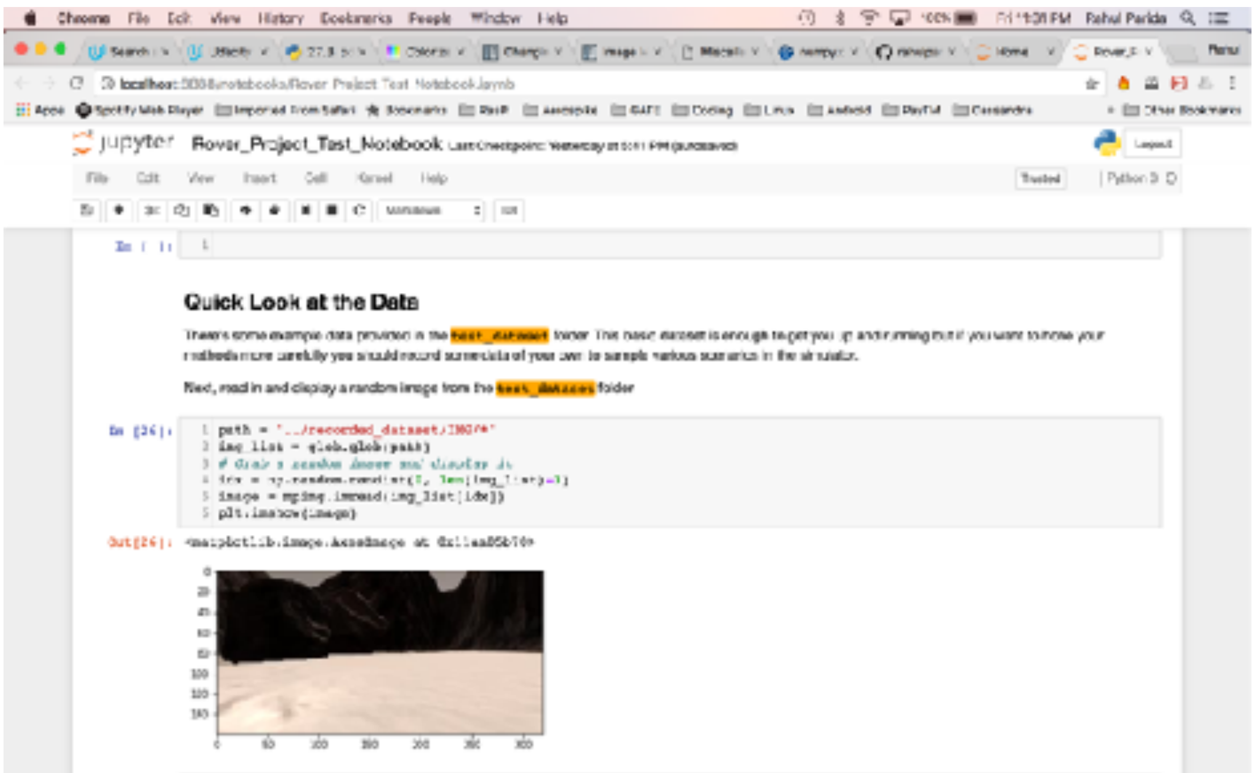
Search And Sample Return Project

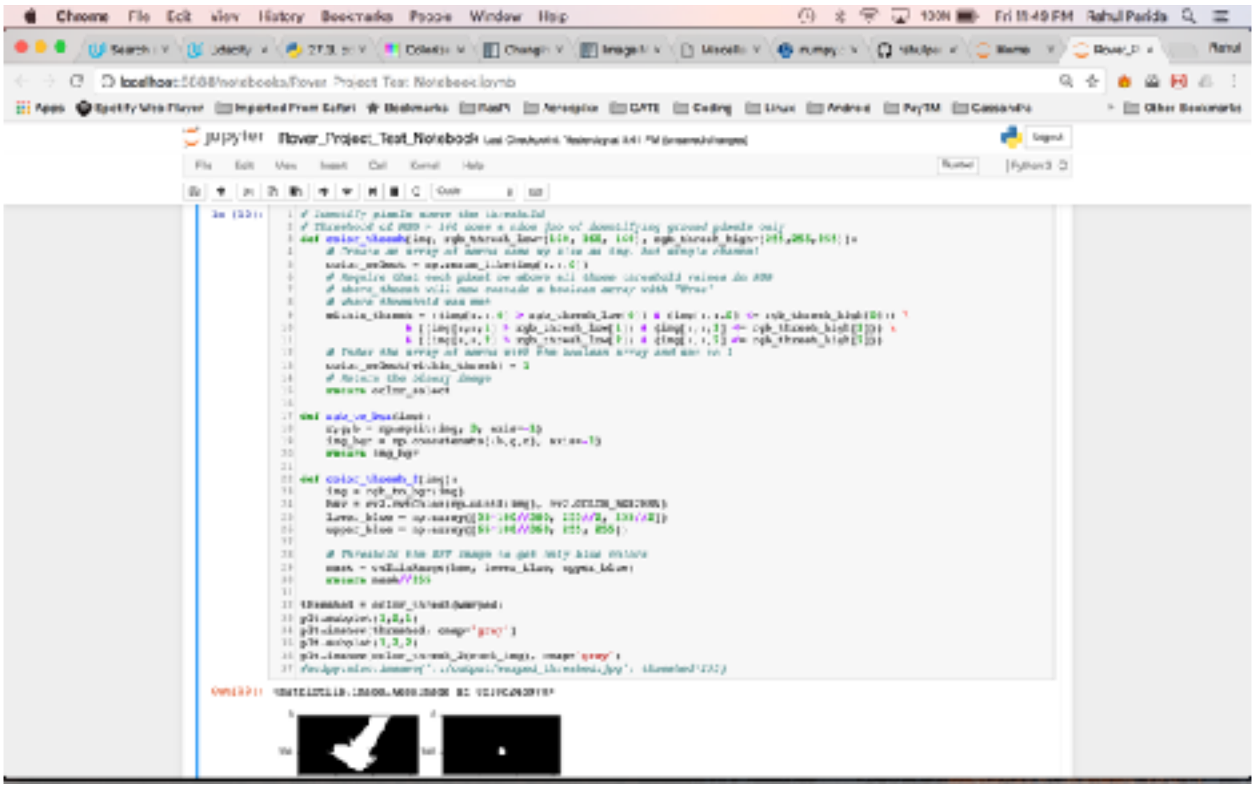
Notebook Analysis

1. After recording data in training mode, I changed the path from which the images were being fetched i.e. changed *test_dataset* to *recorded_dataset* in the following code in the notebook.

