the image. The user can press tab to toggle an overlay of the original image when a filter is being previewed. Along the top of the GUI is a row of several buttons that each perform a specific utility. The layout has been chosen for ease of use in mind, and we will be collaborating with Dr. Oskinski and his colleagues to create an optimal button layout. Figure 8 shows the current build of the GUI.

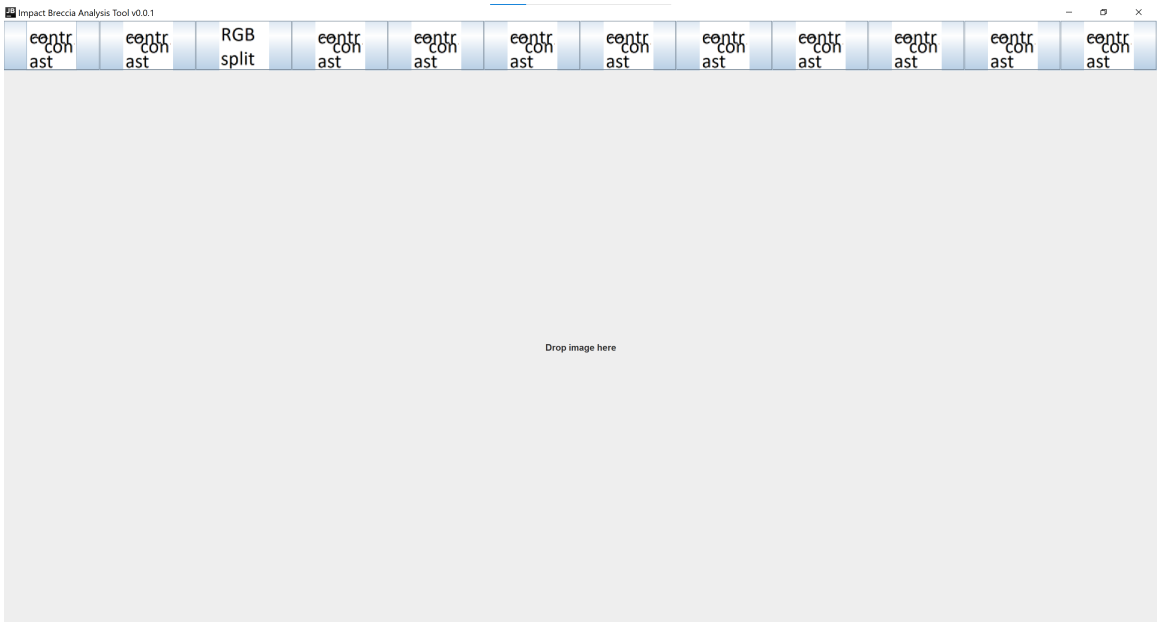


Figure 8: Current iteration of the application's GUI. The RGB splitter component is the only function that has been properly connected to the frontend. All button icons are placeholders.

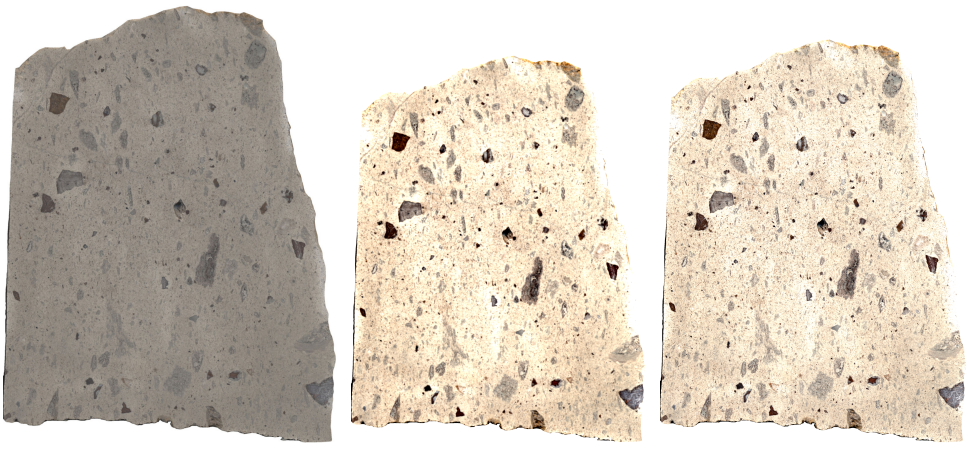
5 System Testing

Our application is an image processing program for impact breccias, meaning that our test cases are image samples of impact breccias.

