Project: Search and Sample Return

The goals / steps of this project are the following:

Training / Calibration

- Download the simulator and take data in "Training Mode"
- Test out the functions in the Jupyter Notebook provided
- Add functions to detect obstacles and samples of interest (golden rocks)
- Fill in the `process_image()` function with the appropriate image processing steps (perspective transform, color threshold etc.) to get from raw images to a map. The `output_image` you create in this step should demonstrate that your mapping pipeline works.
- Use `moviepy` to process the images in your saved dataset with the `process_image()` function. Include the video you produce as part of your submission.

Autonomous Navigation / Mapping

- Fill in the `perception_step()` function within the `perception.py` script with the appropriate image processing functions to create a map and update `Rover()` data (similar to what you did with `process_image()` in the notebook).
- Fill in the `decision_step()` function within the `decision.py` script with conditional statements that take into consideration the outputs of the `perception_step()` in deciding how to issue throttle, brake and steering commands.
- Iterate on your perception and decision function until your rover does a reasonable (need to define metric) job of navigating and mapping.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

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.

In the notebook the goal is to prepare the pipeline for perception part of the robot. It consists of couple of steps.

- Transformation of the picture
- Segmenting by thresholding
- Detecting rock samples
- Converting to different coordinates system
- updating map image

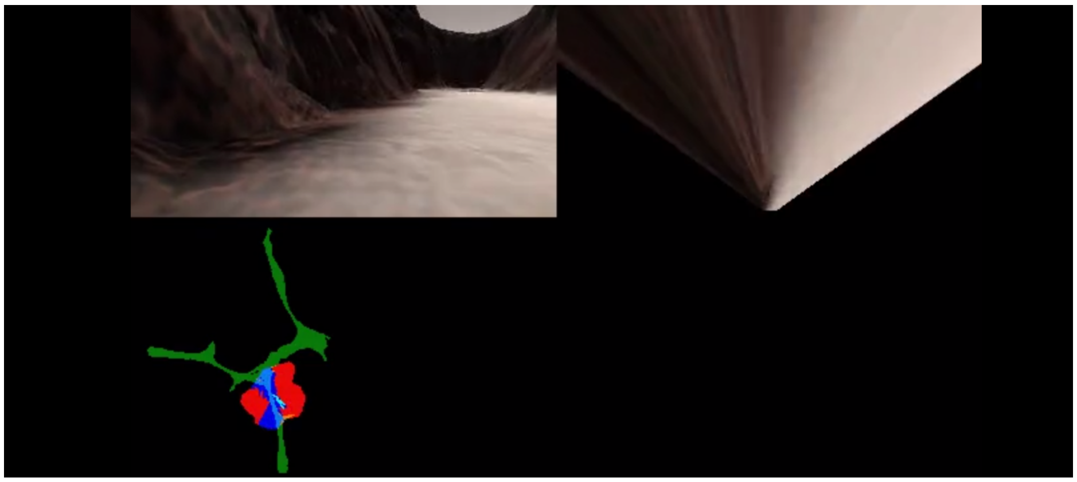
Most of the notebook is already prepared. I made only couple of changes.

- When working on the example I could not figure out how to capture rocks in RGB so I used HSV conversion in opencv. Later I noticed that the walthrough uses a little bit more complex filter.
- great idea with the filter in the walkthrough. I did not realize that the mask can be obtained so easily via transformation of ones matrix. I did not even realized that it is a problem with not suing any kind of mask. I reused that idea in the notebook.
- In the video I tried to demonstrate that the transformation provides good results only when the vehicle is leveled. Since there is some rudimentary telemetry simulation during accelaration there is some camera movement and whenever you drive into the sloap a little bit possibility to accumulate the error is significant. I am trying to remove some of that later in the scripts

The picture below demonstrates that effect

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.

Here we just applied things from above, captured images and stitched together a video. Made it not too long so it can fit into the repo easily.



