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

Software Design

Version 1.1

Jet Propulsion Laboratory Image Analysis

Client: Iona Brockie

Team Hindsight

Charles Beck, Alexanderia Nelson, Adam Paquette, Hunter Rainen

February 26, 2018

Mentor: Austin Sanders

Table of Contents

1. Introduction

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 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. JPL has had a hand in a wide variety of scientific missions. From launching satellites that study Earth, aiding in natural disaster relief (e.g. NASA–ISRO Synthetic Aperture Radar, or NISAR) to other space vehicles that do Astronomy (e.g. Kepler, Voyager, Cassini, etc.). JPL's missions provide insight for better understanding how our home and our universe works making it a better place for everyone to live.

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 which will then be picked up by a future mission.

One of JPL's current projects is the Mars 2020 (M2020) rover, an upcoming mission to Mars that will explore various regions of the Martian surface to search for evidence of past life on Mars. The significance of finding past life on Mars would provide more insight to the development of humans and all species on Earth. If we are able to find evidence of Martian life, it would create many opportunities for space exploration.

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's solution to the dust problem is to blow the dust out of the hole using compressed air. However, JPL's testing of this dust removal tool is slow and manual. For some of these tests, the team at JPL mimics the Martian atmosphere by drilling in a pressurized vacuum chamber. This is a time-consuming process because they have to

bring the vacuum chamber back up to Earth's atmospheric pressure, examine the results by hand, and then bring the chamber back down to Mars atmospheric pressure again to run more tests. Furthermore, because they are having someone review and analyze each test by hand, results are less consistent and are less accurate as they are subject to human error and human bias for what looks like dust. Ideally, the team at JPL wants to automatically analyze their tests while under pressure in the vacuum chamber using only the cameras.

Team Hindsight's solution to JPL's problem is an automatic image processing pipeline. Our tool will make selecting and analyzing images taken in their vacuum chamber easy for JPL's engineers. The solution we envision will have 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. 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.

2. Implementation Overview

To help JPL automatically test their dust removal tool, Team Hindsight's solution is to build an image analysis pipeline that will take in an image of an abrasion covered in dust and output a number for how much of the abrasion is covered in dust. This will allow JPL to run multiple tests in one pressurize and depressurizing session as well as automate the marking up of dust in images. Figure 1.1 illustrates the current process that JPL is using and Figure 1.2 shows how the process will be improved using our solution.