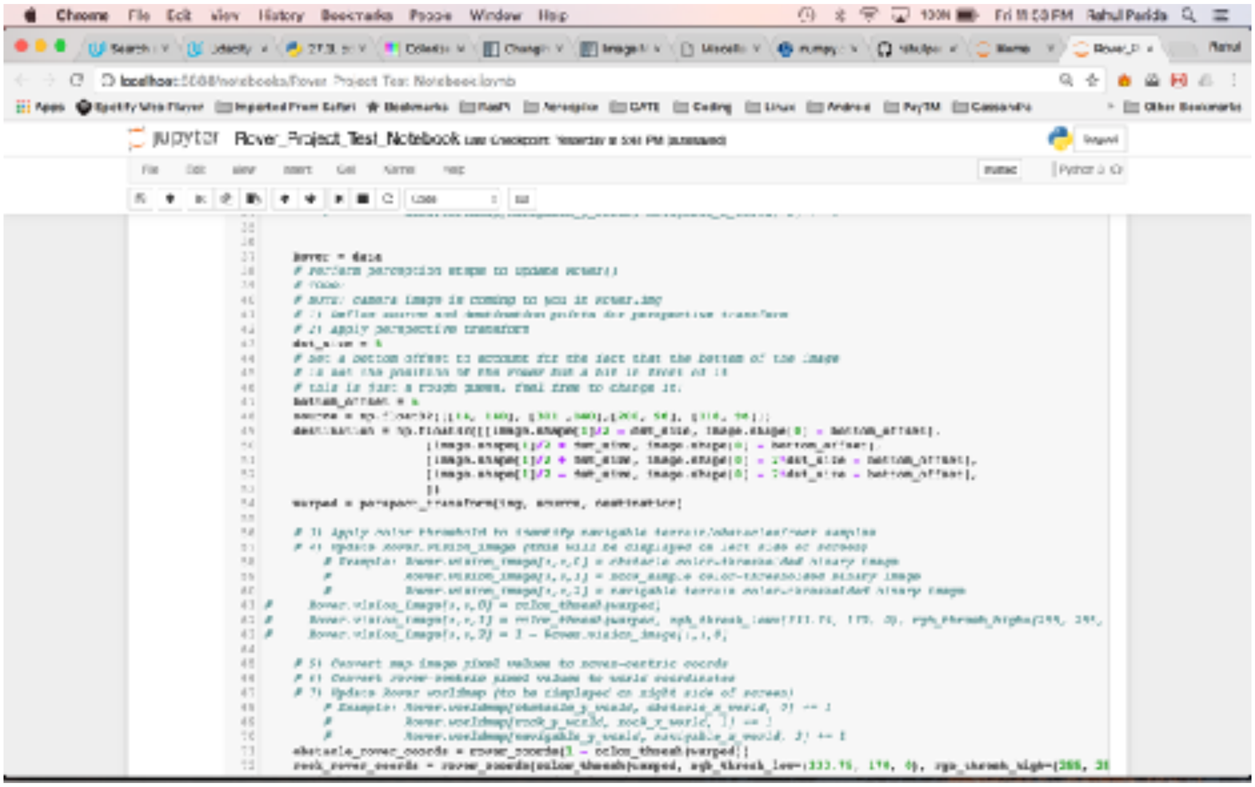


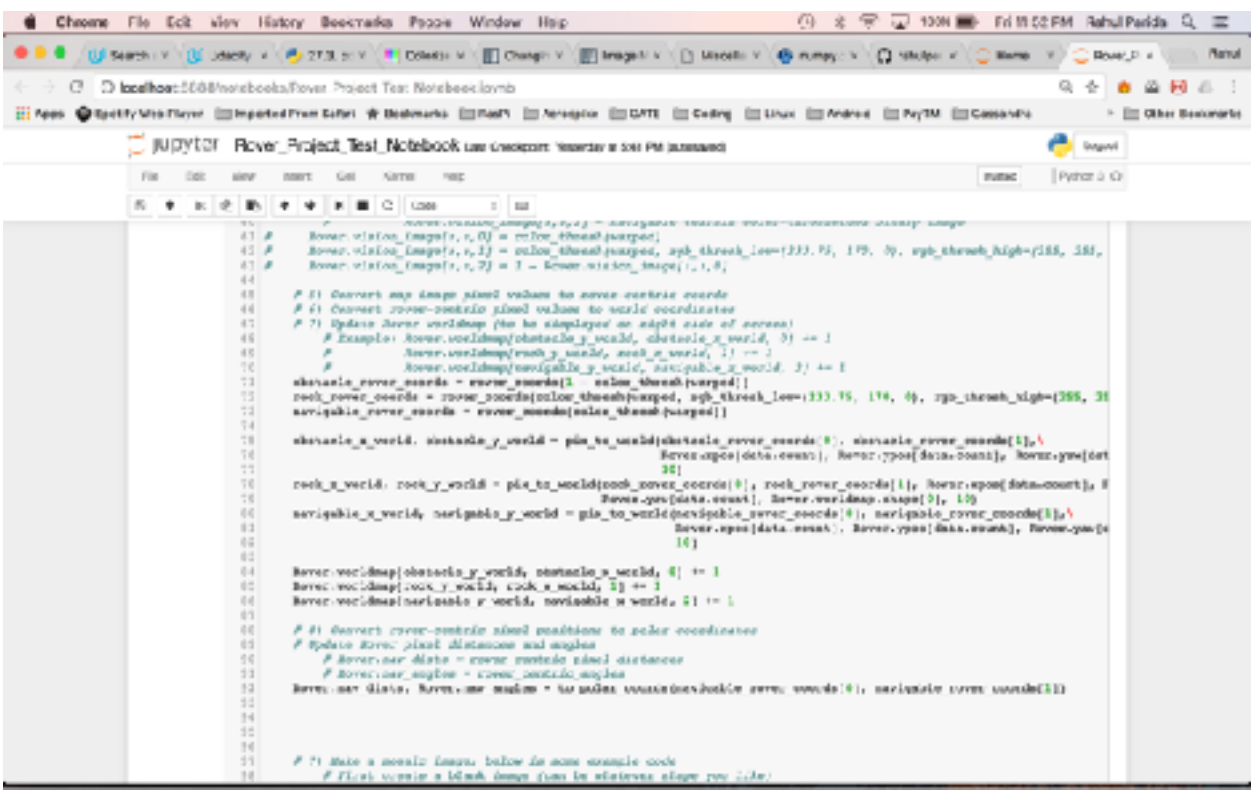
2. Then I used both the calibration images *grid_img* and *rock_img* for finding out the approx coordinates at the edges of the grid square and finding the color range for the rock samples. The approx pixel coordinates were used for the perspective transform