Video is here

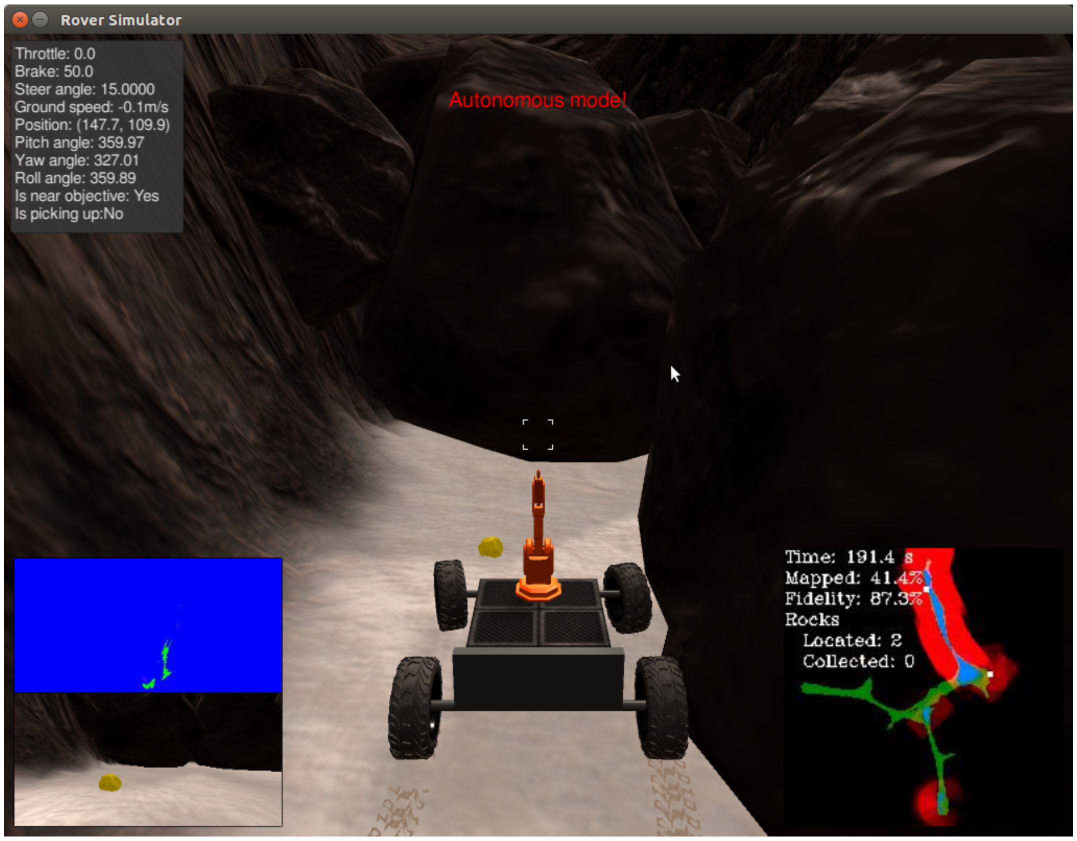
Autonomous Navigation and Mapping

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.

I enhanced the decision_step in the following way

1. I tried to make the rover go a little bit faster. I quickly realized that just speeding up is not good because rover is not able to actually make the turns an crashes quickly. I tried to implement simple heuristic, that tells if there is a obstacle ahead. Rover can move in 3 speeds. Subjectively it seems to help with the speed.
2. I added a simple algorithm to collect samples. Collection itself is not difficult so the scripts alread provide the code. I just needed to uncomment. A little more code had to be added to actually close in to the sample once detected.
    - I enhanced the state of the robot by 2 variables. Is a sample detected? And what is the angle to that sample measured.
    - If detected rover switches to the approach mode and it follows a simple alogrithm.
        - If we are close stop and collect.
        - If we are not close go slowly and turn so you minimize the angle error of the sample

When happening it is a bit choppy but works relatively well.