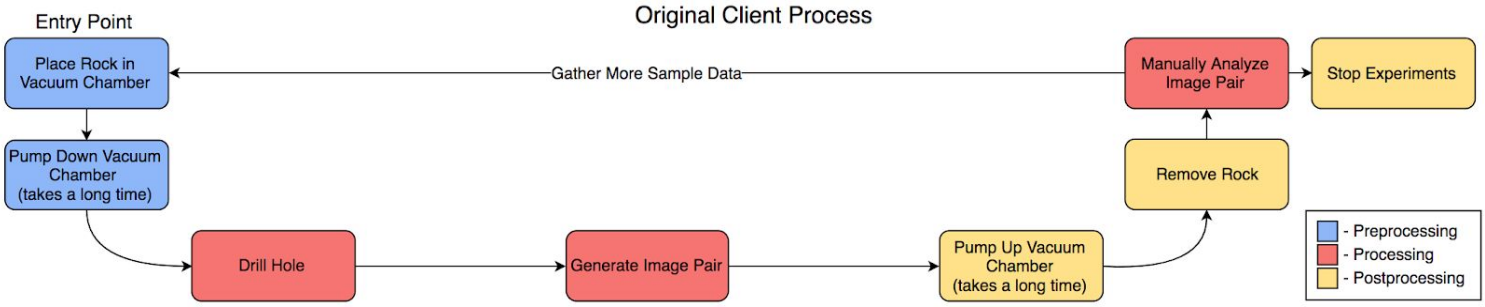

Figure 1.1: Old process from JPL

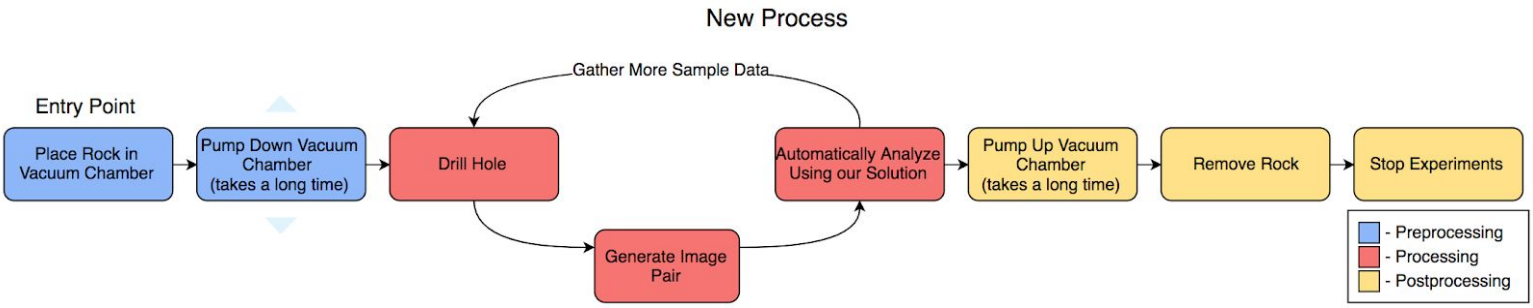

Figure 1.2: New process from our solution

2.1 Tools, Languages, and Libraries

For our solution to work we have integrated multiple libraries to assist in the analyzing of the images and connecting the Model, View, Controller architecture. We will use the following tools, languages, libraries to help us create out software system:

Python 3.6: Version 3.6 is the most current python version at the time of writing this document, we have elected to use this language for a number of reasons. It works well with OpenCV, MATLAB, and has a vast number of libraries to facilitate the input and output of data.

Tkinter: Tkinter is a library that handles the graphical user interface (GUI) that allows us to create a communication line between the user and the back end of the program. This library also allows us to display results to the user and lets the user make changes to the analyzing process. We have chosen to create a GUI so that even a user with little to no programming knowledge can interact with our system.

Pillow: Pillow (PIL) is another python library that gives us the ability to display, save, and change images files in the ".jpg and*.png file formats. This is used because Tkinter originally only handles *.ico and *.gif image files and is incompatible to other types, In order to carry out our analysis protocols we need our front end to handle the same .jpg and .png files. Using PIL, we pass the image file paths to Pandas for storing and displaying images.

Pandas: The pandas library handles the majority of our controller's functionality. Pandas will store image model pairs at each row in a dataframe. While our system will be dealing with many image models, pandas gives the controller the ability to operate on all of image model pairs simultaneously. With our models stored in this dataframe, the controller will be able to select image pairs to run through a set of functions, or run all image pairs in the dataframe through a set of functions.

MATLAB: MATLAB is scientific software used by professionals in the industry (including JPL) that has a variety of tools. In particular, for this project, the color segmentation tools/application has proven to consistently find dust for rock types where the dust is a different color from the rock.

OpenCV: OpenCV is the library that we are using to conduct images analysis. It introduces tools to change image color, hue, and other properties of the image. OpenCV also contains tools to carry out image transformations such as image subtraction, segmentation, histogram analysis, and heat mapping.

3. Architectural Overview

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 architecture that consist 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.

.

3.1 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.

