Search and Sample Return Writeup

Writeup/ README

1. ***Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.***

This document is the writeup for the Search and Sample return project.

Notebook Analysis

1. ***Describe in your writeup (and identify where in your code) how you modified or added functions to add obstacle and rock sample identification.***

The changes that were done to add obstacle identification were in the perspect transform function, to return a Field of View perspective of the camera. If we just inverted the navigable terrain perspective, even navigable terrain could become obstacle. That is something to avoid:

```
def perspect_transform(img, src, dst):

        M = cv2.getPerspectiveTransform(src, dst)
        warped = cv2.warpPerspective(img, M, (img.shape[1], img.shape[0]))# keep same
size as input image

        mask = cv2.warpPerspective(np.ones_like(img[:,:,0]), M, (img.shape[1],
img.shape[0]))
        return warped, mask
```

The changes that were done to detect rock samples were adding a 2nd color threshold function, just for rock sampling. This function consists on thresholding by lower and upper limits, and, by logical operation, it is possible to detect the rock sample:

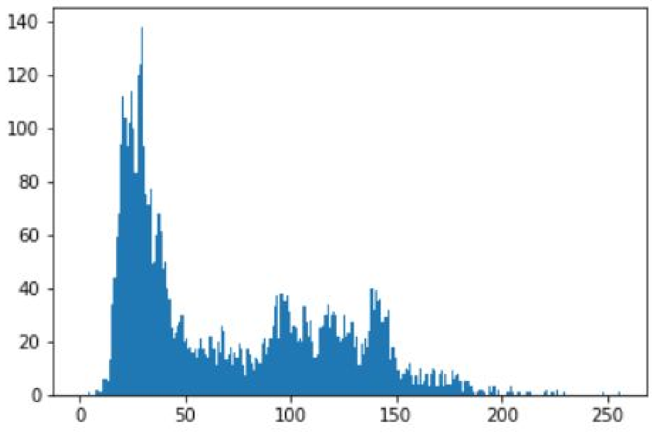
```
def color_thresh_rock(img, rgb_thresh_lower=(160, 160, 160),
rgb_thresh_upper=(250,250,250)):
        # Create an array of zeros same xy size as img, but single channel
        color_select_lower = np.zeros_like(img[:,:,0])
        color_select_upper = np.zeros_like(img[:,:,0])
        # Require 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_lower[0]) \
                & (img[:,:,1] > rgb_thresh_lower[1]) \
                & (img[:,:,2] > rgb_thresh_lower[2])
```

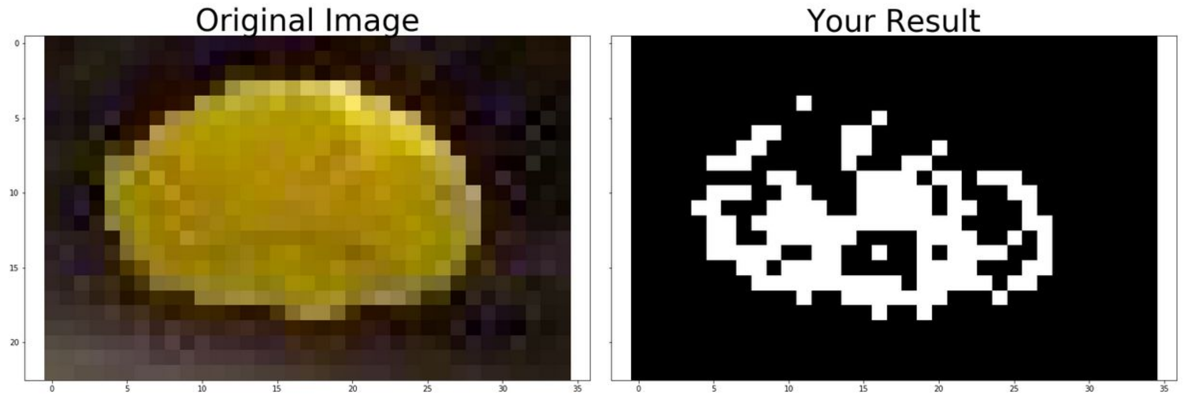
```
below_thresh = (img[:,:,0] < rgb_thresh_upper[0]) \
        & (img[:,:,1] < rgb_thresh_upper[1]) \
        & (img[:,:,2] < rgb_thresh_upper[2])
# Index the array of zeros with the boolean array and set to 1
color_select_lower[above_thresh] = 1
color_select_upper[below_thresh] = 1
color_select = np.logical_and(color_select_lower, color_select_upper)
# Return the binary image
return color_select
```