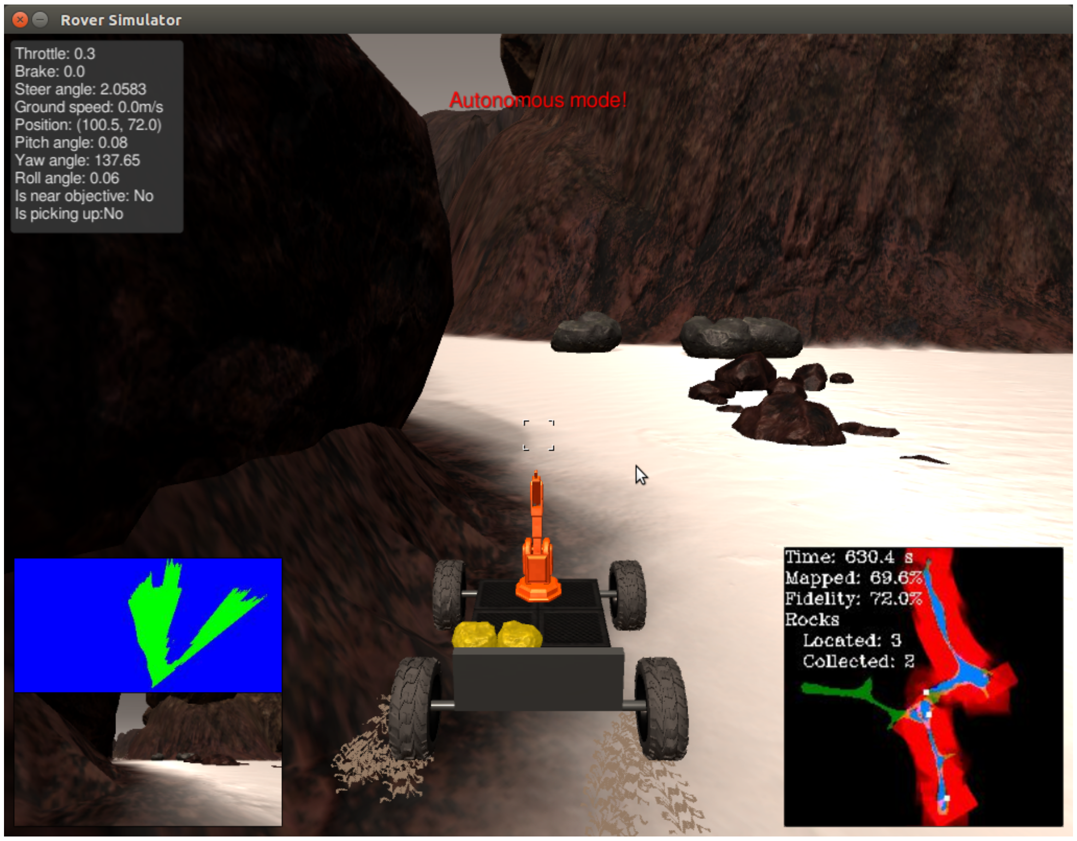

Rover Simulator

Throttle: 0.0
Brake: 50.0
Steer angle: 15.0000
Ground speed: -0.1m/s
Position: (147.7, 109.9)
Pitch angle: 359.97
Yaw angle: 327.01
Roll angle: 359.89
Is near objective: Yes
Is picking up:No

Autonomous mode!

Time: 191.4 s
Mapped: 41.4%
Fidelity: 87.3%
Rocks
Located: 2
Collected: 0

1. I tried to implement a simple incentive for the robot to favor one side over the other when steering. The goal was to achieve a wall crawling behavior so the robot has a bigger chance to traverse more of the map. The algorithm is implemented by not using mean but



Throttle: 0.3
Brake: 0.0
Steer angle: 2.0583
Ground speed: 0.0m/s
Position: (100.5, 72.0)
Pitch angle: 0.08
Yaw angle: 137.65
Roll angle: 0.06
Is near objective: No
Is picking up:No

Time: 630.4 s
Mapped: 69.6%
Fidelity: 72.0%
Rocks
  Located: 3
  Collected: 2

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.

Simulator was run in 1024x768, Good graphics quality.

This is a deceivingly complex exercise. As can be seen in video and the walkthrough tutorial kinda works but it can be tripped up by so many real world robotics problems + there are couple of problems added by the simulation.

Robot struggles in certain positions or parts of the map and occasionally has to be nudged.

How I would improve with further work

Moving around

There is a class of problems where the robot is stuck and makes no progression but the camera sees nothing wrong.

- Robot collides with the environment in a weird way and sees through the objects
- Robot falls through some geometry. While annoying this simulates possible situation in real life where movement is stochastic. Our situation is simplified by the fact we always know where we are in the map.
- Sometimes the robot has problems to get over the terrain even if it seems passable. This happens at the edges more and is exacerbated by the fact that I accentuated driving closer to the wall.

I am not sure how these problems are solved in real life. What comes to mind is some heuristic "when stuck" try to back out. But obviously this does not have to work every time.

Problem with the path planning

The initial naive implementation by averaging the pixels of passable terrain have a lot of problem with obstacles dead ahead.

There is also another fundamental problem with this which I am not sure can be solved without other sensors. I tried to implement the wall crawler by adding bias to the angle but this ends up crawling fairly close to the wall. It would be great to find a way how to keep more distance from the wall. I feel like the angle is kinda misleading and was thinking about switching to some cost based planning algorithm. But there are additional problems suddenly you have to pick where to go etc.

Problem during collecting samples

The algorithm for collecting samples is fairly choppy but seems to work reasonable well. There are two problems during the collection.

- It expects that when you see the sample the path is clear which is not always the case.
- If for some reason rover does lose sight of the sample (speeds over it) it dos not know about it even if it is marked on the map.

Again some better planner that would take the real map into account would be a good solution here I think.