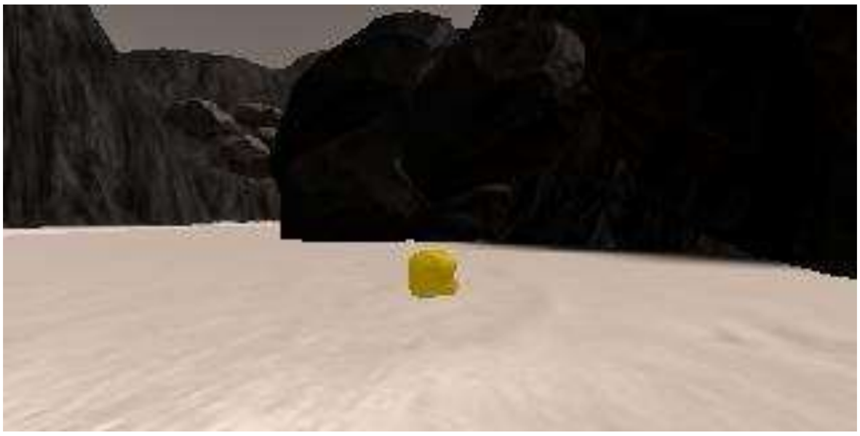
Notebook Analysis

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 Color Thresholding part, I modify the function *color_thresh()* where I set both lower color threshold and higher color threshold for identifing obstacles/rocks better. Below shows the rock example obtained by using the new *color_thresh()* function.



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.

The example code in *process_image()* alread apply a perspective tranform on the input image and get a warped one. For the warped image, I do the color threshold and then map it from rover-centric coordinate on the world coordinate. I define a function *select_object()* to do the color threshold and coordinate conversion. This function *select_object()* is called three times with different color threshold parameters, and thus the obstacles, rocks and navagable terrain can be identified, respectively. Then I create a new image worldmap and pixels corresponding to three objects are maped to different color channels. I run *process_image()* on my own test data, and the results are displayed in a video test_mapping.mp4.

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.

In *perception_step()* function, I add below parts:

1) Define source and destination points, and apply perspective transform. It is for displaying what the camera sees in the worldmap correctly. It is top-down view and helpful to make decision.

2) Apply color threshold to identify navigable terrain/obstacles/rock samples. I use different color threshold for the three objects.

3) Update Rover.vision_image. I map pixels correspoind to the three objects on different channels in Rover.vision_image. It is better to display, and also convenient for following analysis.

4) Convert map image pixel values to rover-centric coordinates, and then convert rover-centric pixel values to world coordinates. We do this for overlaping it with the worldmap.

5) Convert rover-centric pixel positions to polar coordinates. In the polar coordinates, the angles can tell us where we could go and where the rock is.

In *decision_step ()* I make no changes. I tried to do some modification, but always make it worse. I only increase self.stop_forward to 200. It is threshold to initiate stopping defined in driver_rover.py. I hope the robot can stop earlier, or it may hit the obstacle and is stuck.

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

Simulator settings:

Machine: macOS Sierra 10.12

Resolution: 1024x768

Graphics quality: good

PFS: 22

It seems to go well in most time. It may be a little slower. I try to increase the maximum velocity to enhance the efficiency, however, the rover in this case often hit the obstacle and stuck. I also try to increase the brake and the threshold to initiate stopping, but it seems not help. I need spend more time to test. Maybe I need to consider the distance from obstacle and the current velocity, and estimate a suitable brake and time to stop. The current perspective transform is only valid when roll and pitch angles are near zeros. In the future I need to consider the suitable perpective transform for large roll and pitch angles.