## 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 job of navigating and mapping with at least 40% of the environment at 60% fidelity and locate at least one of the rock samples.

[//]: # (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.

### Writeup / README

#### 1. Writeup includes all the rubric points and how I addressed each one.

### 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.

The recorded data are saved in folder: \automode_data\IMG.

In order to threshold the navigable terrain I used color_tresh() function, as provided in the lesson.

Then I added the color_thresh_obstacle() function to color places where are obstacles using the same script as in color_tresh() function that returns places with below RGB color threshold. To identify the rock samples color_rock_thresh()function is added that selects colors between (100, 100, 0) and (200, 200, 50) RGB threshold.

I added scripts from the lessons in rotate_px() and translate_pix()functions

#### 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.

The recorded video is output\mapping.mp4.

The `process_image()` function is modified to include perspect_transform(),color_thresh(,)rover_coords()and pix_to_world() functions as required. To calculate the rover-centric coordinates of the images the binary thresholded and perspective warped images returned from `color_thresh(). Then the rover-centric coordinates are used to calculate the world map image. in order to be seen in the video the binary thresholded image is multiplied by 255.

### 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_step()` is basically replication of the `process_image()` function in Jupyter notebook except that the `RoverState` class is used instead of `Databucket`.

`decision_step()`A conditional block checking if the rover stuck is added. If stuck_counter is above 3 then the rover drives backwards and steer to another direction.

#### 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.

The setting I used for running the simulator are a)screen resolution is 1280 x 720, b) graphics quality is simple and c)frames per second is 25fps

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.

There were several improvements I wanted to make:

1. Prevent rover from getting stuck in circles or ignoring a narrow passageway in favor of a larger but already mapped passageway.
2. Have the rover pick up samples having the rover navigate to the rock samples and pick them up.
3. Adjust speed and brake that the rover does not collide with obstacles and slow speed toward found rock sample.
4. Ensure full mapping of the terrain.
5. Return to the starting position.