5.1 Test Case 1

This is testing the application's ability to adjust the contrast of the image. The test will be a success if the contrast adjustment algorithm is able to properly adjust the contrast according to the adjustment values inputted into the function.

Feature 1: Contrast adjustment algorithm



(a) (b) (c)

Figure 9: (a) Original image (b) Expected result (c) Actual result

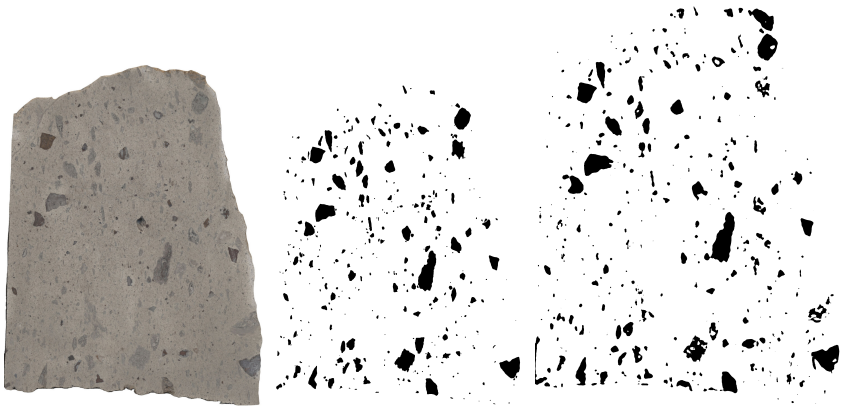
Status: Test Case 1 passed

5.2 Test Case 2

This is testing the background subtraction of our application, as well as its despeckling functionality. This test is designed to compare our algorithm's functionalities to the ones provided by ImageJ. This test will pass if the algorithm is able to perfectly replicate the image produced by ImageJ's despeckling and background subtraction algorithms.

Feature 9 : Background subtraction algorithm

Feature 6 : Despeckling component



(a) (b) (c)

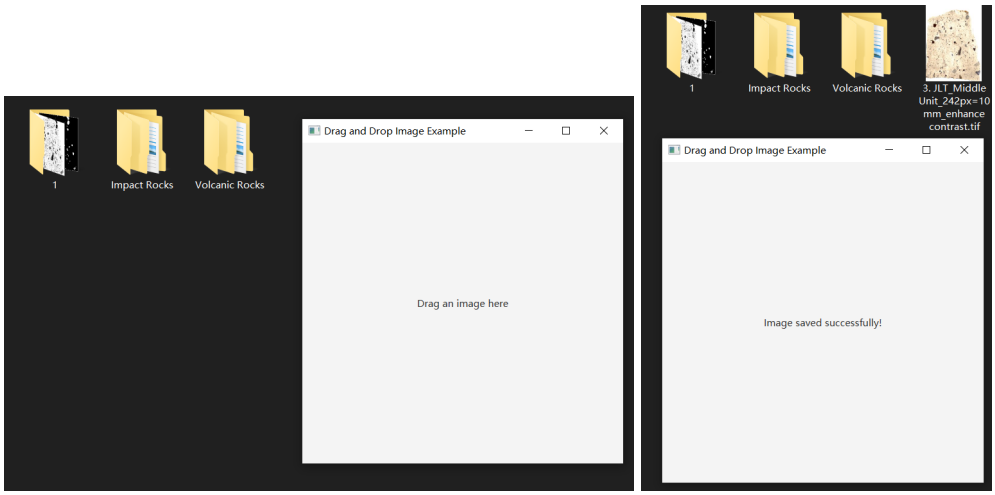
Figure 10: (a) Original image (b) Expected result (c) Actual result

Test Case 2 failed; the despeckle parameter value selection likely caused this issue. Although the results seem similar, there are too many minor differences between our result and ImageJ's result. These differences are important because the despeckling algorithm must be highly accurate. We'll adjust and select a better parameter range to adjust this feature. In addition, we may add a touch-up function to remove the unnecessary elements

5.3 Test Case 3

This is a test of the program's ability to export images. The test will be a success if the program is successfully able to export its current image to the user's computer.

