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.

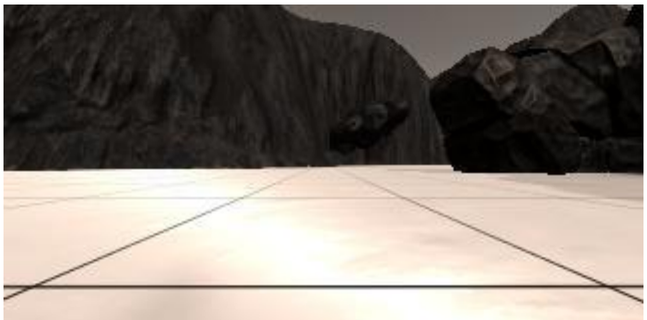
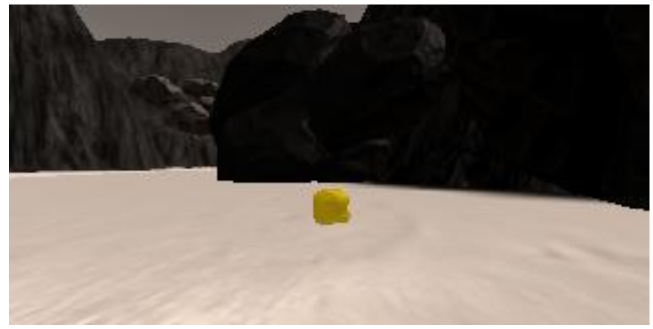
Autonomous Navigation / Mapping

* 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.

Pipeline

1. **Color Thresholding**

I used a combination of color thresholds to generate a binary image (thresholding steps at lines #5 through #45 in `perception.py`).

The way to select obstacles, rocks and navigable area are as follows:

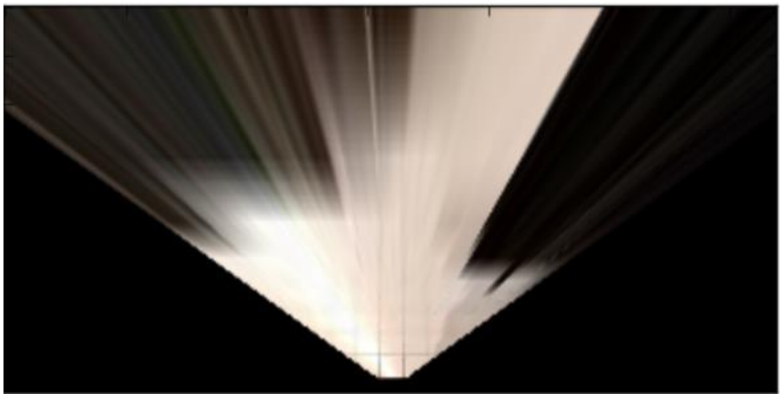
* Set color_threshold to be (160, 160, 160) as default which is the navigable area.

* Set obstacle_threshold to be less than (160,160,160) which means the whole picture is obstacle excluding the navigable area and the black area.

* Set lower rock_threshold to be (100,100,0) and higher threshold to be(160,160,40).

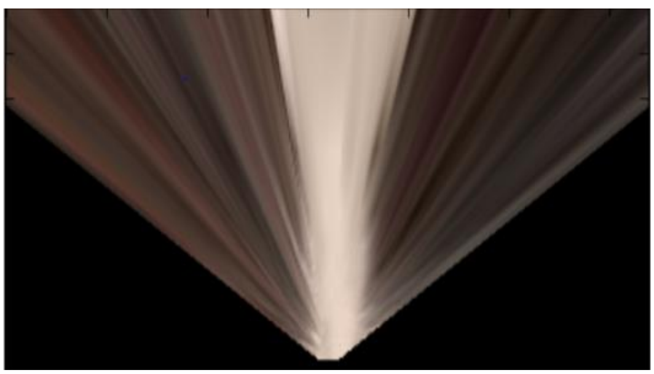

2. **Perspective Transform**

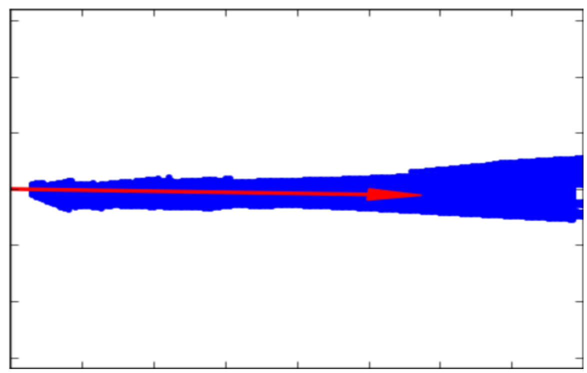
Initially I defined source and destination points for perspective transform. While declaring the points I set a bottom offset to account for the fact that the bottom of the image. I also considered the position of the rover a bit in front of it in offset.



3. Co-ordinate Transformation

In this step, the rover centric image is converted into world map centric. Rover coordinates are first converted into polar coordinates and then pix_to_world function is used to convert them into world map centric.(steps at lines #54 through #85 in perception.py)





Discussion

1. While driving in autonomous mode, rover used to get stuck at the rocks in the path. I added a logic for the rover to move backwards by setting throttle value negative and then turning in the either of the directions.
   There could be a more clever way for the rover to decide in which direction it should go.
2. I have added a counter in the decision.py, so that if the rover gets stuck at a position for more than a particular time, it will move backwards and take a turn.
3. I would like to update decision.py further so as to make the autonomous driving more robust.
4. I also wish to implement rock picking mechanism.