NORTHERN ARIZONA UNIVERSITY
College of Engineering, Forestry & Natural Sciences
Department of Computer Science

Final Report

Version 1.1

Jet Propulsion Laboratory Image Analysis

Client: Iona Brockie

Team Hindsight

Charles Beck, Alexanderia Nelson, Adam Paquette, Hunter Rainen

May 10, 2018

Mentor: Austin Sanders

Table of Contents

1. Introduction

We are Team Hindsight, and we worked on the project ‘Image Analysis of Abraded Rocks to Determine Dust-Free Area’. Our project sponsor is the Jet Propulsion Laboratory (JPL) at the California Institute of Technology with our main point of contact/client Iona Brockie, a Mechatronics Engineer at JPL.

JPL is a federally funded NASA research and development center whose primary role is to construct and operate planetary robotic spacecraft. Mars is one planetary body that JPL has been actively carrying out scientific missions on over that last few decades. JPL’s newest rover, Mars 2020 (M2020) will aim to further deepen our understanding of whether or not the red planet was once habitable. Specifically, the primary goal of M2020 will be to look for evidence of past life on Mars by analyzing and collecting samples of the Martian surface by drilling into rocks, which will then be picked up by a future mission.

In its quest to find evidence of past life, the M2020 rover will use a suite of tools including an onboard drill with a set of drill bits to take measurements of the soil/rock and potentially collect samples from the Martian surface to return to Earth later. To identify what samples to take, the rover is equipped with a Planetary Instrument for X-ray Lithochemistry (PIXL) camera. The PIXL camera looks at a particular region and analyzes it for chemical compounds and elemental makeup. However, before instruments like PIXL can analyze samples, the rover needs to overcome a problem inherent to drilling. When the rover drills into a rock, it creates a lot of dust, obscuring the hole. JPL then blows dust out of the hole using compressed gas.

Team Hindsight’s goal was to create reliable software to automatically analyze the effectiveness of this gas Dust Removal Tool (gDRT). Our tool looks at before and after images of dust being blown out of the hole and returns the area covered in dust after using the gDRT.

2. Process Overview

Overall life cycle:

We went with an agile development process, focusing on producing a finished, working product. We started on Monday of each week with a team meeting including our mentor. At this meeting we focused on setting goals for each team member as well as reviewing what we completed the previous week.

Every Friday we had another team meeting, excluding our mentor. At this meeting we discussed our progress and got together to complete any tasks that were more time consuming. This meeting also included our client every other week, who we discussed our progress with, asked any questions, and requested resources that we needed from her.

Tools:

- Git for version control
- Trello for managing tasks
- Slack and Google calendar for team communication and scheduling
- Google drive for working on documentation in parallel

Team roles:

Hunter: Team lead, developing code for dust detection

Because Hunter had prior experience at JPL and understood the project scope from the get go, the team agreed to make him the team lead. His primary role was to both work on the code (dust detection) as well as divide up the workload amongst the team members.

Adam: Software Architect and coder.

Adam had experience working in industry and is familiar with standard software practices so we elected to make him the software architect. This role was largely responsible for putting together the different parts of code and integrating them into one cohesive product.

Charles: In charge of GUI, as well as documentation

With the prior roles set, Charles having experience with Python, elected to be incharge of the graphical user interface as well as keep up with the documentation.

Alex: Website development, Documentation

Alex's primary role was to keep our capstone website updated, the primary use of the website is to store links to download documentation as well as give readers a landing page to read about our project. Alex also assisted with documentation along the way, and aided anyone who needed help with the coding parts (e.g. dust detection or installation of software).

3. Requirements

As mentioned in the previous section, our team lead has prior experience working at JPL and had a good understanding of the project. To hammer out the specifics, the team exchanged emails and held bi-weekly meetings with our Sponsor to set requirements that would lead to a useful tool.

