Arturo Polanco Lozano
Self Driving Car – Robotics Engineer
Report Search and Sample Return

•***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.***

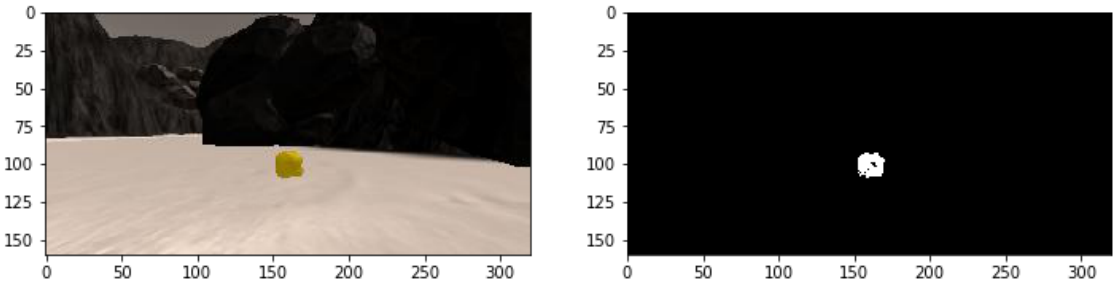
I started adding a mask to the perspect_transform function, there are some areas that are out of the range of view of the camera and the purpose of this is to only visualize information from the navigable terrain.

```
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
```

Then I added the find_rocks function in order to localize in the map the golden rocks using color threshold.

```
def find_rocks(img, levels=(110, 110, 50)):
    rockpix = ((img[:,:,0] > levels[0]) & (img[:,:,1] > levels[1]) & (img[:,:,2] < levels[2]))
    color_select = np.zeros_like(img[:,:,0])
    color_select[rockpix] = 1
    return color_select

rock_map = find_rocks(rock_img)
fig = plt.figure(figsize=(12,3))
plt.subplot(121)
plt.imshow(rock_img)
plt.subplot(122)
plt.imshow(rock map, cmap='gray')
```

The result is binary image with a white region showing the location of the rock.



The rest of the cells where already seen and developed in class.

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

Once a camera image is recieved from the rover the next step is to indentify navigable terrain, rocks and obstacles. The main thechnique used to acomplish this task is to use color threshold, we know the rocks has a specific color, the obstables are darker and the navigable terrain is brighter.

•**Fill in the (at the bottom of the script) and (in) 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.**

Decision.py:

- First checks for Rover's status.
- Then checks if the Rover is moving forward
- If the mode is forward and the navigable terrain is ok and velocity is below the maximum then throttle. Otherwise break.
- If the path of the navigable terrain is not clear then switch to stop mode.
- If the rover is in stop mode but keeps moving then keep braking.
- If the rover is not moving then steer -15, then if the rover can see sufficient navigable terrain in front of it then move forward.

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

The rover could navigate succesfully in the virtual environment, map the path that it covers and identify the rock samples. I found some room for improvement in the following items: In some cases the rover covers over and over again the same part of the map, it would be important to add a method that avoids cover the same path. Also it would be nice to rely the navitagion to a deep neural network, it's going to need good training samples and a convolutional neural network to analyze every pixel in order to detect the action required (something similar to behavioral clonning). Finally the function to return to the starting location could be implemented once the samples are collected (they are only detected and mapped).