Impact Breccia Composition and Glass Clast Analysis Tool

Project Report Final

Group IBAT

Nicolas Jacobs

Yifei Zhang

Yuhan Zhang

CS4470Y Software Maintenance & Configuration Management

Department of Computer Science

Western University

April 1, 2024

Project Supervisor: Dr. Gordon Osinski, Dept. of Earth Sciences

Course instructor: Prof. Nazim Madhavji, Dept. of Computer Science

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

Glossary of Terms and Definitions

Glass clast: Regions of darker material inside an impact breccia.

Image filter: Any singular filter being applied to an image.

Impact breccia: A type of impactite, forms during the process of impact cratering when large meteorites or comets impact with the Earth or other rocky planets or asteroids.

UI/GUI: A user interface (UI) is the space where interactions between humans and machines occur.

API: Application Programming Interface, a way for two or more computer programs or components to communicate with each other.

MVP: Model–view–presenter, a derivation of the model–view–controller architectural pattern and is used mostly for building user interfaces.

The application/IBAT: The Impact Breccia Analysis Tool. The system that we have developed for Dr. Osinski.

RGB: The RGB color model in which the red, green and blue primary colors of light are added together in various ways to reproduce a broad array of colors.

OpenCV: Open Source Computer Vision Library.

OpenCSV: An open-source CSV (comma-separated values) parser library for Java

ImageJ: The current application used by Dr. Osinski and other researchers to isolate glass clasts in impact breccia images.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

Structured Abstract

- **[Context and motivation]**: The project addresses the need for automated and efficient analysis of glass clasts in impact melt-bearing breccias within the field of Earth Sciences. Traditional methods, such as manual analysis using ImageJ, are time-consuming and prone to errors, hindering research progress in understanding impact events and geological processes.
- **[Question/problem]**: The project aims to develop a system that automates the analysis of glass clasts in impact melt-bearing breccias, improving efficiency and accuracy compared to manual methods.
- **[Principal ideas/solution/results]**: The developed system, IBAT (Impact Breccia Analysis Tool), utilizes image processing techniques and automation to streamline the analysis process. It incorporates features such as background subtraction, channel splitting, and thresholding to identify and analyze glass clasts. The system is implemented in accordance with the Model-view-presenter (MVP) architectural pattern. The system includes components for image preprocessing, segmentation, and particle analysis developed using OpenCV libraries and UI component developed using Java Swing. The development approach followed a SCRUM methodology, with separate teams focusing on UI design and algorithm development.
- **[Contribution]**: The main contribution of the project lies in providing a reliable and accessible tool for researchers in Earth Sciences to analyze glass clasts in impact melt-bearing breccias more efficiently. By automating the analysis process, IBAT enables broader and more consistent studies of impact events, leading to valuable insights into geological implications.
- **[Limitations]**: While IBAT offers significant improvements over manual methods, its effectiveness may be limited by the quality of input images and the complexity of breccia samples. Additionally, the system's performance may vary depending on the user's familiarity with the interface and image processing techniques.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

Table of Contents

Glossary of Terms and Definitions 2
Structured Abstract 3
Table of Contents 4
01: Introduction 5
02: Background and Related Work 6
03: Concepts, Equations, etc. 8
04: Development Objectives 9
05: System Requirements 10
06: Development Strategies 17
07: Results — solution created 18
08: Discussion 23
09: Conclusions 24
10: Future Work and Lessons Learnt 25
11: Acknowledgements 26
12: References 27
13: APPENDIX A (Section 7. Results - System Validation) 28

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

01: Introduction

Analyzing the texture of breccia samples formed by meteorite impacts can reveal the information on how breccia is formed. When breccia forms, it may not be uniform inside; Filled with glass shards (clumps of dark material). Initially, the study of these geological formations relied heavily on manual inspection and analysis, a process that was not only time-consuming but also prone to human error and bias. Early efforts to classify and understand the texture, grain size, and shape descriptors of rocks manually limit the precision and comparability of results from different studies.

The key technology currently employed in this area to perform semi-automatic digital image analysis is ImageJ. ImageJ is essential for distinguishing between different components of the breccias based on macroscopic interpretations, including contrast and color variations. This process is important for the semi-automatic estimation of areal fractions and modal abundances of components within the samples.

The field of geosciences has leveraged digital image analysis extensively to enhance the understanding and characterization of impact breccias. Tools like ImageJ have been pivotal, enabling researchers to conduct detailed textural studies of impact melt-bearing breccias. Studies utilizing these tools have significantly contributed to the current understanding by providing methodologies for estimating areal fractions of components, measuring clast sizes, orientations, and other vital parameters.

One of the significant gaps in the current approach is the reliance on outdated technology and tools, which do not fully show the potential of automation in analyzing samples. The ImageJ software, despite its extensive use, lacks certain functionalities required for efficient and accurate analysis. These include advanced despeckling techniques and the ability to automatically "fill in the holes" in the isolated glass clasts. Researchers often resort to primitive functions like the paintbrush or the installation of additional plugins to accomplish these tasks.

In addition, the requirement for extensive manual operation in the post-processing phase, where researchers must adjust the edges of each class manually, introduces the potential for bias and inconsistency, making the process extremely time-consuming.

Our project addresses these limitations by proposing the development of an application that leverages OpenCV, an open-source computer vision library known for its comprehensive collection of image processing and object detection tools. Our aim is to automate the analysis of glass clasts in recovered impact breccia samples, significantly reducing the need for manual intervention.

By developing algorithms that can automatically perform tasks such as background subtraction (Martínez, A.B. 2021), image thresholding (OpenCV, Image Thresholding, 2024) and morphological transformations (OpenCV, Morphological transformations, 2024), our project seeks to minimize the time spent in manual post-processing. We intend to automate the pre-processing pipeline entirely, applying each filter seamlessly to prepare the sample for analysis. This approach will streamline the entire process and aims to produce results that closely match those of manual adjustments. The project will also offer a user-friendly interface for researchers to analyze images efficiently.

By addressing the gaps in current methodologies with innovative solutions, our project will offer a more accurate, efficient, and less labor-intensive approach to studying impact melt-bearing breccias.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

02: Background and Related Work

- **Project background**:

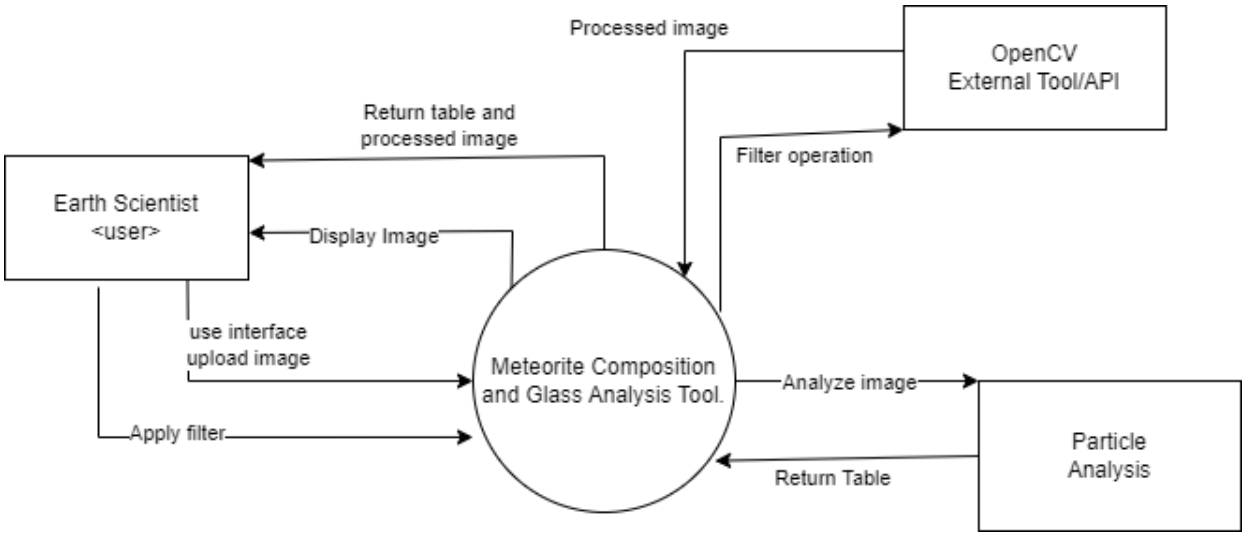


Figure 1: IBAT context diagram

With IBAT, **Earth Scientists** would only need to input a cross-sectional scan of the impact breccia, **IBAT**'s User Interface (UI) provides them with functionalities in the backend to apply varies processing filters on the original image by calling the corresponding **OpenCV** Application Programming Interface (API). **IBAT** also utilizes OpenCSV in the **Particle Analysis** component to generate table for analysis report.

Old System Architecture:

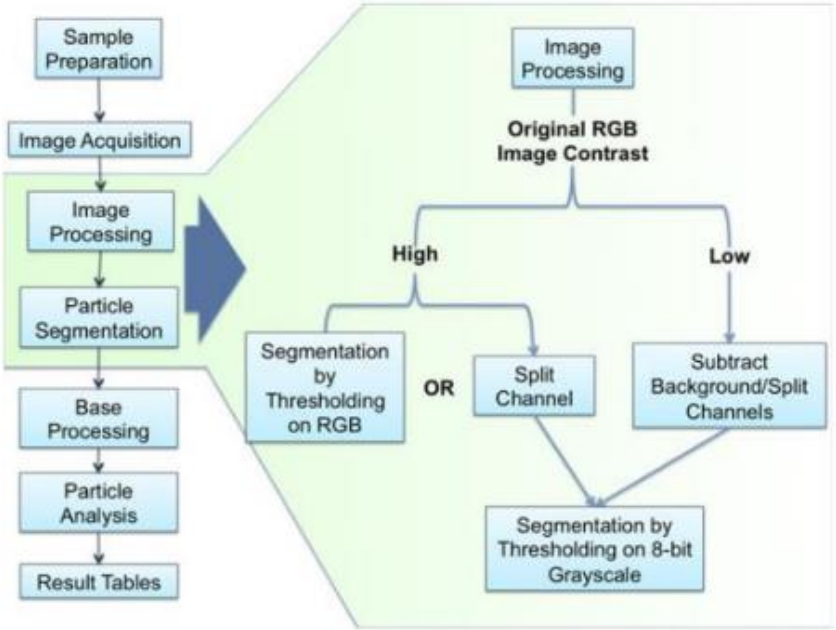


Figure 2: Semi-automatic ImageJ analysis flowchart (Chanou, A., et al., 2014)

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

Motivation for Change:

- Heavy reliance on physical examination and manual measurements, leading to increased time and potential for inaccuracies.
- Automated processes reduce time spent on manual adjustments and measurements significantly.
- Accuracy: Digital enhancements minimize human error and increase the reliability of data.
- Automated methods ensure consistency in analysis across different samples and studies.
- Digital methods make it easier for researchers to share methodologies and findings, fostering collaboration.

Related Literature:

Chanou et al. (2014) introduced a semi-automatic digital image analysis method tailored for the structural comparison of impact-melted breccias using ImageJ. This method automates significant portions of the image analysis workflow, including the estimation of area fractions of components and measurements of clast size and shape complexity, aiming to process larger datasets with higher efficiency compared to manual techniques. By minimizing manual intervention, the semi-automated approach notably reduces the potential for subjective bias, contributing to more objective and reliable analysis results.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

03: Concepts, Equations, etc.

- A digital image requires a filetype (jpeg, png, tif, webp, etc.) to be stored on a computer. These filetypes are primarily used for compression, meaning that the uncompressed data (the image) is the same regardless of its filetype. IBAT was developed to use the most common filetypes and functionality is not restricted to a single filetype.

- Uncompressed images are stored using RGB (red-green-blue). Three values from 0-255 are used to represent the colour of a pixel. We can apply filters to each image to adjust the image's contrast, brightness, hue, etc. by applying functions to the RGB values of each pixel of the image.

- A grayscale image is one in which the value of each pixel is a single value from 0(black) to 255(white); that is, it carries only intensity information. Grayscale images, a kind of black-and-white or gray monochrome, are composed exclusively of shades of gray. The contrast ranges from black at the weakest intensity to white at the strongest. ("Grayscale," 2024)

- Filter parameters are the specific values used to tweak a filter. For example, when adjusting the contrast, three parameters are used to determine the RGB value of the resulting pixel (OpenCV, Changing the contrast and brightness of an image! 2024):

$$O = \left(\frac{\alpha \times I + \beta}{255}\right)^{\tau} \times 255$$

  In this case, the alpha, beta and gamma values control contrast, brightness and Gamma correction respectively.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

04: Development Objectives

O1: Implement the following processing functionalities: Gauss blur, contrast, background subtraction, colour inversion, RGB channel splitting, threshold, erosion/dilation, particle filler, particle remover. These will create an application that dramatically simplifies the process of isolating glass clasts.

O2: Create a clean and understandable GUI for the application. The user should be able to drag and drop their image into the application to begin processing. The GUI should allow the user to view the image as it is being processed, zoom in, zoom out, pan, and view an overlay of either the original image or the previous filter. Each filter should be accessible via a button or hotkey. A good GUI will reduce the time spent creating each image when compared to the current ~9 hour manual process using ImageJ.

O3: Allow the user to freely apply each filter to the image with the ability to adjust each filter's parameter(s). The application should also be capable of automatically applying each filter dynamically via a pipeline. The automatic pipeline should automatically detect the optimal value for each filter parameter, and these optimal values should be accessible to the user during manual processing. The user should be able to undo or redo an applied filter. Granting researchers the ability to manually and automatically process images will improve its usability.

O4: Implement particle analysis function for the resulting binary image, it should include analysis for common properties and measurements for the shape of the glass clasts, these include area, center of mass, perimeter, shape descriptor, etc. The particle analysis function provides significant utility to researchers interested in the properties of the glass clasts in impact breccia.

O5: Collaborate with researchers to refine the application based on actual use cases and requirements. The application will be highly accurate, reducing the time and effort required to isolate glass clasts.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

05: System Requirements

Requirement #01 — *Contrast filter*

- FR1: The filter can adjust the contrast of an image.
    - FR1.1: The filter can automatically detect the optimal contrast adjustment parameter.
- FR2: The filter can adjust the brightness of an image.
    - FR2.1: The filter can automatically detect the optimal brightness adjustment parameter.
- FR3: The filter can adjust the gamma of an image.
    - FR3.1: The filter can automatically detect the optimal gamma adjustment parameter.
- QR1: The filter only allows the user to adjust the contrast within a predetermined range of values. This range spans the optimal values to adjust the contrast by for an impact breccia image.
- QR2: The filter only allows the user to adjust the brightness within a predetermined range of values. This range spans the optimal values to adjust the brightness by for an impact breccia image.
- QR3: The filter only allows the user to adjust the gamma within a predetermined range of values. This range spans the optimal values to adjust the gamma by for an impact breccia image.