From these meetings we developed a set of specific requirements that would that defined a minimum viable product (M.V.P.) for our client. In short, the M.V.P. need to be able to take in a single pair, or multiple pairs of images (abrasion images), automatically analyze those image pair(s) and show the user what percentage of the abrasions are covered in dust. Below is a more detailed list of these requirements. Specifically, more detailed requirements along with requirements not mentioned above, like accuracy requirements and performance requirements

Functional requirements:

1. Take in batches of images of drill/ dust removal testing
2. Detect dust free regions in images
3. Visually display results of application to users
4. Allow users to tweak output of application
    - 4.1. In other words be able to tweak how the program analyzes a given image

Performance requirements:

1. The time our program should take to analyze before/ after images
    - 1.1. For a single pair of images < 5 min
    - 1.2. For batch of 10 images < 30 min
2. How accurate should the application be
    - 2.1. Within 10% of JPL's numbers for a dataset of hand analyzed images given to Team Hindsight by JPL

4. Architecture and Implementation

The generating of our solution involves many layers, from the overview to the tools we are using. What brings together our vision and libraries we are using is the architecture we have decided to work with. This structure is the Model View Controller (MVC) architecture that consists of three main parts: Model, View, and Controller. Each of these parts integrate a specific library or tool that permits our software to carry out a technical analysis all while allowing the user to interact with the software with little effort.

4.1 Model View Control (MVC) architecture

For this project, the Model, View, Controller (MVC) architecture is one viable route to represent the process/structure of our program. We are developing a GUI (the View) as the front end for the user to interact with, a Model that will represent an image/analysis of an image, and a Controller module that will communicate between the user interface and the module responsible for analyzing an image. MVC is a widely accepted approach for integrating a user interface with a backend. It allows for the separation between the user interface and the backend, keeping each pieces logic to itself.

MVC allows our team to keep program functionality modular and efficient. As such, having a controller that interfaces with the view and the model will greatly simplify the design while still giving our client flexibility with program functionality.

Below is a simple diagram for the architecture described.

Figure 1.1: Overview of architecture

4.1.1 Model

The model is how the image data will be stored in a data structure. This includes the original image taken as input, the transformations that image undergoes, and the final output image that is used by our software to analyze the effectiveness of JPL’s tool. The model is responsible for storing information about a single image.

There are multiple images that our software will analyze and each image has its respective model. At any point, users can look at the model and look at how each image is being processed and get a clear understanding of how our program got from the original image to the final analysis.

4.1.2 View

For our user to be able to interact with the program's analysis functions and pass data into these functions, we are integrating a Graphical User Interface (GUI). Since we are using the MVC architecture, the View is an important aspect as it allows the user to tweak the input and output with the program without having to modify the code as well. A GUI gives the user a simple interface to interact with versus a command line interface. What we need our GUI to do is carry out a number of communications between it and the controller to get a proper execution of the program.

4.1.3 Controller

The controller acts as an interface between the view and the model. As per the architecture pattern, the control takes input from the user through the view, then asks the model for the information that was requested in the GUI. The model then returns what was asked for to the control which then pipes it back to the GUI. This system allows for an encapsulation of logic, where the GUI only handles user interaction, and the model handles all of the logic, making the controller the interface between the two.

4.2 Module Implementation and Interface Descriptions

Now that we have given a basic introduction of our systems architecture, we will go over each module in more detail, and how the modules interact with one another. Specifically, this will be a low-level description of how each module handles its own operations on itself or other objects. It will also discuss in greater detail how these modules work together to achieve a MVC style architecture.

4.2.1 Model Description

The Model can be thought of as an Image object or struct, where the image object contains multiple arrays. Each array contains data on each pixel in the image. Every image object is responsible for storing the original image array and any in-between filters used to better identify dust. A filtered image is the result of the original image put through a single function, or set of functions that change the value of every pixel in the image. This includes the final analysis on dust density (how much of the abrasion is covered in dust). As such, the image object (Model) will also include the results of analysis of these images.

The Model is also responsible for doing the analysis, it takes in the initial image (array of pixel values) and any parameters that are needed for the functions to filter the image for analysis. Once the Model finishes analyzing the image, it notifies the Controller, which in turn updates the View.

Figure 1.2: Model Visual

4.2.2 View Description

The View section of our MVC system is a Graphical User Interface (GUI) that allows the user to input a file path that is then used to carry out the multiple analysis functions we have set for this system. The View also allows the user to change parameters such as light thresholds so that the program outputs what they deem is important or more clear to understand. We want to have this interaction between the user and program so that they have more control over the outcomes without having to know how to code or modify the source code.

