Search and Sample Return Project

Writeup

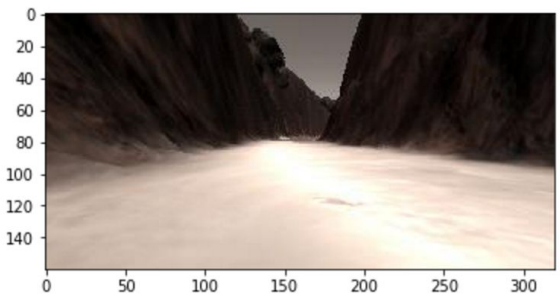
Notebook Analysis

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.

I ran the functions with the original test data and with my own test data, which is in the folder called my my_dataset. I used the same threshold used in the lectures to separate navigable terrain from obstacles. For the rocks I created a new function called find_rocks, and used the parameters shown in the video tutorial (I was stuck choosing the correct threshold). This is an image from my dataset.

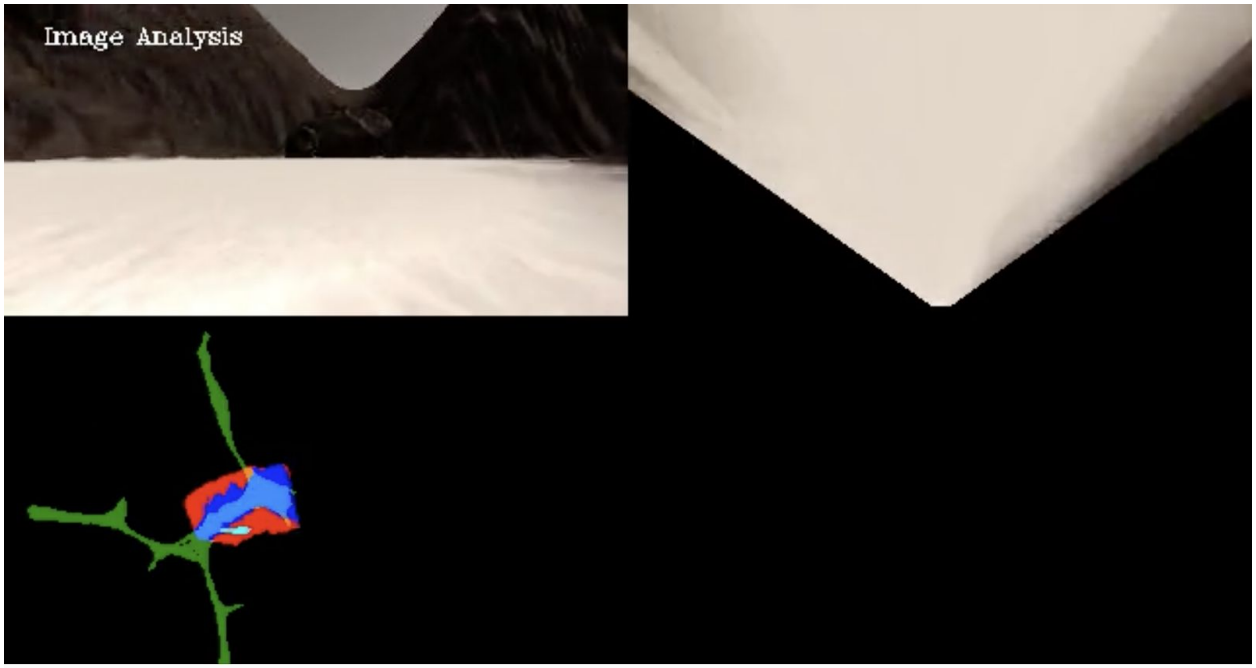
```
In [12]:
path = '../my_dataset/IMG/*'
img_list = glob.glob(path)
# Grab a random image and display it
idx = np.random.randint(0, len(img_list)-1)
image = mpimg.imread(img_list[idx])
plt.imshow(image)
```

```
Out[12]: <matplotlib.image.AxesImage at 0x11945e7f0>
```



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.

To calculate the perspect_transform functions I used the same image that was originally provided. Every other function on the notebook was written the same as in the lectures. I ran process_image() succesfully on my own dataset. Here's a picture of the video (which is also attached):



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.

Perception script:
I did several modifications in an attempt to make my rover choose new paths over already mapped ones.

1. I added the find_rocks() function. This is the same function I have on my Notebook.

2. I added 3 functions that reversed the process of transforming rover view pixels into a world map mapping. 2 of them were actually taken them from the forums (I was already working on them but I found them on the forums so I took them to accelerate the process). This functions allowed me to use the blue pixels that were mapped and transform them into a rover_coords view of these mapped and unmapped pixels.