3. For the color thresholding, I modified the function to accept extra parameters which applied threshold on the RGB values on the image according to the range provided in the parameters. I also defined two other functions *color_thresh_2* which applies the color thresholding using opencv and *rgb_to_bgr* to convert RGB to BGR channel config to apply to the previous function. The color values for shades of yellow are hardcoded (values calibrated manually).



4. In the *perspect_transform* function I added the following code. I have initialized *Rover = data* so I could reuse the code in *perception.py.* The video obtained as a result of running moviepy on my recorded dataset is present in the folder *outputs.*





Autonomous Navigation and Mapping

Perception Step

The following are the operations that I have performed in the perception step.

1. Applied perspective transformation using the *perspect_transform* function which was already provided to convert the camera view to a map view after defining source and destination points for mapping (line 115)
2. Applied color thresholding for detecting obstacles (line 122) which can be calculated as the complement of the color threshold applied to navigable terrain i.e. *color_thresh(warped).* Took the above result's complement for finding out the navigable terrain.
3. Applied my custom thresholding function *color_thresh_2* for detecting rock samples. This function is the same one that was used in the notebook.
4. Obtained rover-centric coordinates for all the three images.
5. Applied transformation and rotation to the above coordinates for getting world map.
6. Then I have defined a threshold of 1.0 degrees for roll and pitch angles for increasing the fidelity while updating the world map.
7. I have also defined two new attributes for the *Rover* object i.e. *rock_dists* and *rock_angles* for keeping track of the rock samples detected.

Decision Step

I have modified and added several pieces of code in the *decision_step* function for picking up rocks and for detecting and recovering from a state where the rover is stuck.

1. The logic when the rover remain in *stop* mode remains the same, however, I have moved the function to the top in *if-else* as *stop* mode now also serves as a way to recover from the state when the rover is stuck and unable to move.