The interaction of the View with the Controller in our architecture relies on the information passed from the View in input/output (I/O) to the controller so that data can be formed into the model that we compare our analysis to. This information can then be communicated to the View from our dataframe. The View will display the original image after a blast of air (after image), and then the analysed after image so that the user can see the output of our program instead of having to open up an image in their file browser. We have given the user the option to either save the image of the analysed portion so that only necessary results are stored. The user can also communicate to the program whether or not they want to change the code in any way and it will then open the source code and allow them to view how our functions work as well as change them if need be.

The GUI has been written using Python's tkinter library and Python 3.6 to allow for quick and seamless integration of Python as well as other libraries such as OpenCV and Pandas Dataframes. We intended this interface to work with Python, so using tkinter as a built in Python GUI library was ideal, and worked nicely with our other Python modules/code.

The interface begins as a simple window with a selection to choose the file handle (Browse…) and then select that file handle for use with the Select button. The user can also just type in a file path and then select it since the text bar can be typed into, the user will type in the file handle and then press select to move onto the next screen.

Figure 1.3 GUI File Select screen

From here the user will be presented with the main analysis window. This window has a frame for displaying the images, label for the file in use, and a section for the functional buttons. There are four different buttons currently on the analysis window such as “Run”, “Save”, and “Change File”. The run function will pass the images into our pandas dataframe so that the rest of the python code will be able to access the information.

Figure 1.4 Analysis window

The program will then run the appropriate analysis on the image set based off of the rock type specified in the drop down menu. When analysis is complete it will display the associated images “abraded_030_after1”, and then “analysis_030_output”. This allows the user to view the original after images alongside our programs output as seen in Figure 1.5.

Figure 1.5: Analysis output

Below the analyzed image there are three percentage values that are labeled 'G' for green, 'Y' for yellow, and 'R' for red. This is done to aid in readability and understanding of the program output. If the user is satisfied with the outcome of the test the user can select the "Save" button to save all the analyzed image data that was created during the most recent run. The analyzed image data being the Greens, Yellows, and Reds generated by the analysis. This saves the analyzed image data to the directory where the image sets where selected from.

To get a more fine tuned user experience we will allow the user to see different options for image analyzing processes that are available for the different rock types. To exit the software there is an “exit” button that closes all windows and ends all processes associated with the software.

4.2.3 Controller Description

The controller module uses a pandas dataframe. Pandas is a Python module that allows for the creation and storage of in-memory tables, also known as pandas dataframes. This structure allows multiple model objects to be stored in it simultaneously, allowing each model to exist individually from the other models. That is, all images exist as their own image without needing to know about each other, while the controller handles the necessary associations between them. This controller table will have five fields:

- index
- before_image

- after_image
- group
- output_image.

Figure 1.6: Pandas Dataframe

These fields represent the index in the dataframe, a model object, another model object, a grouping tag that rows can be sorted on, and a model object, respectively (Figure 1.6).

Pandas also allows for functions to be applied directly to each row in the dataframe. While each model object exists as its own entity, we can apply functions to all model objects in the table, effectively making our model object one entity.

Figure 1.7: UML Diagram for Controller Module

4.2.4 Model Interface

The two primary methods that the control class implements are add_image_pair, and apply_func. The add_image_pair function is the main way to inject Models (images) into the table. The first two strings are for the image name, the second string is for the basepath they both share. The final two arguments are for passing any extra values through to the Model constructor function. This function reads in two images, creates Model objects for them, then stores them in a row of the dataframe.

The apply_func method is the main way that the Controller interfaces with the Model. This function takes a function name in as its first parameter, then any other arguments or keyword arguments that function would need. It then takes said function and extra arguments and applies them to each row in the dataframe. All control functions take in a row of the dataframe as its first argument, then molds the row data into the necessary components to send to the Model. The Model then operates on itself within the dataframe. This is what all of the control functions do when operating on the model.

