

Project:

Search and Sample Return

Project 1

Christopher J. Rukas | Robotics Software Engineer | October 19 2018

# Project Description

Project: Search and Sample Return is oriented as an introduction into the Anaconda/Python development environment, the open source computer vision tool OpenCV, and machine logic with a given set of inputs. This entails navigating a small rover through a canyon type terrain. This document will review the lessons learned.

# Notebook Analysis

The Jupyter notebook provides an easy method of sharing python code and documentation in a WYSIWYG environment.

### Color Thresholding

Our simulated environment uses relatively few colors and allows for easy separation of data based off of the color information. An example threshold function is provided that identifies the ground due to its brightness. On an RGB scale of [0,0,0] to [255,255,255] where 0 is black and 255 is white, the ground is identified as pixels with a color greater than [160, 160, 160].

Several attempts were made with thresholding by looking above, below, and in between various zones.

To identify obstacles the inverse of the threshold function was applied. In Process\_Image it was assumed any area not navigable was an obstacles, but within the Color Thresholding section separate functions were developed for each task.

The function color\_thresh identified the ground, color\_thresh\_above also identified the ground, color\_thresh\_below identified obstacles, color\_thresh\_middle can find edges, and color\_find\_rock identifies rocks within the simulated environment.

color\_find\_rock tries to identify yellow. Since yellow is defined by rgb(255,255,0) we try to identify pixels high in reds, greens, and no blue. This threshold was set at 110, 110, and 50.

### Process Image

process\_image is a user defined function which processes the camera mounted information, identifies the locations of navigable terrain and objects in rover coordinates, and overlays the same information into a global perspective.

#### Perspective Transform

Initially, a perspective transformation is applied to the camera image. This perspective transform is identified by using a grid of a known shape and size, and viewing it from the perspective of the rover. The initial size and shape are passed to OpenCV to perform the transformation.

#### Rover coordinates to world coordinates

The transformed image provides a “top down” view of what the rover sees. This transformed data is with respect to the rover. The rover position is known in x,y, and orientation yaw. By using these three inputs, the rover “view” can be transformed into a global view. This leverages the function pix\_to\_world.

#### Update world map

The world map pixels can be set based off of the rover’s vision from our previous discussion. However applying these updates linearly with time implies that data will be overwritten with newer data. Newer data may not necessarily be better data.

The data class was modified to include a count at each pixel. Each time a pixel is identified as navigable, the green channel goes up by 1. Each time a pixel is identified as an obstacle, the red channel is increased by one.

The world map is then updated to present navigable pixels and obstacle pixels based off of which has a higher count.

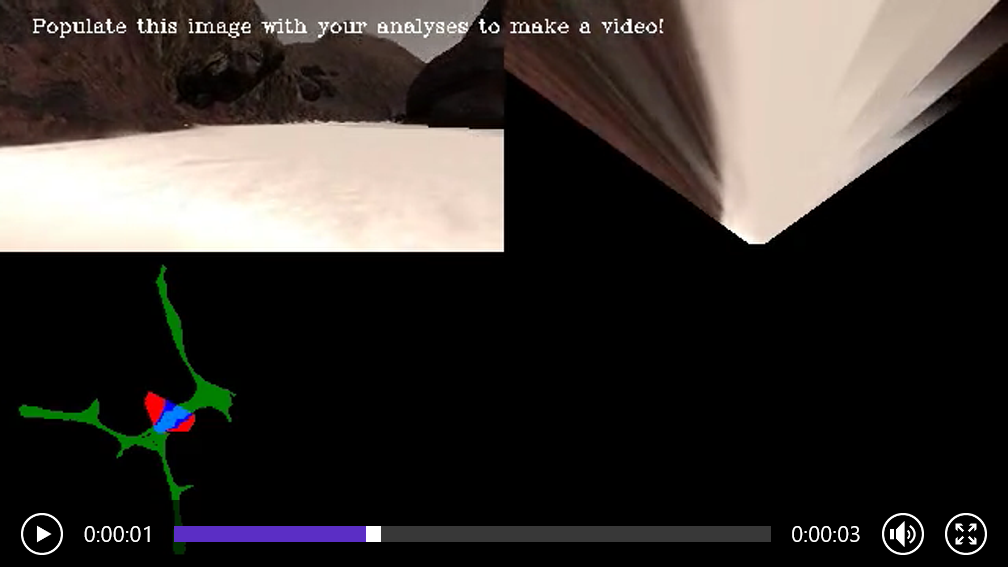
#### Create images

3 sets of images are then created. The rover raw view, a rover perspective transformation, and a world map. These are displayed in a 2x2 grid formation and images are output into the working directory.

### Movie

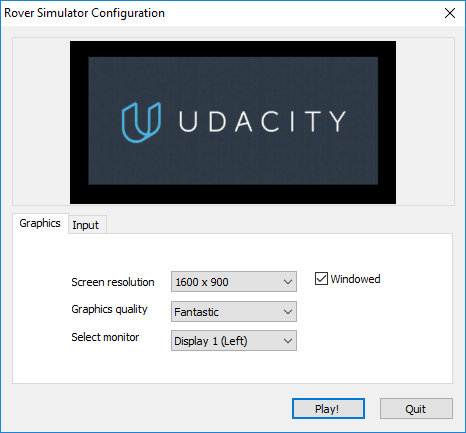
Using moviepy, the images are processed at 60 fps and turned into an mp4.

Please see the working directory ‘../output/test\_mapping.mp4’ to view the video.



# Autonomous Navigation and Mapping

## Settings



## Perception

The perception.py file added several color thresholding functions. These are described previously in the Jupyter notebook.

The unique differences in perception.py are the use of the custom Rover class, as opposed to the Databucket in Jupyter. Variable names were updated correspondingly.

The Rover worldmap was set such that it will only update within a small window of values (plus or minus 3 degrees from nominal) of Rover pitch and Rover roll. This could have been pursued differently, by modifying the rover perception based off the rover pitch and roll.

Perception was also modified to identify rocks. If a rock was identified the navigation is set up to pursue the rock.

## Decision

# perception\_step and #decision step explained