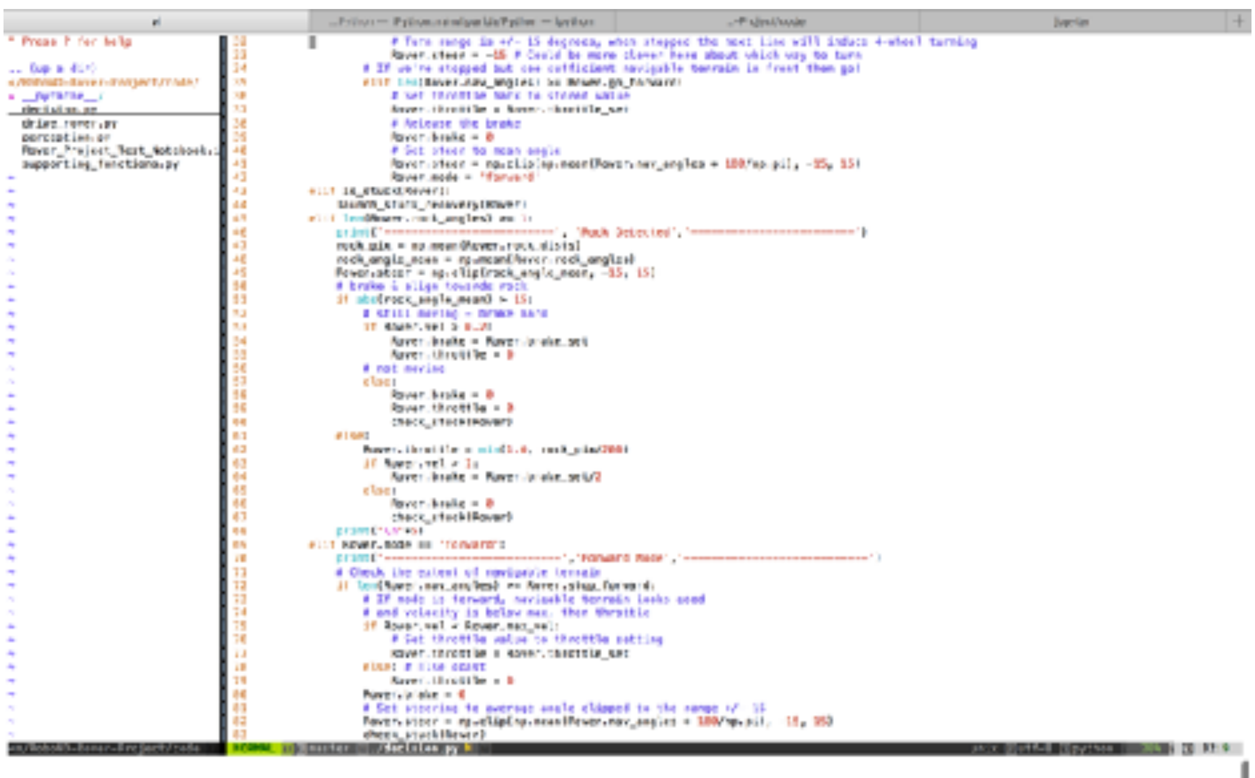
2. The next condition checks whether the Rover is stuck and if it is, it invokes a function called *launch_recovery.* In my code, I detect if a Rover if stuck by analyzing the *Rover.vel* and *Rover.throttle* values. If the *Rover.vel* value is below the threshold *0.01* and *Rover.throttle* is set

to max *1.0.* Then I assume that the Rover is stuck and therefore launch stuck recovery. In stuck recovery all I do is to apply the brakes and set *Rover.throttle* and *Rover.steer* values to 0 and change the *Rover.mode* to *stop.* The same set of operations which were applied when there was a dearth of navigable terrain in *forward* mode. The *stop* mode handles the rest.

3. I have then added a condition which modifies the Rover's behavior if a rock is detected. The *Rover.steer* value is set to the *mean* of *Rover.rock_angles* which I had captured in the perception step. I also maintain a *mean* of *Rover.rock_dists* using which I set the throttle values later.

A. If the location of the rock is present out of the bounds in which the Rover can turn and the Rover is moving at a high speed (> *0.2* threshold), I apply the brakes and set *throttle* to 0. Or if the Rover's velocity is lower than the above threshold, I just set the *brakes* and *throttle* values to zero. This enables the Rover to stop and align itself towards the rock sample.
B. If the location of the rock is present within the bounds in which the Rover can turn, I try to maintain *Rover.vel* < *1* by braking when it exceeds the threshold. I also set the *throttle* value to be the inverse of the mean rock distance from the Rover (rock is closer implies that the throttle should be low).

At every point when the speed is low, I invoke the *check_stuck* function which checks if the *Rover.vel* lower than the threshold (< *0.01*) and sets the *throttle* value to max if it is. This subsequently helps in checking whether the Rover is stuck.



4. The code present in *forward* mode is the same as before.

5. I have also added a small snippet at the end of the *decision_step* which brakes hard when the Rover is near a rock sample and has a very high velocity.

All the above changes enable the Rover to recover maximum rock samples without compromising fidelity of mapping.