For example, the normalize_func extracts two fields from the row it receives, the before_image and the after_image. It then calls the Models own normalize function and lets the Model handle the logic of the operation. The Control class extracts objects from fields in the pandas dataframe and tells the Model how it should change based on the fields and objects extracted.

This pattern is followed for each of the functions listed under Control Functions in Figure 1.7. Each of them take in a row of the dataframe (aka pandas series) gets the Model(s) it needs and tells the Model(s) what to apply to itself.

4.2.5 View Interface

The Controller will receive a configuration object from the view. This configuration object will contain the images, the parameters to supply to the functions, the rock type being analyzed, and the functions to apply to those Models.

Specifically, the configuration will have a basepath to a folder containing the images to analyze, and a list of images that were in said folder. The Controller will create Models for each image in the image list using an image name from the list and the basepath. It will then create pairs of Models based on the prefix of each image in the image list, where the prefix for both images in the pair is “abrasion<image_number>_abraded” for the first image in the pair and “abrasion<image_number>_after<after_number>” for the second image in the pair. The Model pairs would then be stored as before and after images in the dataframe.

The parameters will be a list of lists where every inner list is a set of parameters to its associated function. These parameters can very wildly as each set will be associated with a different function. As such, the functions in a configuration object will be an array of operations. Where every operation in the list is parallel to its associated parameters in the parameter list.

Finally, the functions would be run on the dataframe with all associated parameters, which will result in an output image in the dataframe output column. The controller would then pass the now populated output column back to the view.

4.3 System Implementation

Our architecture arose from the method in which we began developing the software. We found that we were developing a Model structure for analyzing the abrasion images. Since we needed a GUI, or a View, it was logical to adhere to the M.V.C. pattern as we would only need the controller to communicate between the two.

We were able to implement the modules in the way we planned with a slight exception to how the View and the Controller communicate. As it stands, the Controller is performing some operations that should be refactored to exist in the View. This includes how the images are being displayed back to the user. Currently this is happening inside of the Controller rather than passing the images back to the View which then handles displaying the images to the user. The save functionality should also be moved into the View for similar reasons, as the Controller should not be handling how images are saved.

These are the only inconsistency in the architecture as all other elements in the View, and Controller, as well as the entirety of the Model are implemented as intended. Where each module handles the appropriate logic and communicates between the two as necessary.

5. Testing

To validate that our software was working as intended, we planned out and executed on a testing plan for our system. This testing included various programmatic tests to check that all code was working correctly. We also asked our client to use our system and provide feedback on the usability of the system to improve the GUI.

For testing the code itself we decided to use a third party Python testing framework called pytest. This framework allows for script style testing of our code, handling both unit testing, and integration testing. It also gives a percentage of code covered shows how much of the system was actually covered by current set of tests in the system.

We decided to do unit testing, integration testing, and usability testing on our system. Each of these tests a different part of the system to ensure that things are working correctly. Below is a more detailed explanation of what sections of our system each set of tests is checking.

5.1 Unit Testing

Unit testing focused on testing small independent pieces of our tool to ensure reliability and accurate functionality. Before we tested that all of our software modules were working together cohesively, we needed to be sure that they worked well individually. To make sure our software worked correctly, we began by testing the individual parts such as the control functions, utility functions, image analysis, configuration, and graphical user interface (GUI) functions. We designed our software using the Model View Control architecture (MVC). This means that the Controller is set to handle the communication between the Model (image data) and the View (GUI), while the Model handles changes to its own data.

Unit testing was done on the dust detection algorithms for all currently working rock types, which only consists of rock type E. This was done iteratively trying different configurations of matlab's color segmentation until a correct configuration was reached. While the dust detection algorithms were tested, the majority of the system remains untested due to time constraints. This includes analysis functions, utility functions, control functions, and GUI functions.

5.2 Integration Testing

Integration testing focused on testing the interfaces between major modules and components to make sure interactions and data exchanges occurred correctly. Our system consists of three main modules, an Image module as the Model, a Control module as the Controller, and a tkinter Graphical User Interface (GUI) as the View. These are the main components of the MVC architecture. Currently the system has four lines of communication, where data is sent between modules:

- View to Controller
- Controller to Model
- Model to Controller
- Controller to View

As such we planned on testing each transmission of data from one module to another to ensure that the modules are correctly integrated with one another and that data sent between modules resulted in the appropriate changes.