Requirement #02 — *Gauss blur filter*

- FR1: The filter can apply Gaussian blur to an image.
    - FR1.1: The filter can automatically detect the optimal kernel parameter for applying a Gaussian blur.
- QR1: The filter only allows the user to adjust the kernel within a predetermined range of values. This range spans the optimal values to apply a Gaussian blur for an impact breccia image.
- QR2: The interface shown to the user should simplify the process of choosing a kernel parameter. Kernels must be odd numbers, so instead of requiring the user to input an odd number, the user can input the kernel's index instead. Example: $1 \rightarrow 1$, $2 \rightarrow 3$, $3 \rightarrow 5$, $4 \rightarrow 7$, etc. This is done to abstract away the knowledge of how Gaussian blur is applied, simplifying the process for the user.

Requirement #03 — *Background subtraction filter*

Impact breccia images are typically taken using a high-resolution camera against a plain background. During processing, the filters expect every background pixel to be completely white with absolutely no trace of colour, whereas the raw imported image will be shades of some colour. Thus, an automated background subtraction algorithm is required to apply other filters.

- FR1: The filter can detect and remove the background of an image.

Requirement #04 — *RGB channel splitting filter*

The result of image processing is a binary image where each pixel is either black or white. Most of the filters work by refining a greyscale image down to a binary black and white image, however, the initially imported image is in full colour. A colour image is made of three colour channels, red, green, and blue. These channels combine to create a full colour image, however, each channel can be

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

interpreted individually as a greyscale image. The RGB channel splitting filter splits a full colour image into three greyscale images and then selects the best one for future image processing.

- FR1: The filter can split an image into its three colour channels.
- FR2: The filter can automatically detect the optimal colour channel to continue processing.
- QR1: The filter returns a pseudo-greyscale image instead of a pure greyscale image. A pure greyscale image would only use one channel to represent the brightness of the pixel, however, this filter should use all three RGB channels. For example, a pixel with brightness 51 would be represented as 51 51 51 (representing the red, green, and blue channels) instead of simply 51. Although this is suboptimal, it removes restrictions that would otherwise be placed on the other filters. Other filters would otherwise need to be designed to work with pure greyscale images as well as RGB.

Requirement #05 — *Colour inversion filter*

- FR1: The filter can invert the colour of an image.

Requirement #06 — *Threshold filter*

- FR1: The filter can adjust the threshold of an image.
    - FR1.1: The filter can automatically detect the optimal threshold parameter for adjusting the threshold.
- QR1: The filter only allows the user to adjust the threshold within a predetermined range of values. This range spans the optimal values to adjust the threshold by for an impact breccia image.

Requirement #07 — *Despeckle filter*

Despekling is the process of smoothing out small irregularities within regions of a generic image. In the case of impact breccia images, there can be noises from the uneven surface of the breccia that need to be removed. Applying a median blur removes these small speckles and soothes the edges of isolated glass clasts.

- FR1: The filter can apply a median blur to an image.
    - FR1.1: The filter can automatically detect the optimal kernel parameter for applying a median blur.
- QR1: The filter only allows the user to adjust the kernel within a predetermined range of values. This range spans the optimal values to apply a median blur to despeckle an impact breccia image.
- QR2: The interface shown to the user should simplify the process of choosing a kernel parameter. Kernels must be odd numbers, so instead of requiring the user to input an odd number, the user can input the kernel's index instead. Example: $1 \rightarrow 1$, $2 \rightarrow 3$, $3 \rightarrow 5$, $4 \rightarrow 7$, etc. This is done to abstract away the knowledge of how median blur is applied, simplifying the process for the user.

Requirement #08 — *Remove outliers filter*

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

Some small clusters of pixels can remain after earlier filters have been applied in the pipeline. These are small regions of "false positives" that we consider as outliers. Removing outliers is the process of removing clumps of pixels that are below a specific size.

- FR1: The filter can adjust the size of what outliers to remove.
- FR2: The filter can remove outliers below a specified size.
    - FR1.1: The filter can automatically detect the optimal outlier cluster size to remove.
- QR1: The filter only allows the user to adjust the outlier size within a predetermined range of values. This range spans the optimal values to remove outliers for an impact breccia image.

Requirement #09 — *Automatically fill in holes filter*

After applying filters to an image, small holes sometimes form in regions of isolated glass clasts. These holes need to be filled in to create a finalized image.

- FR1: The filter can fill in holes in the processed image. Regions of white pixels surrounded by black pixels are replaced by black pixels.

Requirement #10 — *Erosion filter*

- FR1: The filter can erode the edges of clumps of pixels representing glass clasts.
    - FR1.1: The filter can automatically detect the optimal erosion setting for the image.

Requirement #11 — *Dilation filter*

Dilation is the opposite of erosion. This filter adds to the edges of glass clasts, smoothing out irregularities.

- FR1: The filter can dilate the edges of clumps of pixels representing glass clasts.
    - FR1.1: The filter can automatically detect the optimal dilation filter for the image.

Requirement #12 — *Automatic pipeline*

- FR1: The pipeline applies each filter to the inputted image in the order subtract background → invert → Gauss blur → contrast → RGB split → threshold → despeckle → fill in holes. Each filter should be automatically applied, calculating the optimal parameters to adjust the filter by.
- FR2: The user can select or deselect filters from the pipeline.
    - FR2.1: Selected filters will be applied during the automatic pipeline.
    - FR2.2: Deselected filters will not be applied during the automatic pipeline.
- QR1: All filters are selected by default.
- QR2: The user can select or deselect filters by simply clicking a button to represent each filter. Clicking on a selected filter will deselect it, and clicking on a deselected filter will select it.
    - QR2.1: Selected filters should be represented with the filter's icon.
    - QR2.2: Deselected filters should be represented with a greyed out copy of the filter's icon.

Requirement #13 — *Fill bucket tool*

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

- FR1: The tool can register the coordinates of where the user clicked on the image and can store these coordinates.
- FR2: The user can click a region of the image containing a hole and the pixels will be filled in with black pixels.

Requirement #14 — *Magic eraser tool*

- FR1: The tool can register the coordinates of where the user clicked on the image and can store these coordinates.
- FR2: The user can click a region of the image containing a cluster of black pixels which will then be replaced with white pixels.

Requirement #15 — *Manual processing*

The application supports applying filters manually. The user can select a filter and then use a GUI popup window to adjust the parameters of the filter.

- FR1: The user can select a button from the top row of the application and a GUI window will be launched that the user can use to manipulate and adjust the filter controlled by this GUI window.

Requirement #16 — *Image I/O*

- FR1: The user should be able to import an image by dragging it into the main panel of the GUI.
    - FR1.1: Importing an image after an image has already been imported should not interfere with processing the old image, or the new image. The old image should be erased from memory and the new image can continue as if no image had been imported prior.
- FR2: The user should be able to export an image by pressing a button on the top bar, or by pressing a hotkey. See Requirement #17 for details.
    - FR2.1: The user can use a textbox to input the filename to export the image using.
    - FR2.2: The user can select press a button to select from a list of filetypes to export the image with.
    - FR2.2.1: Supported filetypes are, jpg, png, tif, and webp.
- QR1: The default filename is the name of the original image imported into the application.
- QR2: The default filetype is tif.
- QR3: Filetypes can be selected by pressing a button. Only one button can be selected at a time.

Requirement #17 — *GUI*

The application presents each filter to a user through a GUI.

- FR1: The center of the GUI is a large panel that renders the image to the user. The image displayed is either the current saved image, or a preview of the image with a filter temporarily applied to it.
- FR2: Across the top of the GUI is a panel of buttons that represents the actions that the user can perform. Pressing a button will launch the task associated with the button.
    - FR2.1: There is a button to launch the help tool.
    - FR2.2: There is a button to launch the colour inversion filter.
    - FR2.3: There is a button to launch the Gaussian blur filter.