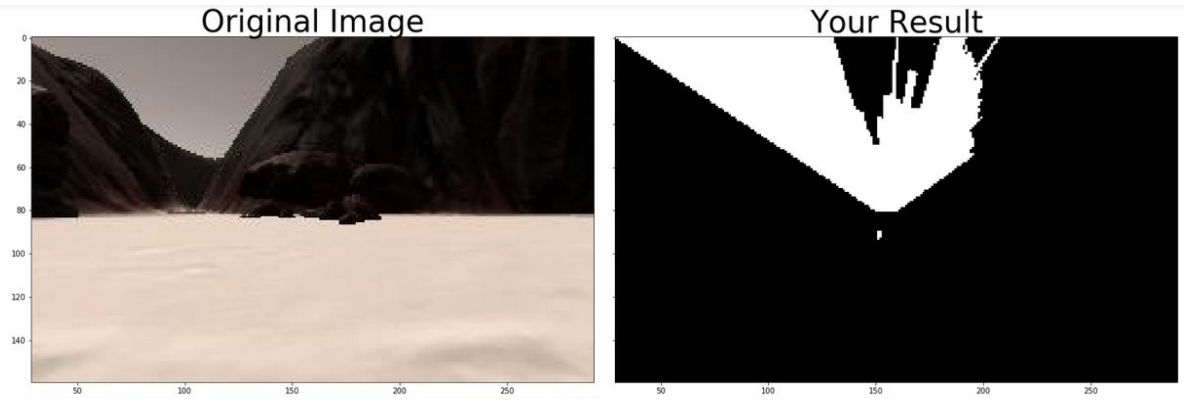
2. ***Describe in your writeup how you modified the process_image() to demonstrate your analysis and how you created a worldmap. Include your video output with your submission.***

process_image was done by following the steps inserted in the python comments. By using all the previous code implemented on the exercises, process_image was populated. Now, to determine which thresholds should be used, a jupyter notebook was created to study statistically which thresholds should be used(this notebook is also present in the submission). By analysis of the intensity histograms of each channel, it is possible to determine approximately the threshold that will give the best results(also some “manual-tuning” may be necessary). The images below show the results that can be observed from the notebook “Thresholds study.ipynb.”

```
Sigma is: 36.5468638491
Total Avg is: 69.5121703854
```







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.***

Regarding the perception_Step(), the previous point explained the basis of it. Everything was kept the same except for the worldmap update. These values were found manually, by trial-and-error, to find the best results. Also, only maps that are within the Rover's pitch and roll threshold, are considered valid, therefore processed. In invalid maps, the Rover maintains the same state as before.(the pitch and roll thresholds were also found after experimentation.)

Regarding the decision_Step() function, the one presented, no modifications were added. There was an attempt of redoing the decision tree(code for this attempt is also present in the submission as decision_2.txt), but it was not presented, due to the fact, that although the mapping is more complete and can finish mapping in less time than the original decision tree, the fidelity is decreased(due to some perspective problems and behaviour problems that it causes aswell).

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 for this are enough for submission. That is, can map 40% of the environment at least for 60% fidelity. Also, it can locate rock samples with ease. But, this is not satisfying results as these can be improved greatly on doing some things differently.

To improve the results, I propose the following changes(will be done over time):

a) In perception_Step(), from the mapping created, create a grid-based map with information regarding navigable terrain, obstacles, rock samples, etc…
b) In decision_step(), by having a dynamically created grid-based map, plan the trajectory for the Rover to follow(A* for easiest implementation). Correct the trajectory as more of the map appears.
c) The grid-based map must be updated regarding if the Rover visited certain areas or not.
d) Since rock samples are located, now pick them up when they are in Rover’s trajectory plan.
e) Increase fidelity by controlling the Rover speed smoothly and deeper analysis regarding pitch and roll thresholding(or create a perspective transform to take into account pitch and roll of Rover.)