Currently no integration testing was done between modules in the system. While integration testing was planned for communication between all modules, we would only need to do View to Controller and Controller to View communication integration testing as it was the only custom logic for communication between modules. However, we were unable to write any integration tests due to time constraints.

5.3 Usability Testing

Usability testing focused on ensuring our end user (JPL) could understand and use our tool effectively to produce quality image analyses. The purpose for our usability testing was to have some members from our client's business, JPL, test how they interact with our application. We wanted to know how our users would interact with the GUI to apply the various analysis methods that have been implemented so far for Rock Type E to images they give the application and receive the analyzed images. They would use the latest complete version of our prototype to conduct this portion of our testing plan.

In our usability testing we were able send our software to our client, JPL and get feedback on what we could improve to make the GUI easier to use. JPL wanted us to change two things. First the analysis window opened slightly smaller than necessary, so some elements were cut off from the users view. Second, they suggested being able to close all windows associated with our software at once. Otherwise JPL would need to enter control + c into the terminal to force quit the program.

To address the first issue, we expanded the window on creation it would be the appropriate size to facilitate all GUI elements. To address the second issue we added an exit button in the analysis window that would destroy the window and fully close the program for the user. In addressing these two issues we were able to finish our usability testing and provide our client with an easy to use GUI for our software.

6. Project Timeline

Figure 1.8 Project Timeline

In the first semester, we focused on gathering our requirements from our client and proving our chosen technologies before the second semester, where implementation would be the focus. The following are the key phases from our first semester.

6.1 Technical Feasibility

We looked at possible technologies we could use to solve our client's problem, like computer vision algorithms. We were given the okay from our client that we were not restricted to using only MATLAB, a language the people in our client's business is the most familiar with. Each member of the team did the following research for what possible tools we could use to prototype later in the semester:

- Charles explored prototypes from pictures of dust/dirt
- Alex compared/contrasted Python and MATLAB to see which one would be used
- Hunter looked at the pros and cons of how to model dust
- Adam explored if it was possible to modify parameters in OpenCV

The prototypes from this phase lead into creating a prototype to be shown to our mentor at the end of the semester before starting the implementation phase.

6.2 Requirements Acquisition

We mainly acquired them through client meetings via Skype calls and emails our team lead sent to our client. Once both parties were satisfied with what our solution is required to do, our client signed off on the document we produced. There have been a few changes to our requirements that we discussed with the client before going through with it so everyone would be on the same page.

6.3 Demo Chosen Technologies

At the end of the semester, we demoed to our mentor a prototype to prove that the technologies we chose earlier in the semester would help us solve our client's problem and produce a working product at the end of the capstone year. They also proved that our solution will address the challenges our client is currently facing. The demos that were demonstrated were:

- Run analysis on two images in under five minutes;
- Use image subtraction to show we can make the edge of the abrasion more visible than in the original picture;
- Use color segmentation on the after images to show that some dust is marked up and things that are not dust in the image are filtered out: and
- Given a binary image show that we can detect regions and identify how much of a given region is covered in dust.

The spring semester was the result of the culmination of our work from the previous semester with the development of our minimal viable product: a software that can detect and analyze dust for one rock type. On our client's suggestion, the first rock type chosen to be analyzed for dust was Rock Type E because the dust is distinguishable from the rock.

6.4 Dust Density Improvement

Adam worked on improving the dust density algorithm we had so that it would give a better visualization of the dust density by marking pixels as red, yellow, and green. It also had to compute and display the percentages of green, yellow, and red pixels in the mask to the user to satisfy one of our requirements.

6.5 Develop GUI

Charles developed a GUI that the engineers at JPL will interact with when using our software. Once the basic functionality of displaying images and the option of selecting images was added, the functions of analyzing the images was integrated and tested to make sure everything was communicating correctly. The GUI was also updated constantly throughout the semester to include more features such as, save analyzed images, and tested when the analysis functions have been updated.