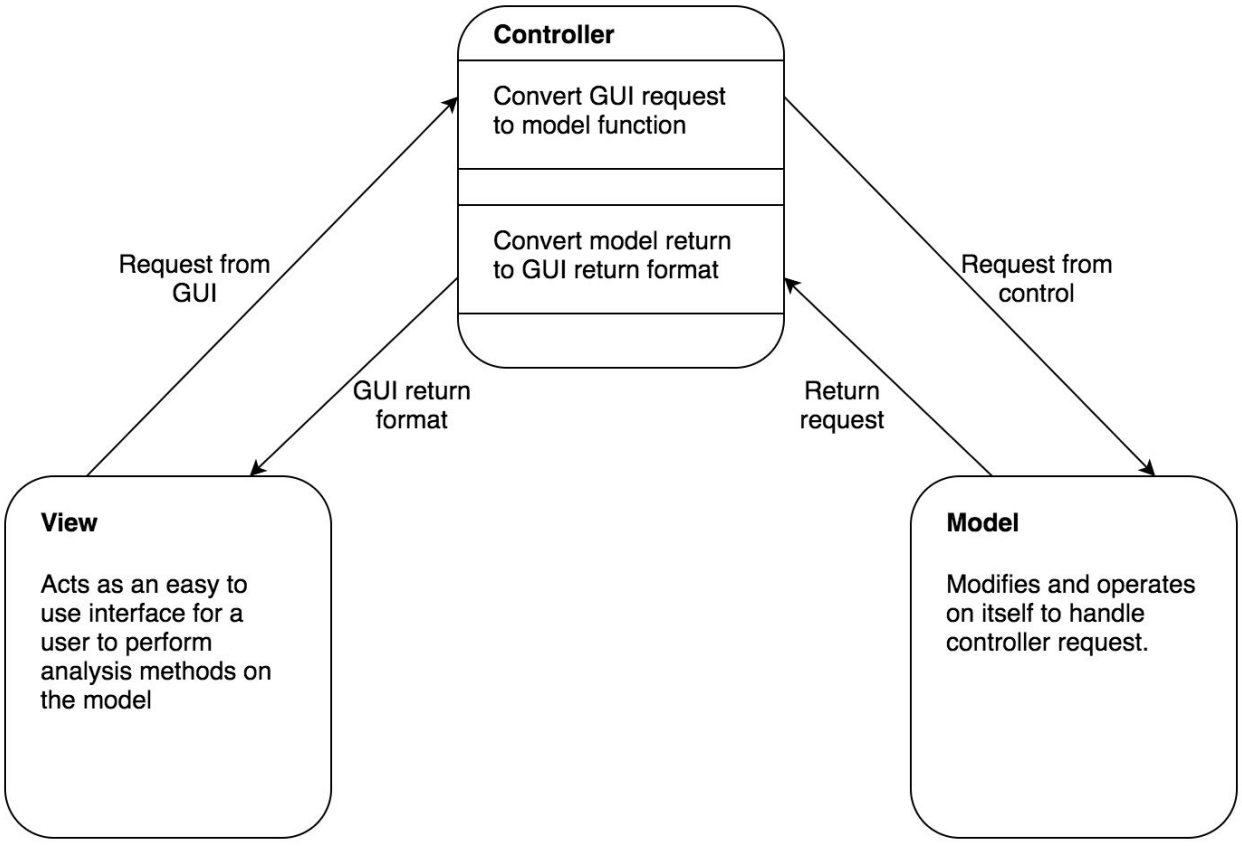

Figure 2.1: Overview of architecture

3.1 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.

3.2 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.

3.3 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. Module and Interface Descriptions

Now that we have introduced our systems architecture the important details are contained within how we have created this system and how each portion of the structure interacts with one another. Connecting the pieces of the MVC give an understanding of the overall solution and here we outline exactly how these modules work .

4.1 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 control, which in turn updates the view.

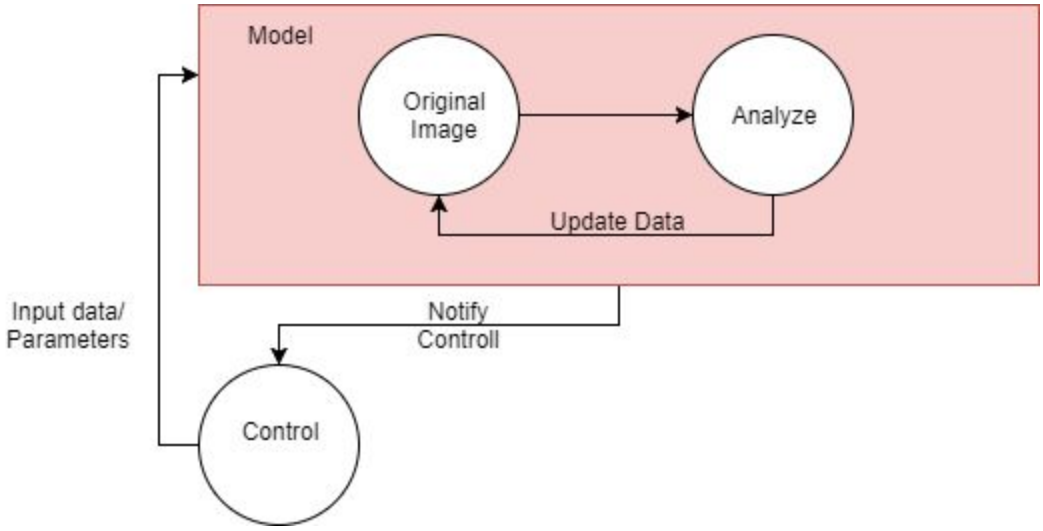

Figure 2.2: Model Visual

4.2 View Description

The view section of our MVC system is a graphical user interface 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 change 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