    - FR2.4: There is a button to launch the contrast filter.
    - FR2.5: There is a button to launch the RGB splitter filter.
    - FR2.6: There is a button to launch the threshold filter.
    - FR2.7: There is a button to launch the erosion filter.
    - FR2.8: There is a button to launch the dilation filter.
    - FR2.9: There is a button to launch the outlier removal filter.
    - FR2.10: There is a button to launch the automatic hole filler filter.
    - FR2.11: There is a button to launch the automatic pipeline.
    - FR2.12: There is a button to launch the fill bucket tool.
    - FR2.13: There is a button to launch the magic eraser tool.
    - FR2.14: There is a button to zoom in.
    - FR2.15: There is a button to zoom out.
    - FR2.16: There is a button to reset the zoom and pan.
    - FR2.17: There is a button to undo the previous filter applied to the image.
    - FR2.18: There is a button to redo the prior filter applied to the image.
    - FR2.19: There is a button to export the image with all filters applied to it.
- FR3: All image filters, the automatic pipeline, and export should launch a popup window when their buttons are pressed. This popup window helps the user adjust the parameters for each filter.
    - FR3.1: Only one popup can appear at a time.
- FR4: Each button's functionality can also be launched by pressing a hotkey on the keyboard instead of pressing a button.
- FR5: If no image has been imported into the application yet, then all buttons and hotkeys should be disabled.
- QR1: Instead of labeling each button with text, each button should be represented by an image that showcases the command being controlled by the button.

Requirement #18 — *Image display and rendering*

The image being processed by the application should be rendered to the screen at all times. This requirement covers all temporary effects being applied to the image.

- FR1: The user can zoom in on the image.
- FR2: The user can zoom out on the image.
- FR3: The user can pan the image.
- FR4: The user can reset the zoom and pan of the image.
- FR5: When a filter is applied to the image, a preview of the filter is rendered.
    - FR5.1: If the user closes the popup window of the filter without saving the filter, then the filter is removed and the image is rerendered.
- FR6: The user can toggle an overlay of a previous image over the current one.
    - FR6.1: The user can toggle a primary overlay of the original image over the current image.
    - FR6.2: The user can toggle a secondary overlay of the previous image before the most recent filter was applied.
    - FR6.3: An overlay is disabled when the user tries to apply a filter.
- QR1: The user can control the zoom of the image.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

    - QR1.1: The user can control zooming in on the image.
        - QR1.1.1: The user can scroll their mouse wheel forwards to zoom in.
        - QR1.1.2: The user can pinch inwards on a trackpad to zoom in.
        - QR1.1.3: The user can press a hotkey to zoom in. See Requirement #17 for details.
        - QR1.1.4: The user can press a button to zoom in. See Requirement #17 for details.
    - QR1.2: The user can control zooming out on the image.
        - QR1.2.1: The user can scroll their mouse wheel backwards to zoom out.
        - QR1.2.2: The user can pinch outwards on a trackpad to zoom out.
        - QR1.2.3: The user can press a button to zoom out. See Requirement #17 for details.
        - QR1.2.4: The user can press a hotkey to zoom out. See Requirement #17 for details.
- QR2: The user can control the pan of the image by holding down the left or right mouse button and then dragging the image around.
    - QR2.1: The user can drag anywhere on the main panel to pan the image, they don't only have to explicitly drag the image itself.
- QR3: The zoom and pan can be reset simultaneously.
    - QR3.1: The user can click the middle mouse button to reset zoom and pan.
    - QR3.2: The user can press a button to reset the zoom and pan. See Requirement #17 for details.
    - QR3.3: The user can press a hotkey to reset the zoom and pan. See Requirement #17 for details.
- QR4: The zoom factor is 1.1x.
- QR5: When panning the image, dragging the cursor one pixel corresponds to panning the image by one pixel.
- QR6: Overlays can be toggled by pressing a hotkey.
    - QR6.1: The primary and secondary overlays cannot both be rendered at the same time.
    - QR6.2: Overlays are activated by pressing a button once and then disabled by pressing the button again.
    - QR6.3: Activating the primary overlay while the secondary overlay is being rendered will disable the secondary overlay and activate the primary, and vice versa.

Requirement #19 — *Filter history*

The application should maintain the history of each filter applied to the image. The user should be able to use filter history to undo a previous filter and to redo an undone filter.

- FR1: The user can undo a filter, reverting back to the image before the most recent filter was applied.
    - FR1.1: If there is no filter to be undone, then nothing should happen when trying to access a non-existing previous filter.
- FR2: The user can redo a filter that was previously undone. The same filter using the same parameters should be reapplied.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

    - FR2.1: If there is no filter to be redone, then nothing should happen when trying to access a non-existing future filter.
- FR3: In the event that the user applies a filter to the image, but then undoes the filter, implementing a new filter will erase the original "future" filter from memory. For example, consider the user applies the filter to subtract the image's background and then applies the contrast filter. After this, the user undoes the contrast filter and reverts back to the version with the subtract background filter applied to it. After this, the user applies a Gauss blur filter. When they apply the Gauss blur filter, the contrast filter is erased from memory.
- QR1: Each filter applied to the image should be saved into a designated folder that contains the filter history for the current image. When a filter is applied, save a copy of the image with this filter applied to it to this folder. When undoing or redoing a filter, load this file into the application as the current image.
    - QR1.1: Files for filter history should be saved as such: <original file name>_<filter history id>_<filter applied at this step>.tif. Example: image15_4_contrast.tif
    - QR1.2: Files for filter history should be saved with the .tif extension.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

06: Development Strategies

Development platforms**:**

- VS Code
- IntelliJ

Development frameworks and libraries:

- OpenCV
- Java Swing
- Java AWT
- OpenCSV

Development tools used:

- GitHub

Programming languages used in the project:

- Java

Use of Open source software as part of your solution:

- ImageJ was referenced throughout development as a guideline for what the final product should resemble.

Development methodology:

- IBAT was developed using SCRUM.
- Each component was developed individually and added to the working build of the project once completed.
- Two development teams worked to create IBAT. One team focused on the UI and applying filters manually, and the other team developed the automatic pipeline.

Use of datasets:

- Sample impact breccia images provided by Jamie Graff and Dr. Osinski were used to test and refine the application’s filters.

Agile development platform:

- Jira

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

07: Results — solution created

System architecture and design:

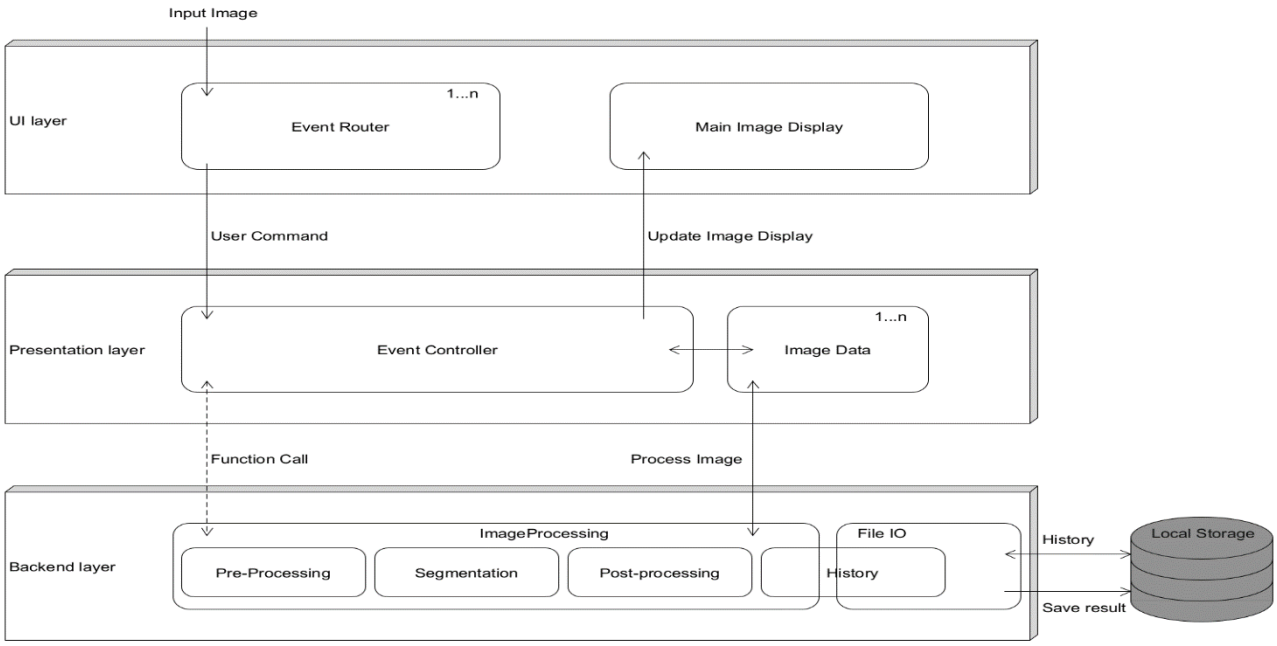


Figure 3: IBAT system architecture diagram

Our system architecture builds upon the existing flowchart of ImageJ, enhancing it with components implemented using OpenCV for automated glass clast analysis. We have implemented a pre-processing module, a segmentation module, a post-processing module, and a history module. These modules encapsulate the processing techniques used in the old ImageJ workflow. We have also created a straightforward GUI that allows the user to easily identify the image processing functionalities of the software and allows the user to perform processing operations in the order they desire.

We utilized the Model-View-Presenter (MVP) architectural pattern to segregate the UI layer (view) from the data processing layer (model) with an additional controlling layer (Presenter). This promotes the inter-operability, modularity and maintainability of the system.

We have also used many design patterns in different layers:

1. Singleton design pattern is used to create a unique morphological operations manager as the processing requires a kernel for convolution and a single instance makes it easier to maintain the state of the kernel.
2. Decorator design pattern is used to wrap the image Mat object with additional parameters to help identify the state of the image.
3. Facade design pattern is used extensively to provide a customized and simplified interface to the OpenCV library. This reduces the coupling of our system with the library and allows us to implement more complex logic such as automated filter and the automatic algorithm.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

4. Flyweight design pattern is used to reduce the amount of memory required to process the image, as sometimes the image size can be very large. We keep a single copy of the image in the Main image object, and all the modifications would be performed on that object.
5. Memento design pattern is used to create operation history in a dedicated History class, we had to store the state of the image on the user's local machine as the data size is large and some of the processing filters are irreversible. This allows the undo/redo functions to be implemented.
6. Observer design pattern is used in the UI to dynamically displays the previews of the processing filter applied on the original image, this allows the user to adjust the parameters without going back and forth using the operation history.
7. State design pattern is used to restrict certain operations being perform at the wrong time. The main image is designed to have three states: RGB, gray, and binary, the processing operations allowed at different state varies. This avoids errors and ensures that the analysis result is meaningful.

- Quality Attributes: Our design prioritizes usability, reliability, and performance. By automating the analysis process, we improve usability and reduce the likelihood of human error. Reliability is ensured through rigorous testing and error handling mechanisms. Performance optimizations are implemented to ensure efficient processing of large data.
- Rationale: Our design decisions were guided by the need for a user-friendly interface, accurate analysis results, and scalable performance. We extracted all the image processing techniques that is helpful for the task from OpenCV and built a user interface around those techniques. We focused on modularity to facilitate future enhancements and maintainability.
- Key Component Interfaces: The view and presenter components were implemented using Java Swing and Java AWT, and the model components and each sub modules were implemented with the OpenCV API.
- Runtime Behavior: See the sequence charts below:

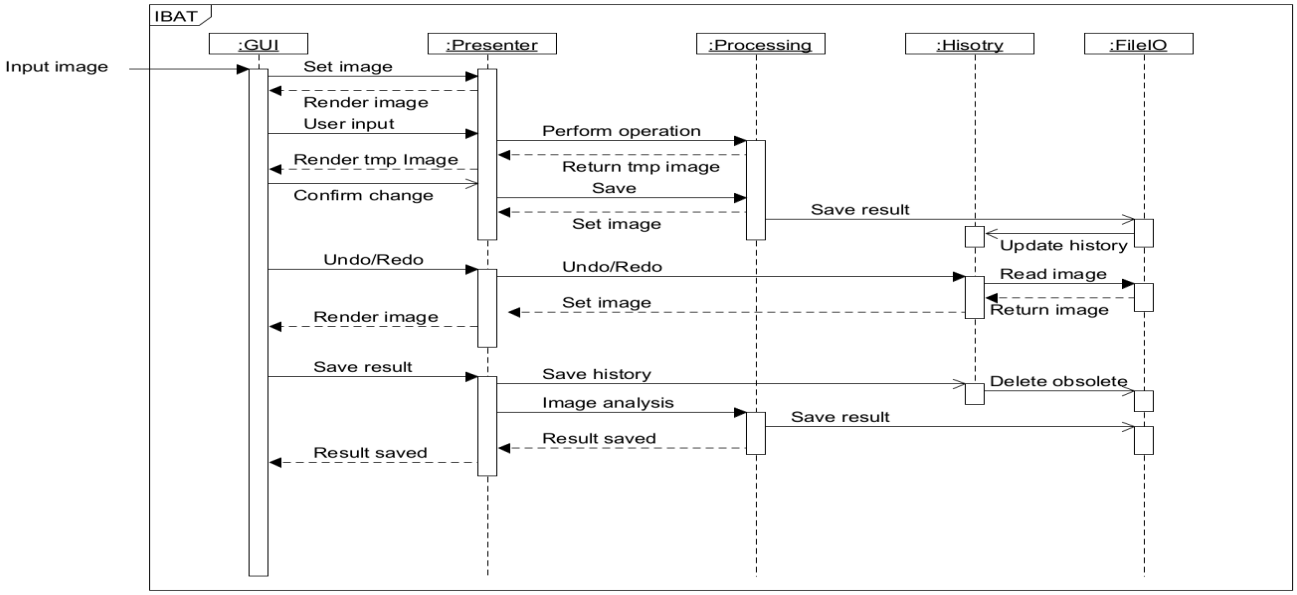


Figure 4: IBAT sequence chart

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

System implementation:

In our experience, these technologies proved to be highly effective and reliable for system development. Visual Studio Code and IntelliJ offered intuitive interfaces and powerful features, enhancing our productivity throughout the development process. OpenCV provided comprehensive support for image processing tasks, allowing us to implement various filters and algorithms seamlessly. Java Swing and Java AWT enabled us to create a user-friendly interface for the IBAT system, ensuring a smooth user experience, OpenCSV simplified the handling of CSV files, streamlining data input and output operations. Additionally, GitHub and Jira played a crucial role in enabling collaboration among team members, allowing us to efficiently manage code changes and track project progress. Overall, we could not have created IBAT without these powerful technologies, they played a significant role in the successful development of the IBAT system.