6.6 Find Center of Abrasion

Because of the nature of how the dust could be obscuring the edge, most edge detection algorithms were unable to correctly identify and draw the abrasion's perimeter. Alex developed a function in MATLAB that uses hard-coded values that are uniform for most of the images given to draw a circle on the image of where the abrasion is located. In the GUI, users can adjust the location and size of the circle as they see fit.

6.7 Rock Type B Analysis

After deciding which rock type out of the other rock types we had left would be easiest to detect dust, Rock Type B was chosen. We could not reuse what we have for Rock Type E for this rock type. Hunter developed a different color segmentation algorithm for the light and dark areas since the dust in Rock Type B is a similar color to the rock. A histogram analysis was implemented to reduce the noise and improve the accuracy of the Rock Type B analysis.

6.8 Circular Region Analysis and Circular Dust Density

This was added to our development schedule because the yellow dust density values were problematic with the original square region analysis. Adam worked on applying a circular mask to analyze the image to abrasion area and allows the system to accurately generate dust coverage percentages for regularly shaped gas blows. The square region analysis was found to be more accurate for irregularly shaped dust areas.

6.9 Increase Accuracy of Rock Type E Dust Analysis

It was found that for some images for Rock Type E would be incorrectly reported as having some areas in the abrasion as clear of dust when the original image would clearly have dust in the area. The cause of this issue was the region being incorrectly reported was in a shadow, and this messes with the accuracy of our algorithms. We requested from JPL to improve the lighting when they take these images to remove the shadows and have implemented a function to combine the light and dark masks.

6.10 Deploy MVP for Usability Testing

To get user feedback on our GUI and the performance of our software, we sent our product to JPL so they could get the software installed on their systems and a questionnaire to fill out, so we can get feedback and make improvements.

We are currently wrapping up the final documents and sent the software to JPL for testing. We also did our acceptance test demo with our client and received positive comments on what was shown.

7. Future Work

Though there were features not quite in the scope of our time line, there are many things that could be improved or added. As such, we created our product with modularity and extensibility in mind. For example:

- Analysis could be parallelized, currently, our program analyzes each image pair sequentially. Could analyze multiple pairs at the same time. Could also parallelize the analysis algorithms improving run time of analysis on a single image pair.

- More rock types could be added. Currently, we handle only rock type E. There are more challenging rock types we did not get to. Thankfully, with the way we built our tool, the team at JPL can create a new dust analysis algorithm (that returns a binary image where white pixels are dust), that they can add into our tool chain with relative ease.

8. Conclusion

We are Team Hindsight, and we are working on the project 'Image Analysis of Abraded Rocks to Determine Dust-Free Area'. Our project sponsor is the Jet Propulsion Laboratory (JPL) at the California Institute of Technology, JPL is a federally funded NASA research and development center whose primary role is to construct and operate planetary robotic spacecraft. Mars is one planetary body that JPL has been actively carrying out scientific missions on over that last few decades. JPL's newest rover, Mars 2020 (M2020) will aim to further deepen our understanding of whether or not the red planet was once habitable.In its quest to find evidence of past life, the M2020 rover will use a suite of tools including an onboard drill with a set of drill bits to take measurements However, before instruments on M2020 can analyze samples, the rover needs to overcome a problem inherent to drilling. When the rover drills into a rock, it creates a lot of dust, obscuring the hole. JPL then blows dust out of the hole using compressed gas.

To test the calibration of the gas dust removal tool JPL needs to drill abrasions in a vacuum chamber at Mars atmosphere, which takes an hour to a whole evening to change. Then they generate the image pairs, and remove the rock from the vacuum after pumping backup to Earth atmosphere. Finally they can analyze the abrasion by hand and get their percentages which takes 45 minutes to an hour. Their process of calibration is slow, time consuming, and has a degree of human error.

Team hindsight has created reliable software to automatically analyze the effectiveness of this gas Dust Removal Tool (gDRT). Our tool looks at before and after images of dust being blown out of the hole and returns the area covered in dust after using the gDRT. Our desktop application (Windows and Linux) has a graphical user interface (GUI) that will let users select which rock type to analyze and how to analyze it.

