UDACITY ROVER PROJECT

This project aims at helping the user understand and provide a hand-on experience on the basics of Robotics. This write-up is a humble attempt at explaining what was taught and what I implemented based on it. The RoverSim is a simulation software that is replicated after the Mars Rover.

AIM OF THE PROJECT: The Rover should cover maximum area, while trying to collect rock samples, typically colored in golden yellow.

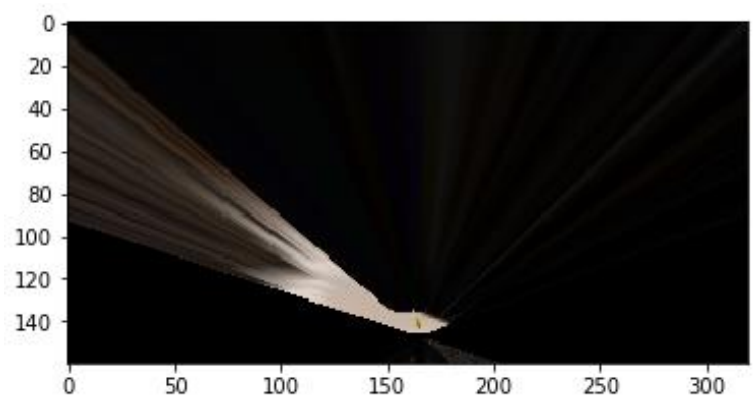
EXPECTATION: The Rover will send in images that need to be parsed for information – navigable terrain, rock samples and obstacles. The Rover will need to travel the terrain, avoid obstacles and collect rock samples.

<table>
<tr><th>CRITERIA</th><th>MEETS SPECIFICATIONS</th></tr>
<tr><td>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.</td><td>Another color threshold function was added to the existing one. The new one had upper and lower limits added since the rock samples and the obstacles had very specific color threshold values. The existing color threshold function was used only for detecting navigable terrain.

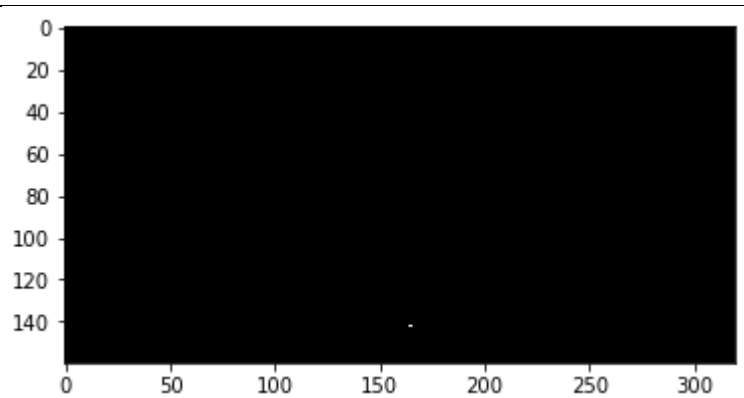
```
def color_range_thresh(img, rgb_thresh=(160, 160, 160)):
    # Create an array of zeros same xy size as img, but single channel
    color_select = np.zeros_like(img[:,:,0])
    # Require that each pixel be above all three threshold values in RGB
    # above_thresh will now contain a boolean array with "True"
    # where threshold was met
    limit = 20
    rgb_red_low = rgb_thresh[0] - limit
    rgb_red_hi = rgb_thresh[0] + limit
    rgb_blu_low = rgb_thresh[1] - limit
    rgb_blu_hi = rgb_thresh[1] + limit
    rgb_grn_low = rgb_thresh[2] - limit
    rgb_grn_hi = rgb_thresh[2] + limit
    above_thresh = (img[:,:,0] > rgb_red_low) \
                & (img[:,:,0] < rgb_red_hi) \
                & (img[:,:,1] > rgb_blu_low) \
                & (img[:,:,1] < rgb_blu_hi) \
                & (img[:,:,2] > rgb_grn_low) \
                & (img[:,:,2] < rgb_grn_hi)
    # 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
```
</td></tr>
<tr><td>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 moviepyfunctions provided to create video output of your result.</td><td>The process_image() function was populated to do the following:
1. Define the source and destination coordinates for warping the Rover image

```
source = np.float32([[125,95], [205,95], [310,140], [10,140]])
destination = np.float32([[160,140], [170,140], [170,145], [160,145]])
```

2. Apply color thresholds for navigable terrain, rock samples and obstacles to the warped image and receive a binary channel for each



</td></tr>
</table>

	 3. Cover the pixels into Rover centric coordinates 4. Subsequently convert the Rover centric coordinated into world map coordinates 5. Based on the world map coordinates, the worldmap image is updated
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.	Perception was modified following the test notebook logic. Apart from that, the following additional logic was put into place. 1. World map coordinates were converted to polar coordinates 2. Logic to convert them into distances and angles was added Decision logic was as follows: 1. If there are enough angles to navigate with appropriate length, and no rock sample is in sight, the Rover moves based on the logic that was pre-loaded – move until there is navigable terrain available. 2. If a rock sample is spotted in the Rover vision and the near_sample flag is not set, then the angle and distance from the same, calculated in perception_step is used to navigate the Rover to the sample. 3. If the rock sample is spotted and the near_sample flag is set, then the pickup function is called for the Rover to pick up the rock sample. HIGHLIGHTS OF DECISION_STEP 1. Instead of navigating the mean angle, decision_step is modified to take the angle that is at 2/3rds of the range so that the Rover follows the wall hugger method. This way, it is ensured that the Rover will cover maximum distance 2. Due to the above reason, the angle to be turned when the Rover faces an obstacle is maintained at -15 radians.
Launching in autonomous mode your rover can navigate and map autonomously. Explain your results and how you might improve them in your writeup.	The following observations were made when the Rover was run in Autonomous mode: 1. When the Rover is launched, it seems to 'dawdle' around at the beginning since it is trying to find an obstacle and then trace its boundary. 2. Rock samples were mapped correctly once they came into the Rover's vision. IMPROVEMENTS 1. First and the most important is to circumvent the 'bug' that causes the Rover to freeze after it picks up the first rock sample 2. Another point is to try to avoid that 'wavy movement' when the Rover moves from side to side when travelling the terrain. 3. Even after explicit coding to turn the Rover by 15 radians if there is no terrain available, the Rover sometimes seems to bump into obstacles. ROVER SPECIFICATIONS IN WHICH THE PROJECT WAS CARRIED OUT 1. Resolution: 320 X 240 2. Graphics Quality: Good