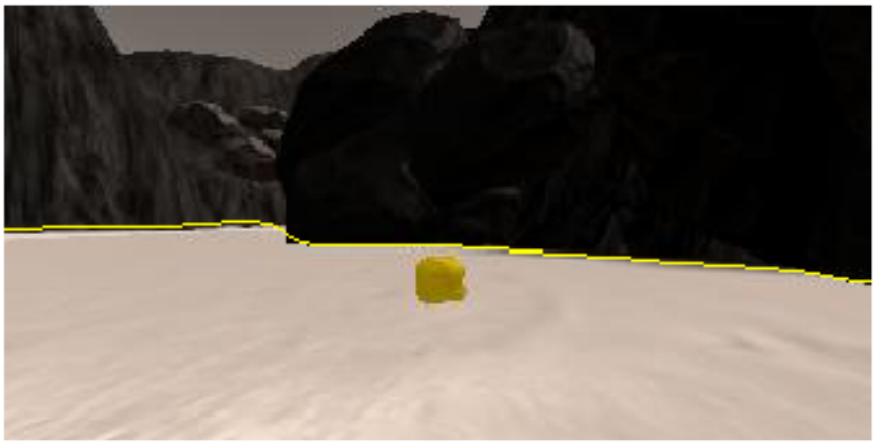
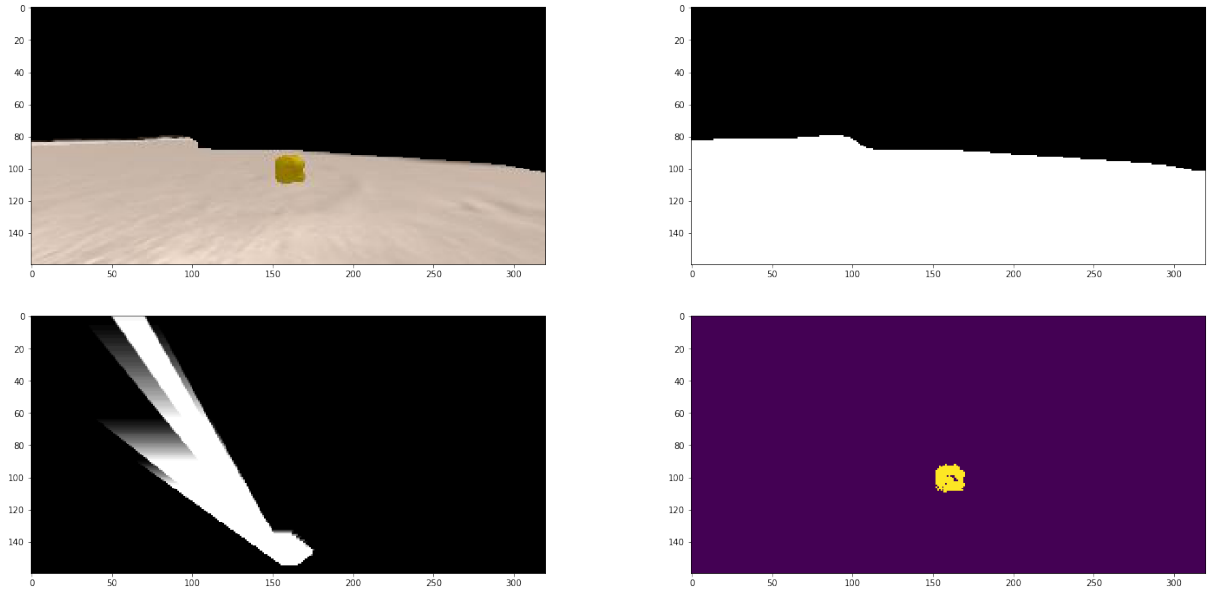
RoboND-Rover-Project Report

Notebook Analysis

1. Run the functions provided in the notebook on test images (first with the test data provided, next on data you have recorded). Add/modify functions to allow for color selection of obstacles and rock samples.