Then, the GUI will send the information selected by the user to another program that will handle the image processing using industry standard computer vision algorithms. We have reduced the human error and inaccuracy associated with moving the rock from the vacuum chamber. We have decreased the time it takes to analyze images from 45 minutes to a a few minutes for a batch of image sets. Our analysis is consistent as our algorithms are being handled by a computer and basing all results off the same metrics. This is consistent and in many cases just as or more accurate than being analyzed with the human eye. By automatically analyzing these images with a program, our team will save JPL many weeks of testing and improve the accuracy and consistency of their results.

In giving JPL a tool to speed up testing their gDRT, we have allowed them to do more testing in a shorter time frame. Our client has stated that there is a chance our software will be used for the life cycle of the M2020 rover on Mars. We hope JPL continues to use our product and that they will launch the M2020 rover confident in the capability of their tools they are sending to Mars.

9. Glossary

Interface: How the user interacts with software.
GUI: Graphical User Interface
Color Segmentation: Separating pixels based of color ranges.
Mask: An image used to layer over another image.
Binarized: Pixels in image changed to either black or white.
CLI: Command Line Interface or console.
gDRT: Gas dust removal tool.
Abrasion: A shallow hole drilled into some surface.

Appendix A: Development Environment and Toolchain

- **Hardware:** The software was developed on windows and linux/MAC OS X machines. For the software to work we recommend any computer capable of running MATLAB and openCV, as these are the main tools we use for image processing. For the software requirements we have the following

- Matlab version 2017b or higher
- Python 3.6
- Windows, linux, (Mac is buggy with user interface colors, but still technically works if you press the non-colored buttons)

- **Toolchain:** The tools we used in the development of this software include python 3.6 and many packages associated with it. These packages are as follows; we used Tkinter for the graphical user interface(GUI), we sued numpy and pandas dataframe for the storage and manipulation of image arrays, and we used matplotlib for matlab and plotting functions to display the results of our analysis. The pandas package allowed us to apply the same functions to all the images in the set and store the image pairs for quick analysis and access to image data. The Tkinter package allowed us to create a simple and straightforward GUI that also displayed results in a simple form. The use of MATLAB allowed us to have clear and precise color segmentation that made finding dust pixels easier. Finally, using OpenCV and its functions made coloring the covered areas of the mask generated by MATLAB possible, as well as getting percentages of coverage.

- **Setup:**
  Open command line as Administrator
  - In command line navigate to matlab root:
    - You should be able to find your matlab root directory by opening matlab and typing "matlabroot" into the console
  - In the regular command line(not in matlab) navigate to matlab root:

  ```
  Program Files\MATLAB\R2017b\extern\engines\python
  ```

  - Next run the setup.py to get matlab engine to run in python:

  ```
  Program Files\MATLAB\R2017b\extern\engines\python> python setup.py install
  ```

  Go to Hindsight GitHub repository at: https://github.com/Beckcjb/Hindsight
  - Select “Clone or Download” dropdown and select “Download”. This will download all the files from the repository.
  - Unzip Hindsight-master to location of choice
  - In setup.py file edit line 26 to show current location of MATLAB python engine
    - Replace matlab_path string in line 26 with location of MATLAB extern engines python

      ```
      Program Files\MATLAB\R2017b\extern\engines\python
      ```

  From command line interface navigate to the folder in which the Hindsight software is located. (Default directory named Hindsight-master, can be anything you want)
  Run the setup.py file to get a complete installation of packages and environment path setup:

- C:/User/Programs/Hindsight-master> python setup.py install

- **Production Cycle:**

Running Software:

From command line interface navigate to the folder in which the Hindsight software is located.
Run the index.py file to start MatLab engine and run the Hindsight Software. The following command will start the software

- C:/User/Hindsight-master> python index.py

Once software is running you will see the window from figure 1.3 in the Architecture section of this document. If you need to restart the software and are in the start screen close the window and reset the cmd with “ctrl+c”. Software has exit button, but if something goes wrong or you want to exit the application before it completes its task type “Ctrl + c” in the command line interface. **NOTE:** The window is not interactable if software is running analysis. Not elegant to close this way, we recommend using the exit button on the software. To restart the software just type C:/User/Hindsight-master> python index.py into the command line interface.