Project: Search and Sample Return

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.

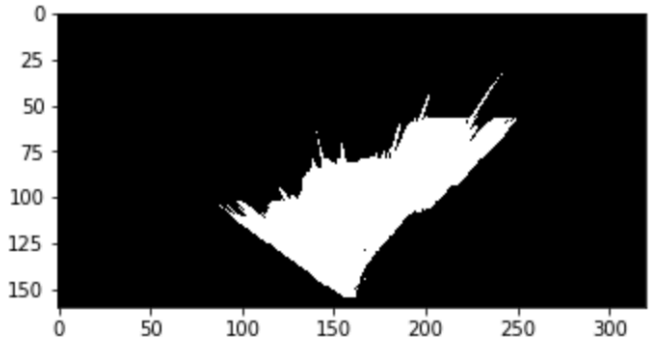
You're reading it!

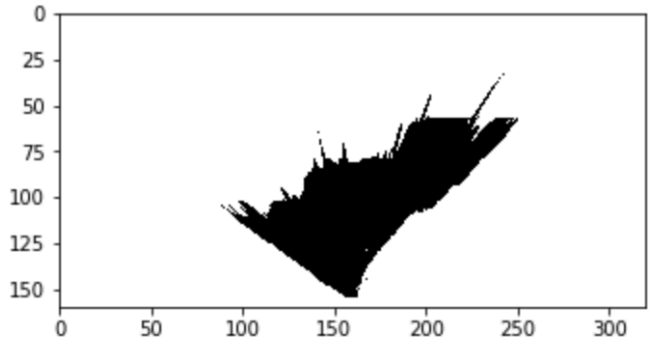
Notebook Analysis

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.

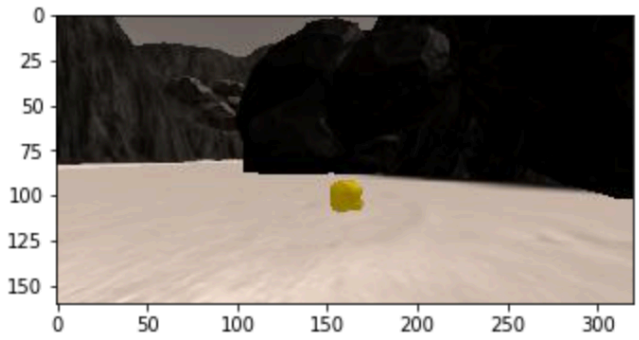
Here is an example of how to include an image in your writeup.

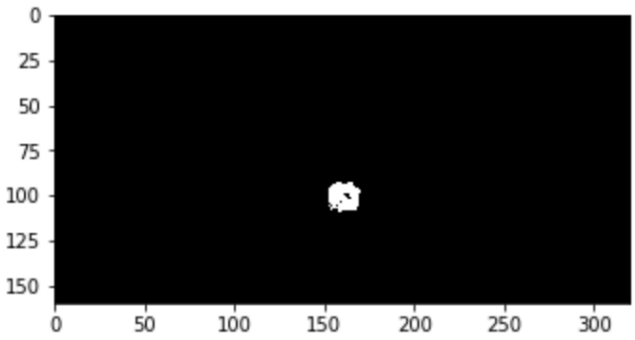
I started with testing the provided code in `color_thresh` to find the navigable terrain and obstacles by inverting the navigable area





Next I wrote a similar thresholding function named `rock_thres` to find rocks on the image. I used a filter where R>110, G>110 and B<50 which appears to be work fine:





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.

This is the main funciton which does all the job. I just followed the provided instructions in the comments to combine the previously defined and tested functions to do all the work.

First, it was required to do a perspective transform to see the map from top down, then the color threshold function is used to distinguish between navigable world and obstacles, as well as the function I crafted is used to find rock samples. After that, I have 3 different sets of worlds from top down view and transform the views to rover views using the `rover_coords` function. Finally, I switch the rover coordinates to world coordinates in order to update the world map.

See the provided video for the result. The video was generated via provided function which applyed `process_image` to each frame.

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.

For `perception_step`, it is just a copy of `process_image` with modifications to support Rover object as the input. To improve the fidelity the reviewer recommended to avoid updating Rover's world map when pitch and roll angels are far from 0 which completely makes sense as perpective transform is inneficient in such case.

The `decision_stop` is also the provided one with two differences. First is addition if statements to handle rock disovery, navigation to rock and its pickup (lines 16-34 in `decision.py`). The second is to make the rover pull back a bit (`Rover.throttle = -0.5`) when it appears to be stuck in addition to turning.

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

Note: running the simulator with different choices of resolution and graphics quality may produce different results, particularly on different machines! Make a note of your simulator settings (resolution and graphics quality set on launch) and frames per second (FPS output to terminal by `drive_rover.py`) in your writeup when you submit the project so your reviewer can reproduce your results.

I used 1024x768 resolution with Good graphics quality. The rover is able to map 40% of the environment with 60% fidelity in the most of the casess while picking up the rocks. Some times it gets stuck and requires a manual help to unstuck. This would be the first thing I would try to improve for example via more sophisticated way to discover this situation and get out of it.