Writeup by Nakkeeran

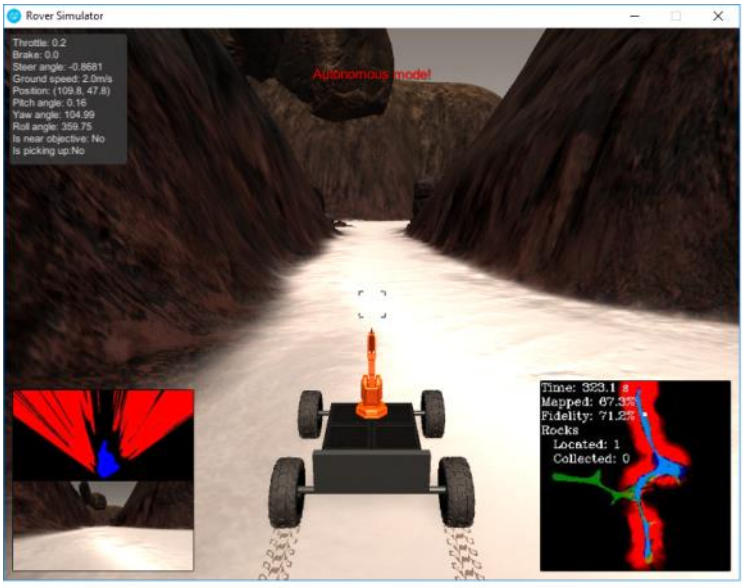
Wednesday, August 30, 2017 9:50 AM

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.
[//]: # (Image References)
[image1]: ./misc/rover_image.jpg
[image2]: ./calibration_images/example_grid1.jpg
[image3]: ./calibration_images/example_rock1.jpg

[Rubric](https://review.udacity.com/#!/rubrics/916/view) 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.

[NK]

Started with understanding the Simulator, quiet an interesting stuff. I did play around with different navigation buttons and understand how rover moves and pick up the Yellow Rock.
It helped me to understand the different parameters including Pitch, yaw, roll. Finally, did a quick recording of my own dataset. In that folder you'll find a robo_log.csv file along with the output dataset Images recorded.

Steps followed:

1. Launch the simulator on windows laptop and choose "Training Mode" then hit "r".
2. Navigate to a directory to store dataset images and then drive around and find any yellow rock in the quickest possible time.
3. Hit "r" again to stop data collection.

2. 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.

[NK]

In this task, i am able to execute and complete all code snippets provided in Rover_Project_Test_Notebook.ipynb as is.
Run the cells in the notebook from top to bottom to see the various data analysis steps using jupyter notebook.
This helped me understand, how the perspective transform works from recorded dataset of my own.
The last two cells in the notebook are for running the analysis using those test images to create a map of the simulator environment
and write the output to a video called nk-test_mapping.mp4 into the output folder.
This should give you an idea of how to go about modifying the process_image() function which actually does the perspective transformation.

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.

[NK]

For Autonomous mode, we need perspective transformation to convert the Rover's vision and position to a top-down view and converted into a warped image.
Then will get the Rover coordinates and get the distance and available angles for the navigable terrain pixels.
But again we need process the image finding proper route based on color threshold of rock, general obstacles, ground plane with corresponding RGB (180,180,180) and create a binary image for the obstacle.
Then Updated the Rover.worldmap to display the obstacles as red, samples in green, and terrain in blue.
At end updated the Rover.nav_dist and Rover.nav_angles, which are received from to_polar_coords function.

note:

For every frame the Rover sees, that image is processed, first to see the there is a navigable terrain, by doing a color thresh and get the binary image, which shows whether it has a terrain to move forward.

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!

[NK]
First I checked whether there is any visible terrain, that i will get from the Rover.nav_angles. if it is not null , then i will check
whether my Rover is moving in the Forward or Stopped mode or Stuck on Rocks from Rover.mode status.
When my Rover is in "Forward", I will check, whether I have enough of terrain path to navigate by checking the number of Rover.nav_angles greater than the Rover.stop_forward limit.
If I more nav_anlges, then I will check the throttle, if it is less than the max_vel, then give an acceleration by increasing the throttle to throttle_set, else decrease the throttle. Then I will release the brakes and turn in the direction of more navigable terrain by taking the mean of nav_angles and check the mean in the limit of -15(right turn) & +15 (Left Turn).
When I don't have enough of navigable terrain, I make my rover to stop, by assigning the Rover.mode=stop, apply the brakes, stop the acceleration and no steering.
When the Rover is in stop mode, if Rover has enough of navigable terrain by checking the number of Rover.nav_angles greater than Rover.go_forward, if so, increase throttle, release brake, make turn and move forward.

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

[NK]
Used following screen resolutions to run the simulator:
Screen Size - 800 x 600
Graphics quality - Good.
I see the rover navigates with more than 71% of Fidelity and for more than 60% of navigation. It is showing the samples on the world map whenever the rover is passing them.

Plan to improve are following
1.It goes into circular infinite loop particular at dead-ends, junctions (3 way path).
2.Enhance the code to pick up all the samples and conclude at the starting point.
3.Sometimes near some big rock, it is getting stuck, I need to handle those obstacle conditions properly to get out the rocks.
4. Increase the speed of navigation and also pick up the rock.

From <http://localhost:8891/files/Documents/GitHub/RoboND-Rover-Project/writeup_template.md>