3. Most of the changes were done in the perception_step() function:

- The first version only contained the normal things to make the function work: a perspective_transform(); a color_thresh() applied to obstacles, navigable terrain and rocks; and update of the Rover.vision_image; a conversion of the 3 maps (obs, nav and rocks) to rover_coords; locating and positioning this mapping of rover coords into the world map and updating it; using the rover coords to get angles to define the steering angle of the rover.
- For the second version I added functionality to the code. First, I only allowed the map to be updated if the pitch and roll of the rover were under a certain threshold (line 166 and 175). This allowed for a more accurate mapping (as suggested in the classes). Also, In line 186 I added an if statement which changed the nav_angles handed down to the rover if there was any pickable rock in sight. What the statement does is to use the rock pixels as the info to update the mean angle of steering instead of the navigable terrain. Basically, once it sees a rock it kind of forgets the whole world around it and it only focuses on the rock.
- After this, I wanted to address the problem of the rover visiting the same places over and over again. For the third version I added a weighting system using the 3 reversing functions I mentioned before.
    - First, when the Rover updated the mapping, I only added 1 to the nav_terrain channel each time it mapped (instead of immediately setting up the value to 255). This allowed me to have a “threshold” of the pixels that had been mapped several times through time (line 176).
    - Then, according to this threshold (which I ended up setting to 250) the code would create a second map of the whole area setting the pixels over the threshold to 1 and everything else to 0 (line 215, 216).
    - I would use that entire map of 1’s and 0’s and reverse the process to only grab the part of the whole map corresponding to the rover coords map in that moment (line 224).
    - Then, I would separate the lists of pixels into the ones with ones and the ones with zeros (so I would separate the pixels mapped already and the ones not mapped) (lines 226-229).
    - Finally, I compare these lists to the original pixels the rover is using after the color_thresh() and rover_coords() step. The pixel values of the rover vision in that moment that had also been mapped before would be diminished (divided by 0.5) and the ones not mapped would be incremented in value (multiplied by 2) line (232-235).
    - This would bias the mean of all the nav_angles towards the pixels not mapped before (and therefore, most probably the places not visited by the rover).
    - So in summary, I managed to alter the mean angle of the nav_angles by applying a weighting system to them, by making already mapped sections weigh less and unmapped sections weight more.

Even though this worked and allowed the rover to choose new areas over visited ones, some problems arose.

- Problem1: It happened quite often that the rover would get stuck trying to go forward while being right in front of a rock. This happened because sometimes the left side of the image would have many unmapped pixels and the right side would have less unmapped pixels, but more pixels in general than the left side. This would result in the mean angle being biased towards the center (because I'd have a medium number of unmapped heavy-weighted pixels (left) vs a big number of mapped light-weighted pixels (right)).
- Solution: I Duplicated the obstacle map and sliced the center part in 3. I then compared in real time the number of obstacle pixels of these 3 slices (left-center, center and right-center) (lines 142-149). If the center slice has more obstacle pixels than both the left and right slices, it very probably means there's a rock, so I'd make the rover steer to the left. You can see the workaround in the console when the rover is heading straight to a rock. It will print "Rock ahead!" and then steer (line 195).

Decision script:

- Problem2: This problem was addressed in the decision script. Because of some angles being underweighted, the rover would sometimes get a very weak mean angle to make turns, resulting in the rover sometimes getting stuck to the walls when trying to turn and accelerate at the same time.
- Solution: I started making "memory" lists. The code records the position of the rover every once in awhile (I started with 4 seconds but modified it to 116 frames), keeping not more than 2 recordings in the list at all times. Every frame, the code compares these positions and makes a decision. If the positions are different by a +- 0.4 value, it means the rover is moving as it should. If the positions are the similar by a 0.4 difference, it probably means the rover is stuck, so it makes the rover steer 30 degrees (this comparison between degrees is done the same way, with lists storing values and comparing them through time). Once the rover steered 30 degrees, it erases all info of the lists so it can start accelerating again. If it's still stuck, it'll repeat the process once the position list fills up again (lines 26-55).

Apart from this, I made rather small modifications to the decision script.

- I added conditions to the if statements so the rover could pick up the rocks. Basically, if the rover is near a rock, I would make it fully stop, and not steer or move forward.
- I added a small system to make the rover turn the opposite way as its last steering angle before it stopped. I realised most times the rover is moving forward and would move to either side if it found a rock. I wanted the rover to continue the path it was taking so the best thing would be for it to steer the opposite way (if it turned left to grab a rock, the path it was taking would now be to the right) (lines 17-23).

Drive_rover script:

- I modified values of some of the variables. I considerably incremented the Rover.brake setting, to make it stop on time to grab the rocks (because of the way I implemented it, if the rover was going to fast and stopping slowly, it would go past the rock, and the forget about it and continue it's path). I also very slightly incremented its speed.

- On the perception script I modified the stop forward and move forward values. When the rover finds a rock, this values are set to 0 (the stopping of the rover is handled by the decision script if the rover is near a sample). When the rover would return to navigation mode, the values were set to 150 and 1000. The reason of the increment in the go_forward value to a 1000 was to compensate slightly for the second problem I mentioned. Since the steering angles were sometimes weak, I figured needing more angles before moving forward would allow for more steering before acceleration.

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

I feel the system override when the rover finds a rock works pretty well. For the most part, every time it finds a rock it is able to correctly catch it. The drawbacks are that sometimes the rover doesn't respond very well. If it can't slow down on time and loses the rock out of sight, it forgets there was a rock and just keeps moving. Also because of the way it approaches the rocks' locations, the rover is prone to getting stuck all the time.

Regarding the weighting system, I feel I did an ok job in making the rover choose new paths over old ones. Sometimes it really surprised me as it would hesitate little in choosing an unmapped path over a visited one. The time it now takes to go over the whole map is considerable smaller than the time it took before the weighting system was implemented. Also, before this system, the rover would sometimes get stuck in loops, going around in circles. The weighting system also solved this, which I think was one of the best things about implementing it. However, some serious problems arose because of these systems. First of all, the rover was prone to get stuck quite more often than before (because of all the tempering with the mean angles). The solutions I provided for these problems were enough to make the rover be able to keep exploring, but I think they are more like patch solutions than good coding. I would feel like patching a leak just for 3 more leaks to come out in its place. Having had more time and experience, I feel the solutions I provided could be greatly improved (or completely reworked from scratch).

This project led me to understand how hard robotics can be. There are too many variables in play. Besides this is just a simulation. The robot in here doesn't break or doesn't get truly stuck under a rock. If this were to be a real robot, I would have broken it a thousand times, and that's certainly not good.

I really loved working on this! I'm very very new to robotics (first project ever involving robotics!), so if I could get some guidance to know if my approaches were on the right track or completely off, that would mean the world to me.