Search and Sample Return Project

Describe how you modified or added functions to add obstacle and rock sample identification.

Rock sample -

To identify rock samples which are golden in color I defined a function named rock_thresh which identifies the pixels having value above the specified value of red and green channel in golden_thresh and value below the specified value of blue channel and then returns a binary image.

Obstacle –

To identify obstacles which are dark in color I defined a function named obst_thresh which identifies the pixels having a value below the specified value of red, green and blue channel in obstacle_thresh and then returns a binary image.

Describe how you modified the `process_image()` to demonstrate your analysis and how you created a world map. Include your video output with your submission.

In the process_image function first I applied the perspective transform. Then I applied color threshold to identify navigable terrain, rock samples, and obstacles. Then I converted the identified pixels of navigable terrain to rover-centric coordinates and then converted those rover-centric pixels to world coordinates by performing the necessary rotation and translation and then project it on ground truth map. Now to differentiate between the navigable terrain, obstacles and rock sample in the ground truth map I have assigned red color to obstacles, blue to navigable terrain and green to rock samples.

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.

In the percepition_step() I have added perspective transform, color threshold to identify obstacle, navigable terrain, and rock sample. Then I have added the conversions of navigable terrain pixels to rover-centric coordinates and of rover-centric coordinates to world map coordinates. Then I updated the rover world map and rover pixel distance and angle.

I have made no change in the decision_step ().

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

After running rover in the simulator in autonomous mode the rover is mapping on an average 50% of the terrain with the fidelity of 90%. It is also locating an average of 2-3 rock samples in the map. The rover still can't deal with the obstacle in the middle of the navigable terrain efficiently, it gets stuck. So, we can improve the decision_step to take into account more of such situations.

Specifications of the simulator-

Resolution - 512*384

Graphic quality - fantastic

FPS - 25

Note- I am developing on a windows machine and had to make following changes to the code to get it to work:

In jupyter notebook-

```
df = pd.read_csv('../test_run/robot_log.csv', delimiter=',', decimal='.')
```

In [8]:
```
# Import pandas and read in csv file as a dataframe
import pandas as pd
# Change the path below to your data directory
# If you are in a locale (e.g., Europe) that uses ',' as the decimal separator
# change the '.' to ','
df = pd.read_csv('../test_dataset/rover_img/robot_log.csv', delimiter=',', decimal='.')
csv_img_list = df["Path"].tolist() # Create list of image pathnames
# Read in ground truth map and create a 3-channel image with it
ground_truth = mpimg.imread('../calibration_images/map_bw.png')
ground_truth_3d = np.dstack((ground_truth*0, ground_truth*255, ground_truth*0)).astype(np.float)

# Creating a class to be the data container
# Will read in saved data from csv file and populate this object
# Worldmap is instantiated as 200 x 200 grids corresponding
# to a 200m x 200m space (same size as the ground truth map: 200 x 200 pixels)
# This encompasses the full range of output position values in x and y from the sim
class Databucket():
    def __init__(self):
        self.images = csv_img_list
        self.xpos = df["X_Position"].values
        self.ypos = df["Y_Position"].values
        self.yaw = df["Yaw"].values
        self.count = -1 # This will be a running index, setting to -1 is a hack
                        # because moviepy (below) seems to run one extra iteration
        self.worldmap = np.zeros((200, 200, 3)).astype(np.float)
        self.ground_truth = ground_truth_3d # Ground truth worldmap

# Instantiate a Databucket().. this will be a global variable/object
# that you can refer to in the process_image() function below
data = Databucket()
```

In supporting_function. py –

Line 21-

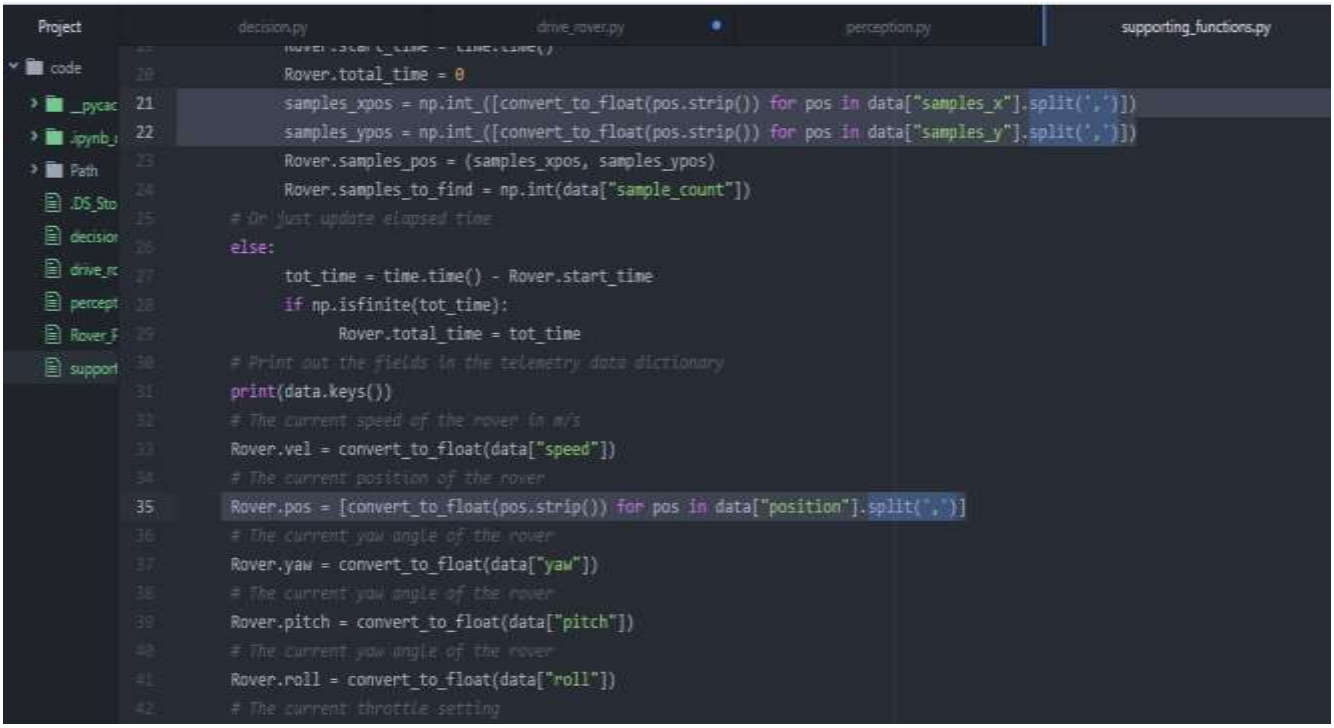
```
samples_xpos = np.int_([convert_to_float(pos.strip()) for pos in data["samples_x"].split(',')])
```

Line 22-

```
samples_ypos = np.int_([convert_to_float(pos.strip()) for pos in data["samples_y"].split(',')])
```

Line 35-

```
Rover.pos = [convert_to_float(pos.strip()) for pos in data["position"].split(',')]
```



Please Note that if you are on a mac ,you need to change the “ **,** ” to “ **;** ” at all places mentioned above to get the code to work on your machine.