(before image), 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 graphical user interface (GUI) has been written using simple tkinter libraries 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 anywhere so it was important that we had a basic interface that could interact well with the rest of python 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 my typed into, the user will type in the file handle and then press select to move onto the next screen.

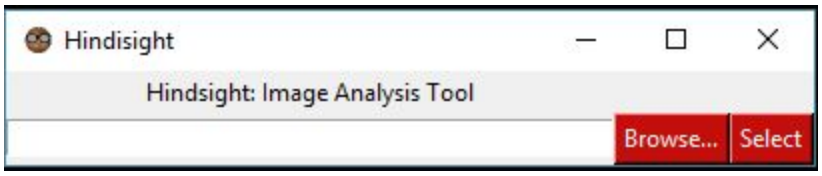

Figure 2.3 GUI File Select screen

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.

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_before”, “abraded_030_after1”, and then “analysis_030_output”. This allows the user to view the original before and after images alongside our programs output. There are also selections on the analysis window that will allow the user to run just a color segmentation, image subtraction, or heat map analysis .The user can use these selections to the complete image analysis that involves all three steps.

If the user is satisfied with the result, then they can select the save button and the analyzed image will be saved to a directory of their choosing. The user can select

the “Change File” button to change the file directory for the image set they want to use. 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.

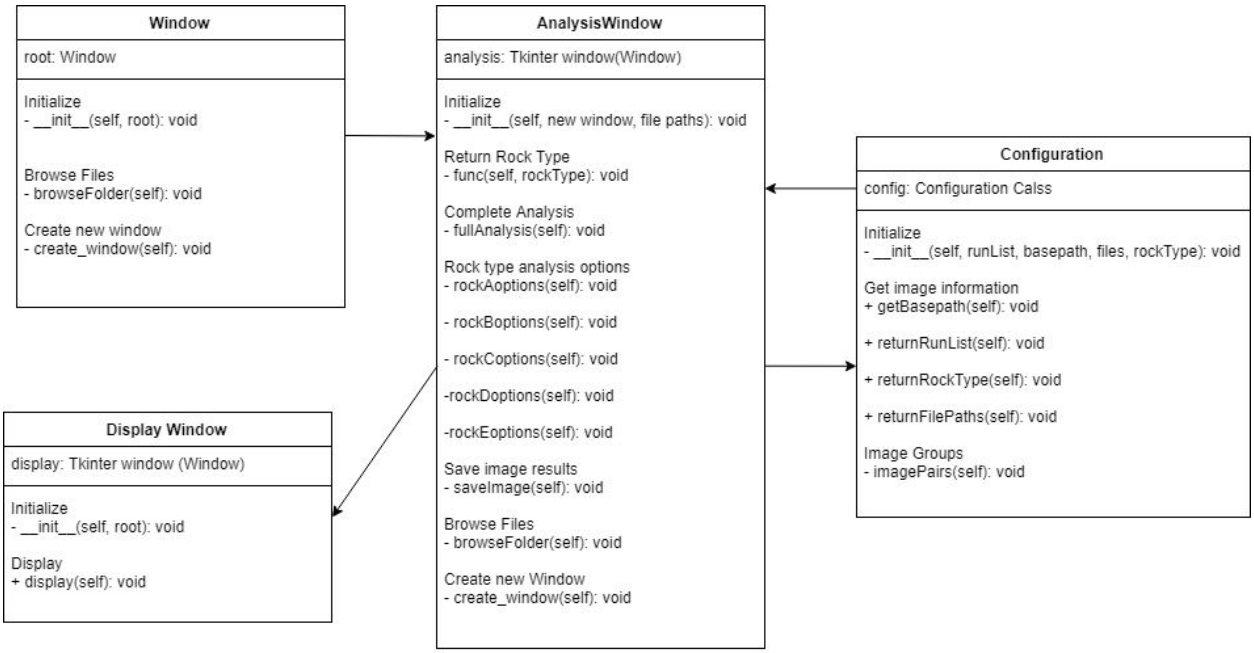

Figure 2.4 : UML for View (GUI)

4.3 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.

	after_image	before_image	image_group	output_image
0	<image.image.Image object at 0x10601a6d8>	<image.image.Image object at 0x10601a128>	abrasion020_abraded.JPG	None

Figure 2.5: 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 2.5).