System testing:

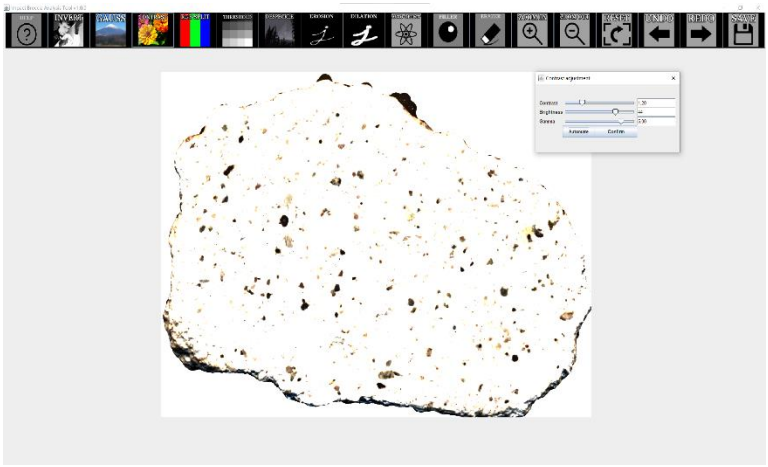
- Unit testing was conducted for each individual processing functions and GUI components to ensure their functionality in isolation.
- Subsystem testing was conducted to validate the interaction between the backend and the UI, each UI should successfully call the corresponding backend function and update the rendered image.
- Integration testing was conducted to verify the integration of new modules with the existing modules and user interface. Each new module added was tested with the existing modules to ensure the image is processed successfully and in correct order.
- Regression testing was conduct after each Git commit, to ensures the new change does not introduce unintended issue.

Operational Evidence of the system:

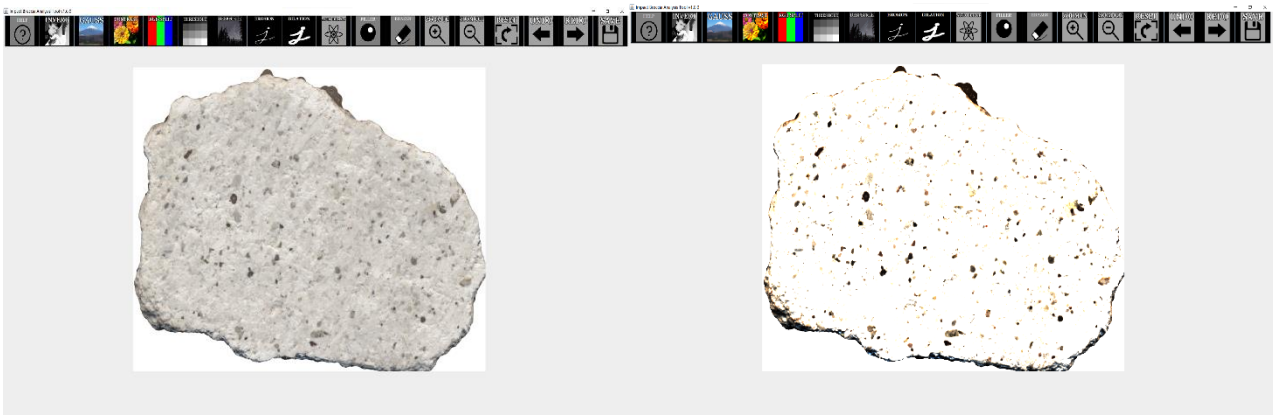
- Scenario 1: The user opens IBAT and wants to input an image.

- Scenario 2: The user wants to apply a specific filter on the image with customized parameters.

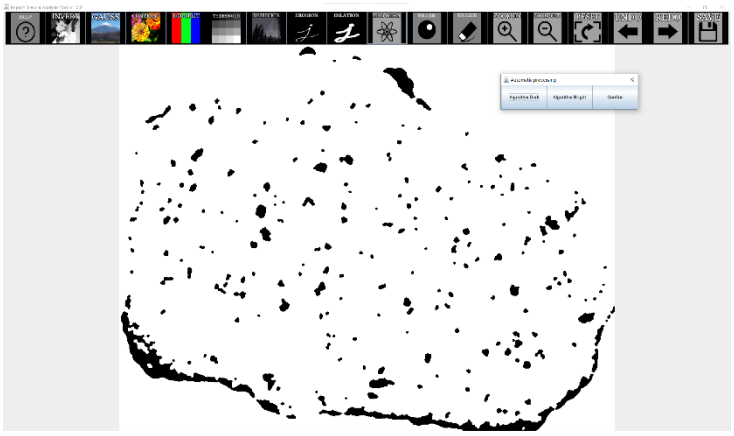
Nicolas Jacobs
Yifei Zhang
Yuhan Zhang



- Scenario 3: The user wants to undo (left)/redo (right) previous operations

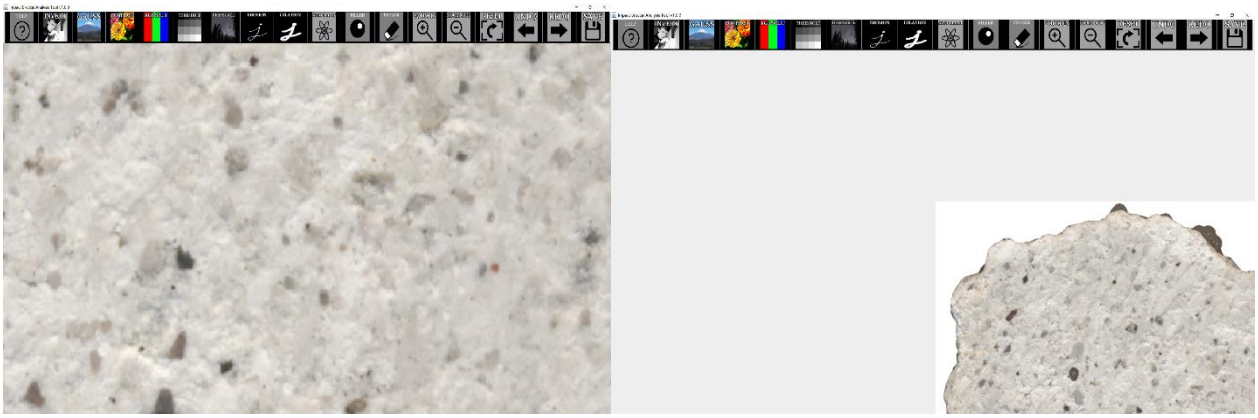


- Scenario 4: The user wants to use the automatic algorithm to identify glass clasts

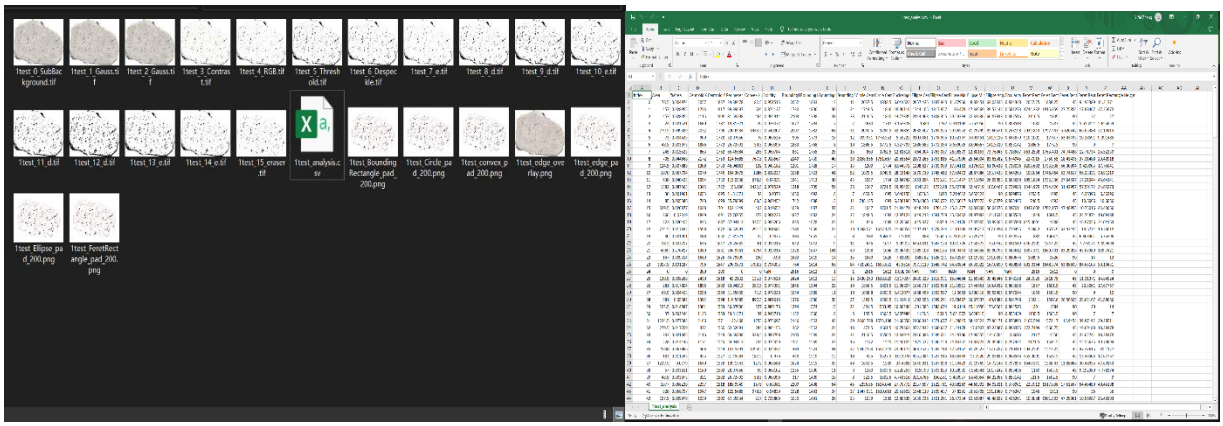


- Scenario 5: The user wants to zoom and pan the image.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang



- Scenario 6: The user wants to export the intermediate images and analysis results



Our system successfully automates glass clast analysis, reducing manual effort and improving accuracy (O1, O3). It provides a user-friendly interface for researchers to conduct comprehensive studies of impact events (O2, O5) and generates meaningful analysis output (O4).

