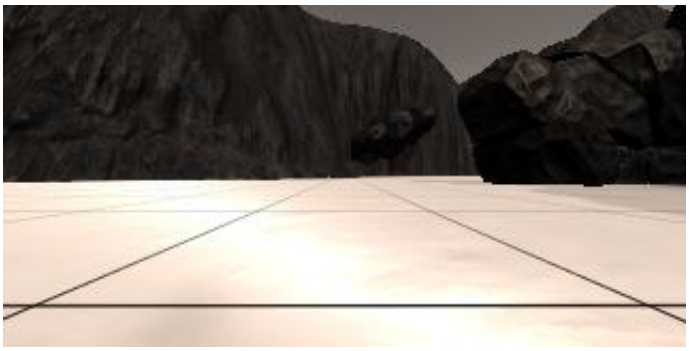
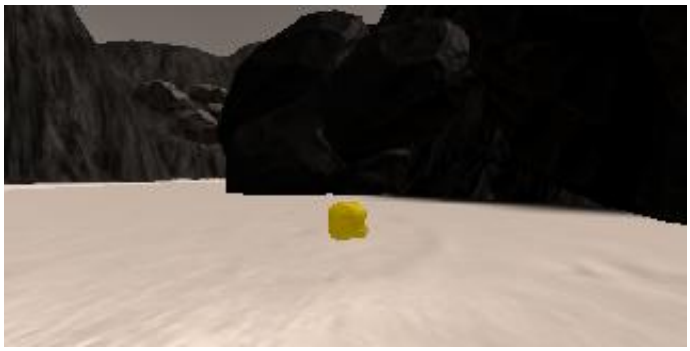
Writeup / README

Beshari Project Write Up

Notebook Analysis

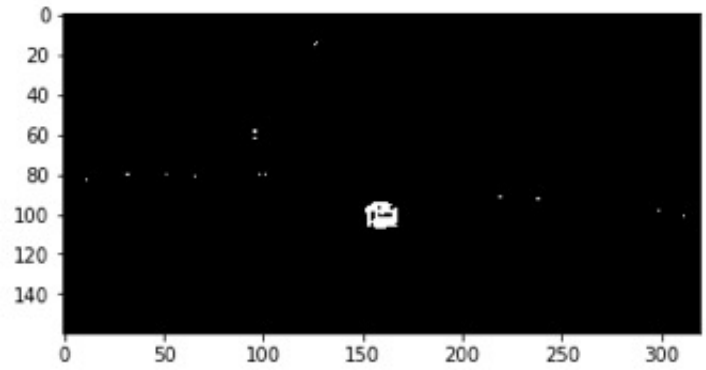
1. Run the functions provided in the notebook on test images (first with the test data provided, next on data you have recorded). Add/modify functions to allow for color selection of obstacles and rock samples.

Test Images

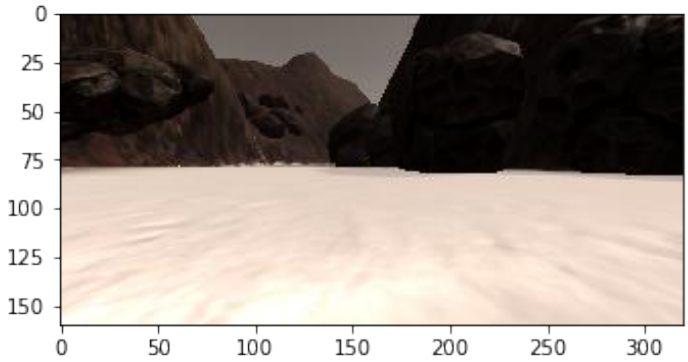



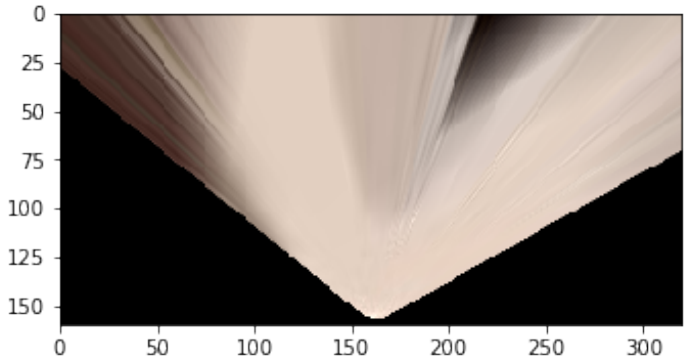
A new function was added to the notebook:

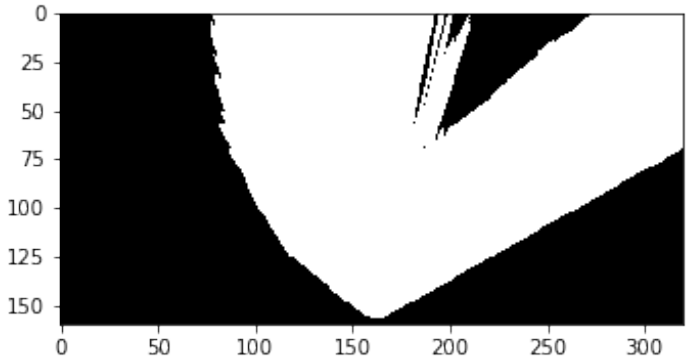
```
def color_thresh_rock(img, rgb_thresh=(20, 0, 10)):
    # Create an array of zeros same xy size as img, but single channel
    color_select = np.zeros_like(img[:,:,0])
    # Require that each pixel be above all three threshold values in RGB
    # above_thresh will now contain a boolean array with "True"
    # where threshold was met
    above_thresh = (img[:,:,0] > rgb_thresh[0]) \
                & (img[:,:,1] > rgb_thresh[1]) \
                & (img[:,:,2] < rgb_thresh[2])
    # Index the array of zeros with the boolean array and set to 1
    color_select[above_thresh] = 1
    # Return the binary image
    return color_select
```

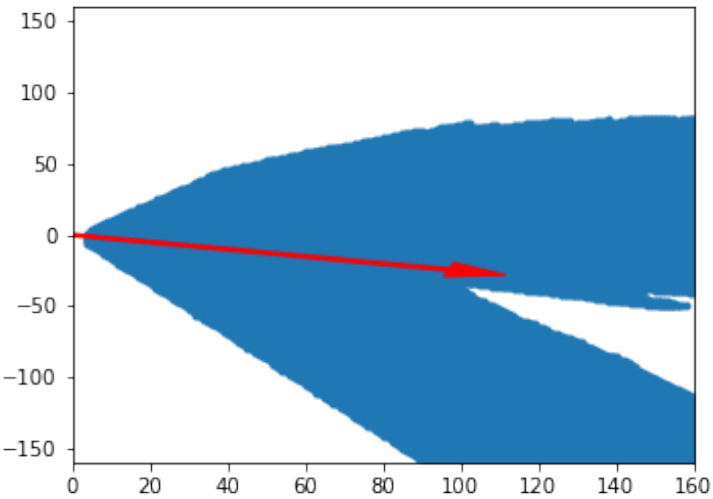


1. Populate the `process_image()` function with the appropriate analysis steps to map pixels identifying navigable terrain, obstacles and rock samples into a worldmap. Run `process_image()` on your test data using the `moviepy` functions provided to create video output of your result.









and we got a movie result (Please see in the Output Folder named **test_mapping**

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.

Ther perception step has been populated and modified accrodingly, in short: a new function wass added, the same `color_thresh_rock` function mentioned above

the data pipeline goes as:

Images were perspective-transformed first, then thresholded, thirdly, the pixels were rotated in perspective of the Robot, fourthly, angles and distances of the pixels are calculated in polar coordinates based on the terrain thresholded, warped picture, and finally the picture is mapped on the world map

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

The results were satisfactory: a lot could be done better, however, The robot (has a name in the code"RoboBesh") navigates autonomously with very little assistance manually and able to "see", map the world

and locate rocks

Better Algorithm could be used in the decision and manoeuvring analysis. If I would pursue this project further, I would write an algorithm that splits the picture in two, left and right and have the robot crawl the walls.

I would also not have the robot to go to pre-visited places where they have been mapped already.

I would also improve the algorithm where the RoboBesh could see and go, but he is not moving physically (Stuck) to do a getting-out manoeuvre from where he is stuck. **Note: running the simulator with different choices of resolution and graphics quality may produce different results, particularly on different machines! Make a note of your simulator settings (resolution and graphics quality set on launch) and frames per second (FPS output to terminal by `drive_rover.py`) in your writeup when you submit the project so your reviewer can reproduce your results.**

1280 X 1024 on fast fps