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.

4.3.2 UML Diagram for Control Module

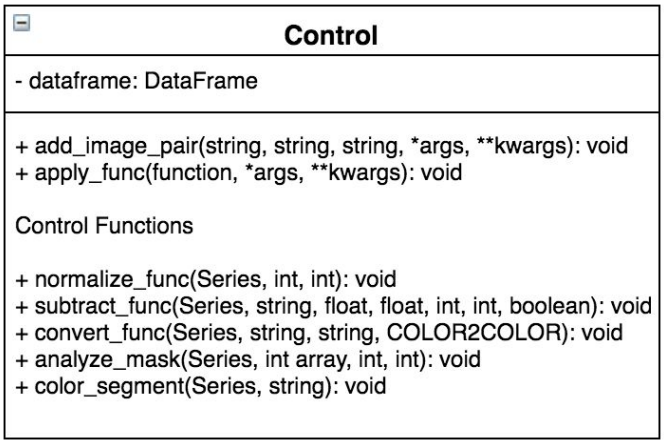

Figure 2.6: UML for control

4.3.3 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 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 control 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 2.6. 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.

4.3.4 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 image models based on the prefix of each image in the image list, where the prefix for both images in the pair is “abrasion<number>_” followed by “abraded” for the first image in the pair and “after<number>” for the second image in the pair. The image 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.