Feature 7 : Image IO



(a) (b)

Figure 11: (a) before saving (b) after saving

Status: Test Case 3 passed because the image was successfully exported to the user's computer.

5.4 Test Case 4

This test case is a test of the system's ability to add a Gaussian blur to the user's image. The test will be a success if the algorithm is successfully able to add blur to the image.

Feature 2 : Gaussian blur filter

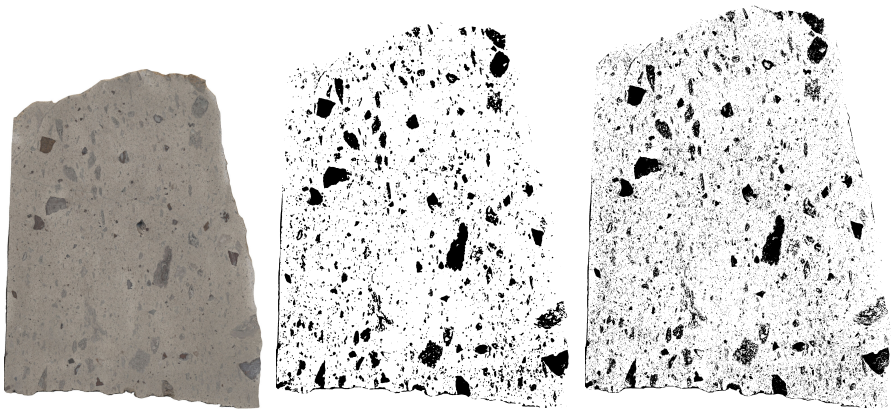
(a) (b) (c)

Figure 12: (a) Original image (b) Expected result (c) Actual result

Status: Test Case 4 passed. The application was able to apply the Gaussian blur effect to the image sample.

5.5 Test Case 5

This test case is to verify that the system is capable of applying the threshold algorithm correctly. It will be a success if the algorithm is able to accurately apply the algorithm for the given input values.
Feature 6 : Thresholding



(a) (b) (c)

Figure 13: (a) Original image (b) Expected result (c) Actual result

Status: Test Case 5 passed. The output matches the expected results, meaning that it can accurately apply threshold adjustments to given images.

6 System Requirements

See System_Requirements_2.xlsx document

7 Project Plan and Tracking

7.1 Chart

See PR2_Project_Plan_and_Tracking_Chart.pdf for an overview of the full sprint plan.
See Sprint_Timeline.png for a detailed view of the remaining sprints.
All diagrams can be found in a respective png file associated with this report.

7.2 Spreadsheet

See PR2_Project_Plan_and_Tracking_Spreadsheet.xlsx

References

[1] OpenCV. (2024). "Morphological Transformations," [Online].Available: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html. Accessed: 10/02/2024.

[2] Martínez, A. B. (2021). Computer Vision system for rock classification using Artificial Neural Network. https://uvadoc.uva.es/handle/10324/48724

[3] George D. Greenwade. *The Comprehensive Tex Archive Network (CTAN)*. TUGBoat, 14(3):342–351, 1993.

[4] Alexis David Pascual. *Autonomous and Real-Time Rock Image Classification using Convolutional Neural Networks*. Electronic Thesis and Dissertation Repository, 2019, No. 6059. Available at: https://ir.lib.uwo.ca/etd/6059.

[5] R. J. Lisle, P. Brabham, J. Barnes. *Basic Geological Mapping*. 5th edition. Wiley-Blackwell, 2011.