System Validation:

A complete analysis conducted by IBAT using one of the sample breccia images that Dr. Osinski provided can be found in Appendix A.

Changes made based on user feedback include the addition of eraser/hole filler tool, user interface enhancements and performance optimizations.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

08: Discussion

Impact of the System:

- IBAT system would revolutionizes geological analysis by automating glass clast isolation in impact breccias, significantly reducing analysis time and improving accuracy. This shift enables deeper insights into breccia composition and impact processes, enriching planetary science and meteorite studies.
- The IBAT system may also have interdisciplinary applications. IBAT's image processing and analysis capabilities can be applied not only in geology but also in other fields of science where image segmentation and analyzing are required. Stolze, N. et al (2019) presents an innovative method for automating the counting of yeast and mold colony forming units (CFUs) using ImageJ as shown in Figure 5. This appears to be something within IBAT's capability and proves its potential application in the medical science field.

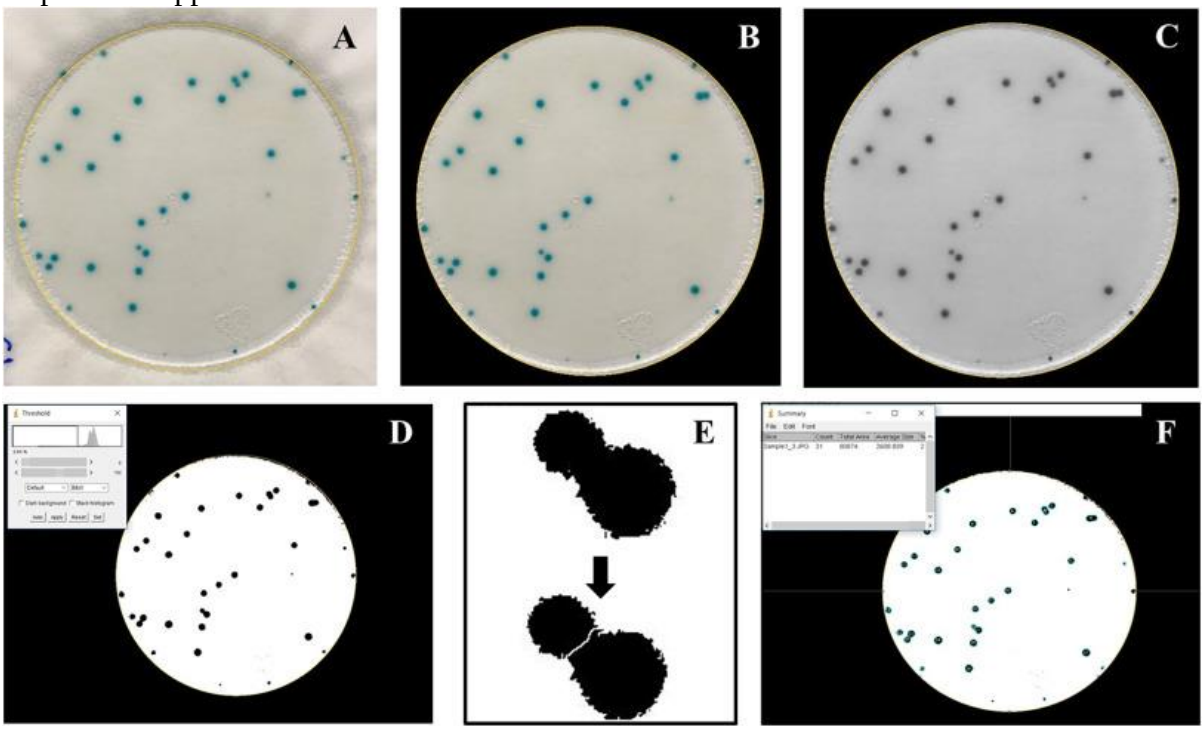


Figure 5: Automating the counting of yeast and mold colony forming units (CFUs) using ImageJ (Stolze, N., et al, 2019)

Limitations of the System:

- IBAT's performance depends on input image quality and the complexity of breccia samples. Current limitations include its adaptability to various sample types and technical resources for development.
- Potential Solutions for Limitations: Incorporating machine learning techniques such as CNN (Pascual, A.D. 2019) for broader sample adaptability and collaborating globally on diverse datasets can enhance IBAT's capabilities. Seeking academic and industry partnerships and investment, is essential for overcoming technical constraints and advancing IBAT.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

09: Conclusions

The motivation of our application has been to improve the process of the initial semi-automatic isolation step. Our objective was to build an application that automates the glass class isolation process with high accuracy so that the time spent manually adjusting the image afterwards is significantly shorter than it was when using ImageJ.

Our project's objective has been to build an image processing application that can automatically isolate glass clasts inside impact breccia images. We built a GUI using the Java Swing library and utilized the OpenCV library's image processing filters to create algorithms that can accurately isolate glass clasts. If the automatic pipeline does not accurately process the image, the application was built to allow users to manually apply each filter.

Our project created an application that can accurately isolate glass clasts within impact breccia images. This tool will give researchers access to data and isolated image maps faster than they could before. We were able to implement an automatic algorithm which requires little to no user input and generates comprehensive analysis report for researching purposes (See Appendix A for the entire process of the automatic algorithm).

The previous process of isolating glass clasts used ImageJ. After speaking with researchers, our understanding is that it takes several hours to complete the process of isolating an image with ImageJ. Our application aims to be an advancement over ImageJ because it can automatically isolate glass clasts, unlike ImageJ which requires the user to manually adjust each filter and then perform a lengthy stage of manual pixel-by-pixel adjustments afterwards.

The largest limitation placed on the development of our application was time. Having more time to fine tune the application's pipeline and to improve the UX would have made the application better.

Our conclusion as software developers is that communicating with stakeholders is an incredibly important aspect of creating software. We are building an application specifically for the stakeholders to use, so proper communication is necessary to ensure that we create the exact application that they want to use.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

10: Future Work and Lessons Learnt

Future work:

- The “industry” of analyzing impact breccias will certainly not end with the completion of this application. To improve the process in the future, new applications might be created using our application as inspiration.
- Some “wish list” features were considered during development, but ultimately not included due to time constraints. Other tools and applications could still be created to automate/implement the features that we were unable to implement.

Lessons learnt:

- Software development requires extensive planning for the project to succeed.
- While a component is being considered, do proper research on the feasibility of implementing it into the system.
- It is always crucial to start development of your system sooner rather than later because deadlines appear faster than you expect, and then you’re scrambling to get something done.
- Maintain good communication within your development team so that everyone is aware of the current state of the application, what features are current in progress, and what features are being planned.
- Software development teams absolutely require a project manager to help oversee the system’s creation because otherwise the development process becomes rather messy.
- Properly assign work to each member of the development team so that multiple versions of the same component don’t get created.