5. Implementation Plan

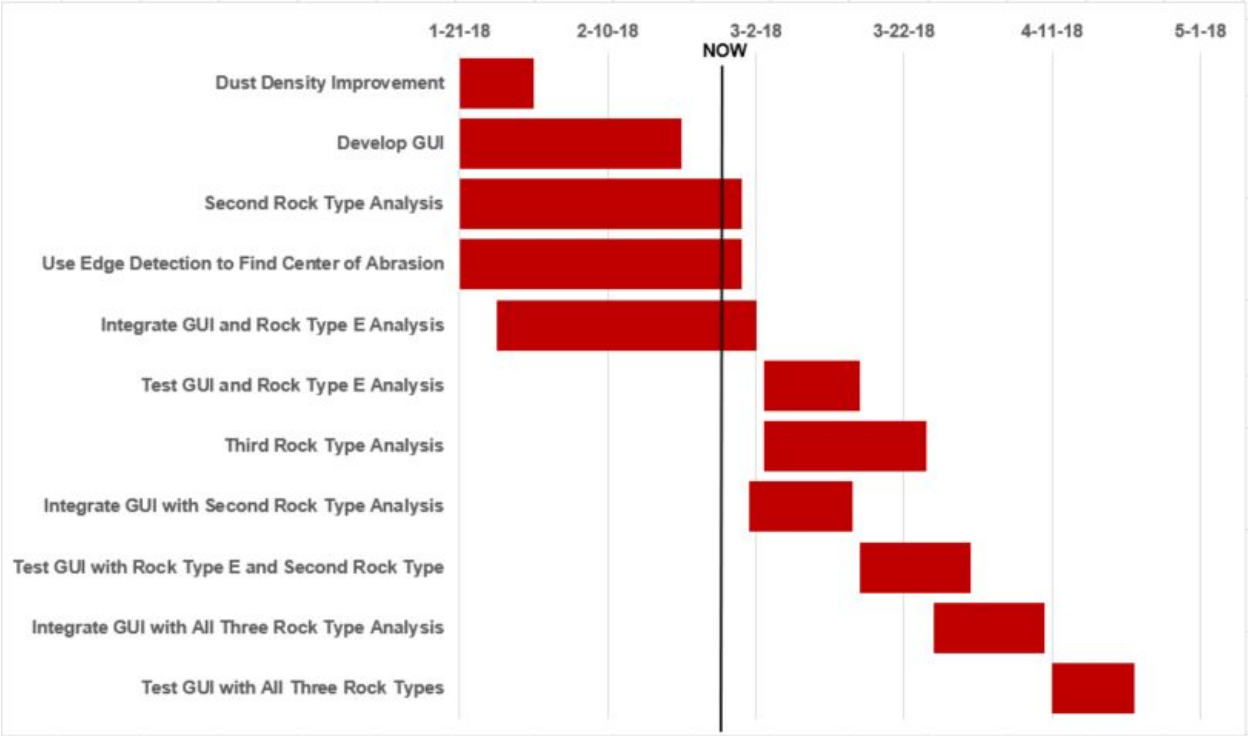

Figure 3: Design-centric Implementation Schedule

This section will focus on the timeline of how we plan to implement each component discussed in the earlier sections and focus on some key items. Figure 3 shows the contingent schedule for the spring semester looks like for our team. Here are the current items that we are developing to implement:

1. Dust Density Improvement - The main goal of this step was to improve on how the script for marking up the dust density in the image using red, yellow, and green colors to indicate whether the area was completely in dust (red), had some dust (yellow), or was dust-free (green). The result would be a heat map that we would compare to the marked up images given to us by our client to meet one of our requirements, which was having our results be within ten percent of the images from our client's business. The improved heat map images are closer to the original resolution of the image before it was processed and is a better representation of the dust density in regions of the image.

2. Develop GUI – Using the prototype of the GUI made last semester and the sign off from the client, we are working on a GUI that will be combined with the rock

analysis. The GUI works in its base form and is being converted from Python 2.7 to Python 3.6.

3. Find Center of Abrasion – Assuming that we will know the pixel to centimeter ratio based on the camera parameters, we plan to find the center of the abrasion using edge detection techniques. The algorithm that would be most useful is the Canny abrasion since it's the least likely to use weak edges and applies a Gaussian filter on the image given.

4. Second Rock Type Analysis –We have implemented a prototype for Rock Type E so we have started to work on a dust analyzing for a second rock type. The second rock type is Rock Type B, which was described by the client as being very white for the dust and rock. There's currently a variation of the color segmentation being implemented for the second rock type since this method worked best with Rock Type B.

5. Additional Detail on Scheduling – Integrating and testing both the GUI and analyzer depend on when the rock type analysis gets done so the schedule may shift to reflect any difficulties that may arise. The first integration and testing, with Rock Type E, can only be started once the GUI is finished developing since we have the analyzer for Rock Type E currently done. We decided it would be best to focus on three of the five rock types due to the nature of the other rocks as well as attempting to get as many accurately analyzed rock types as possible. This will allow us to give our client a useful tool that they can use and possibly modify for the other two rock types if they wish to in the future.

6. Conclusion

In conclusion, JPL is a NASA research 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. The significance of finding past life on Mars would provide more insight to the development of humans and all species on Earth. Specifically, the primary goal of M2020 will be to look for evidence of past life on the Mars by analyzing and collecting samples of the Martian surface which will then be picked up by a future mission.

One problem the team at JPL has encountered with drilling is the dust generated from their tools. When the rover drills into a rock, it creates dust that obscures the inside of the rock. To remove this dust, JPL uses compressed air to blow dust out of the hole. Team hindsight job is to create software that looks at the before and after images of dust blown out of a given hole, and analyze the pictures to see how much dust was removed and how much dust still covers the surface of the drilled hole.

Our software will use the Model, View, Controller (MVC) architecture as it is intuitive to understand and works well with our envisioned solution. The Model is responsible for taking in image data, analyzing it, and updating/storing any insight gained from analyzing an image. The View is the output the users will see through the GUI. Users can control what images are analyzed and how they are analyzed through the view. The Controller will communicate between View and Model and will update each accordingly.

Team Hindsight is on track to finish analysis of one rock type (out of five) by the end of February, and plan to have a working (but rough) prototype for a second rock type soon after. The successful completion of our project will save JPL countless days of testing and help the team better understand the effectiveness of their dust removal tool. JPL will launch the Mars 2020 rover confident that their scientific instruments will be able to analyze rocks free of dust filled holes.