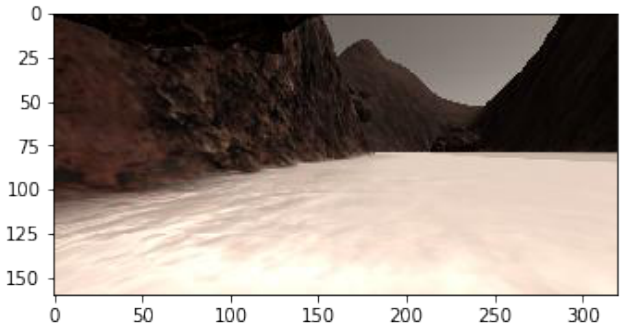
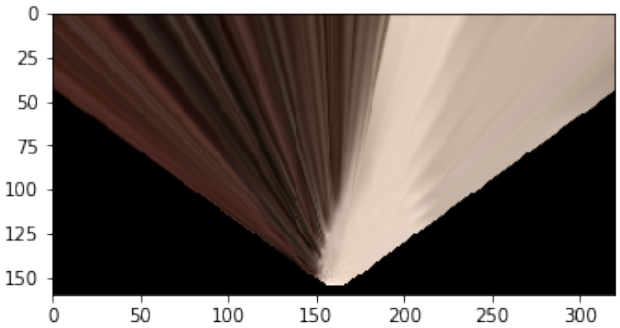
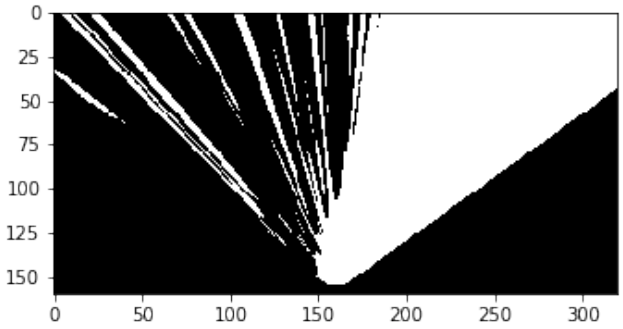
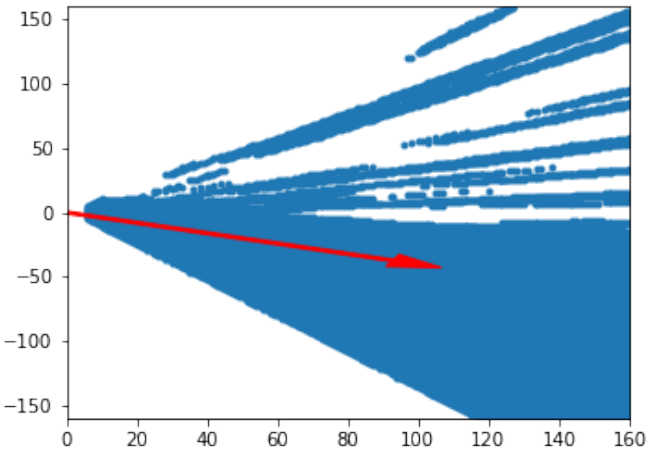
The first function created was based on Segmentation on superpixels. Through the SLIC algorithm i planned to separate the scenne in two regions and make the analysis on it. This approach presented some issues: poor handeling of shadows, the code got slow and when the rover was trap with obstacles it makes a lot of iterations without solution and de script code dies. In the other hand, we could obtain over 60% of mapping and 40-50% of fidelity.





As we can see in the image above, i could separate the rocks succesfully. To make this possible i wrote two function: the first one is just for detect purpose using a threshold value to separeted them. The second one is to separate the rock from the enviroment when get detected.

The second implementation was though to be a complement for the above function and the color_thresh function that i received with the notebook. This segmentation algoritm its a bitwise operation between two different algorithms over the HLS channel color. I detect that working on the Luminance channel we could get better posibilities to improve the contrast of the image. Tha was possible using the CLAHE algoritm. By the other hand, i used the OTSU algorithm to segment in a binary way the image. Its important to specify that i used the functions availables on OpenCV. The results are shown below:

2. Populate the `process_image()` function with the appropriate analysis steps to map pixels identifying navigable terrain, obstacles and rock samples into a worldmap. Run `process_image()` on your test data using the `moviepy` functions provided to create video output of your result.

For this part i tried to follow “the steps” inside the function. To map the pixels the only thing you must do is to keep in mind the order of the process (get coordinates, transform to polars and get angles and distances) and do not forget the three main parts of the map: free space to navigate, obstacles and rocks. Following the class videos and the throughout video made me understan the concepts behind the code.

For the video with moviepy, i have gotten some wear mistakes about the code functions, so i prefered to test the code ongoing on the simulator environment

Autonomous Navigation and Mapping

1. Fill in the `perception_step()` (at the bottom of the `perception.py` script) and `decision_step()` (in `decision.py`) functions in the autonomous mapping scripts and an explanation is provided in the writeup of how and why these functions were modified as they were.

First, we have to change the data structure from data to Rover. In addition to, there are some other parameters that were necessary changed to succesfully complete the project. To see the areas in the map, i must added *Rover.vision_image* to see the navigable terrain (Blue) , obstacles (Red) and rocks (White). I think the main change compared with the jupyter notebook was to add the roll and pitch angles to avoid the "dancing on the rover movement" and improve the fidelity as the class intruction suggested. And lastly, i tried to make some control through the angles information to make a easiest drive stright forward to the rock. To do that, i tried to store the angle and rocks distance from the rover to tell him where to go and limit its vision range.

On the *decision_step()* i was not capable to make the rover picking up the totally rocks. I achive to pick one rock but when it dropped, the rover does not continue its path (It just get stopped in that position). So, i left the code commented and left the the rover Located the rocks without picking them up to complete the requirements of the project.

2. Launching in autonomous mode your rover can navigate and map autonomously. Explain your results and how you might improve them in your writeup.

As i mentioned before, the code its capable to map the environment 40-70% with a range of fidelity 60-85%. The first parameter depends on the rover origin, velocity and frames per second of the simulator. The second one, mostly depend over the roll and pitch angles control, getting better results when we only mapping when they are to small. In summary, the rover will completed the requirements of the project.

There are two main parts to improve:

- The perception algorithms and obstacle avoidance. Sometimes if you crashhed with some rock, the rover does not turn around, it just tried to continue driving because the mapping suggest him that there are paths to follow. Also, sometimes, we could get trapped driving in circles because the rover is sensible to shadows and in the map (Using a medium measure of the "free space" to follow) this will be the only available path. This is also because it doesn't take into account where the rover has already been to prevent go to completed visited areas and not actively looking for new regions of the map to explore.

- Picking up the rocks: Keep information about the origin of the simulation. Keep information about the number of rocks picked up and its localization on the world map. Improve the straight forward driving and fix the bug that does not allow the rover continue its path after picking up the first one.