11: Acknowledgements

We acknowledge the team behind the creation of the OpenCV library, as many of the components in our application rely on the tools provided by OpenCV.

We would like to thank PhD student Jamie Graff for his support, feedback, and assistance throughout the development of the application. Jamie was immensely helpful to us by providing us with samples of impact breccia images to test our application's filters against, as well as offering feedback on the mechanics and implementation of each component of the application.

We would like to thank Dr. Osinski for the opportunity he granted us by becoming the supervisor for our capstone project.

We would like to acknowledge the work done by Antonio Bernal Martínez, whose paper on using computer vision for rock classification (Martínez, A.B. 2021) helped us understand and refine several of our processing filters. We also give acknowledgement to their object detection approach (Martínez, A.B. 2021, "Objection Detection" section), which inspired us on the implementation of background subtraction algorithm in our application.

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

12: References

Chanou, A., Osinski, G. R., & Grieve, R. A. F. (2014). A methodology for the semi-automatic digital image analysis of fragmental impactites. *Meteoritics & Planetary Science, 1–15*. https://doi.org/10.1111/maps.12267

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

Pascual, A. D. (2019). Autonomous and Real-Time Rock Image Classification using Convolutional Neural Networks. *Electronic Thesis and Dissertation Repository*, (6059). Retrieved from https://ir.lib.uwo.ca/etd/6059

Stolze, N., Bader, C., Henning, C., Mastin, J., Holmes, A. E., & Sutlief, A. L. (2019). Automated image analysis with ImageJ of yeast colony forming units from cannabis flowers. *Journal of Microbiological Methods, 164*, 105681. https://doi.org/10.1016/j.mimet.2019.105681

Changing the contrast and brightness of an image!. OpenCV. (Date of last access: April 1, 2024) https://docs.opencv.org/4.x/d3/dc1/tutorial_basic_linear_transform.html

Image Thresholding. OpenCV. (Date of last access: April 1, 2024) https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

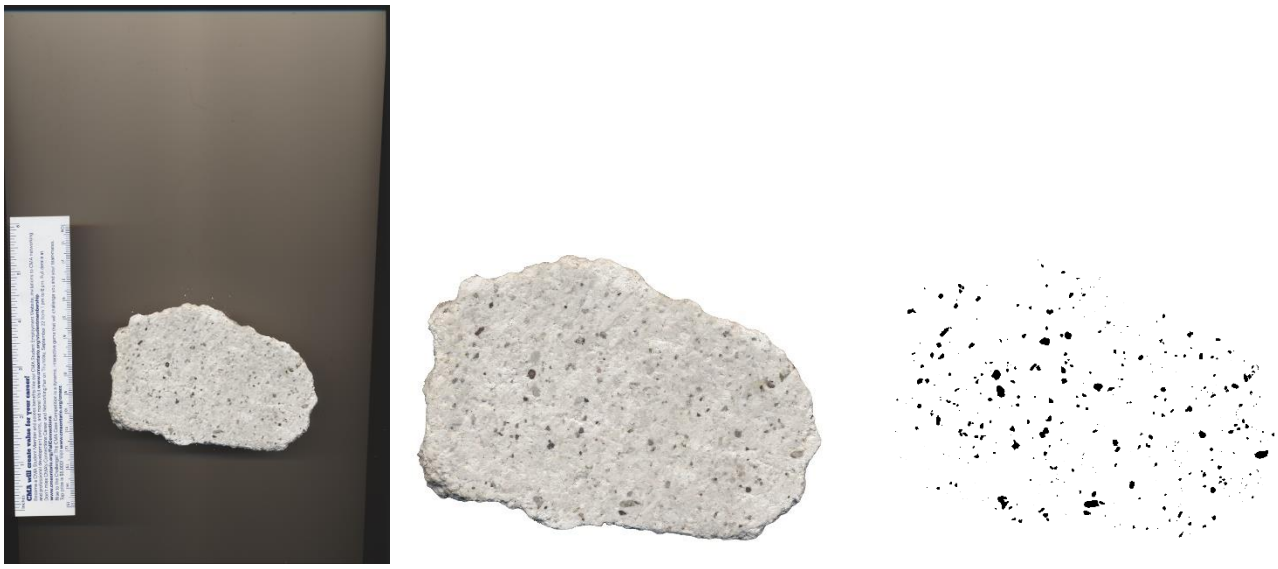
Morphological transformations. OpenCV. (Date of last access: April 1, 2024) https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

Wikipedia contributors. (2024, February 3). Grayscale. In Wikipedia, The Free Encyclopedia. (Date of last access: April 1, 2024) https://en.wikipedia.org/w/index.php?title=Grayscale&oldid=1202807394

Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

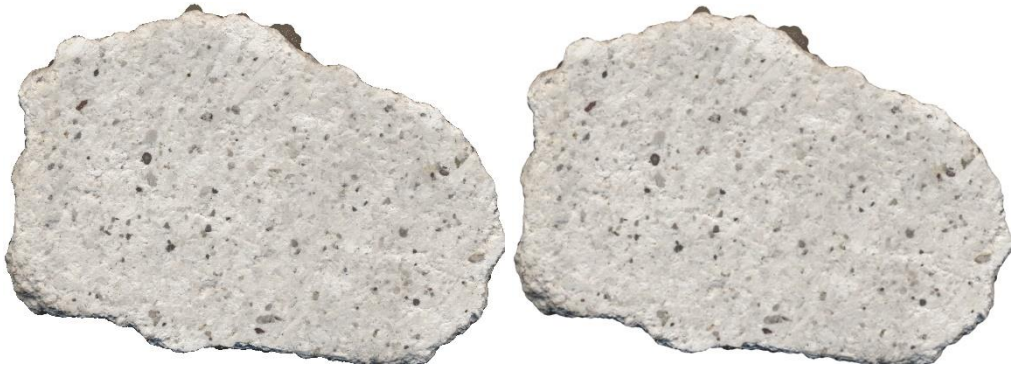
13: APPENDIX A (Section 7. Results - System Validation)

Sample image and analysis result produced using ImageJ:

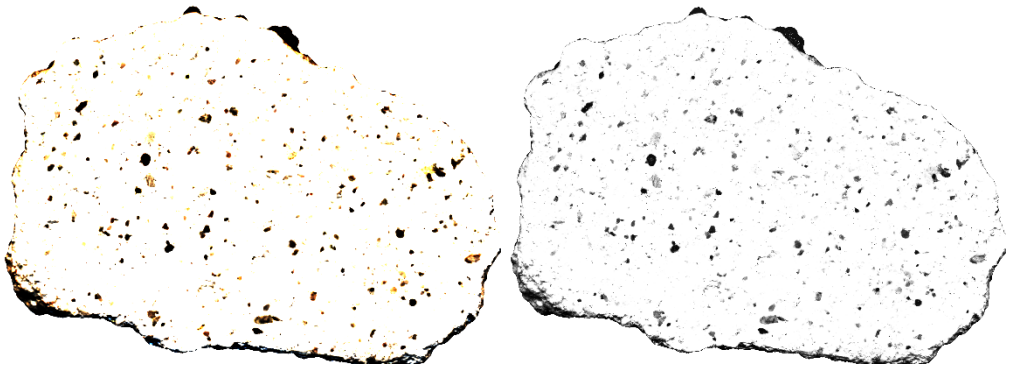


Result produced by IBAT's automatic algorithm with few corrections:

1. Background subtraction, Gaussian blur



2. Contrast adjustment, RGB split



Nicolas Jacobs
Yifei Zhang
Yuhan Zhang

3. Thresholding, Despeckling

4. Erasing false positives, Morphological transformations

5. Result overlay, Particle analysis report