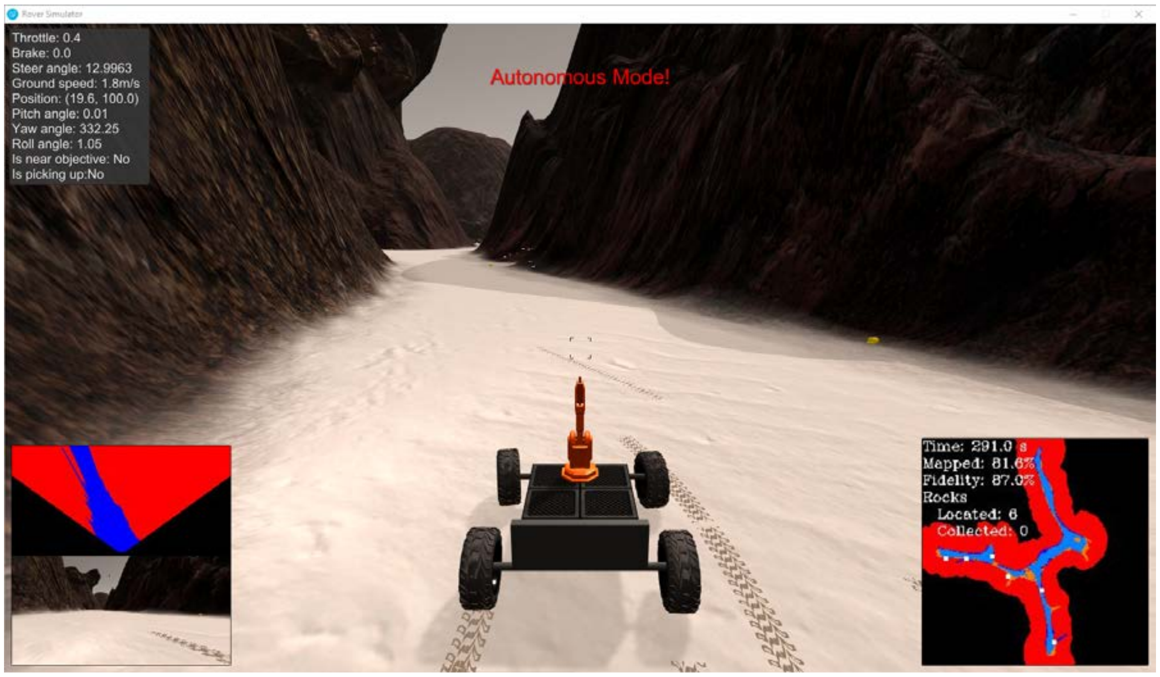

```
Throttle: 0.4
Brake: 0.0
Steer angle: 12.9963
Ground speed: 1.8m/s
Position: (19.6, 100.0)
Pitch angle: 0.01
Yaw angle: 332.25
Roll angle: 1.05
Is near objective: No
Is picking up:No
```

```
Autonomous Mode!
```

```
Time: 291.0 s
Mapped: 81.6%
Fidelity: 87.0%
Rocks
  Located: 6
  Collected: 0
```

Project:
Search and Sample Return

PROJECT 1

Christopher J. Rukas | Robotics Software Engineer | October 21 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 processing visual information around the rover to navigate 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.

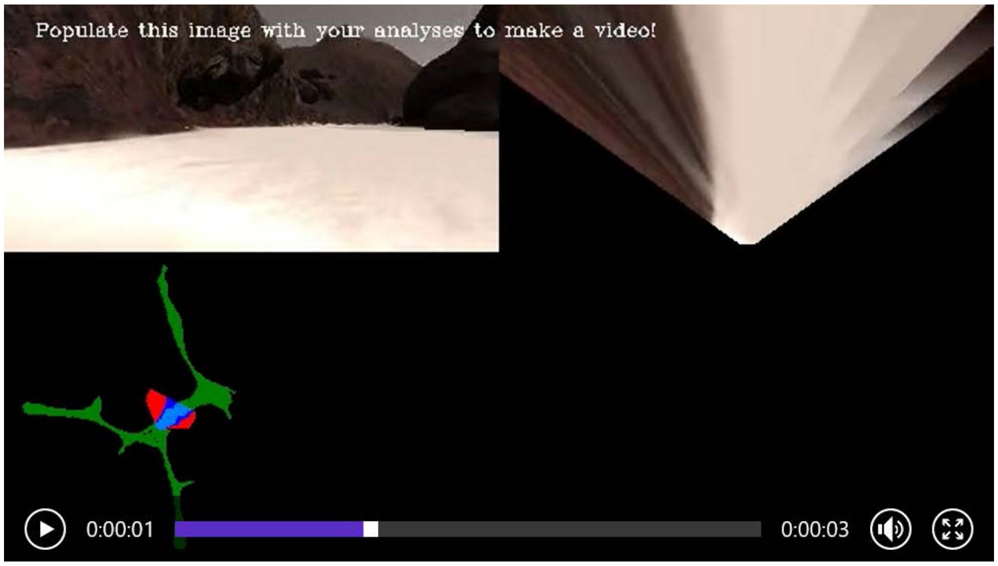
Create images

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

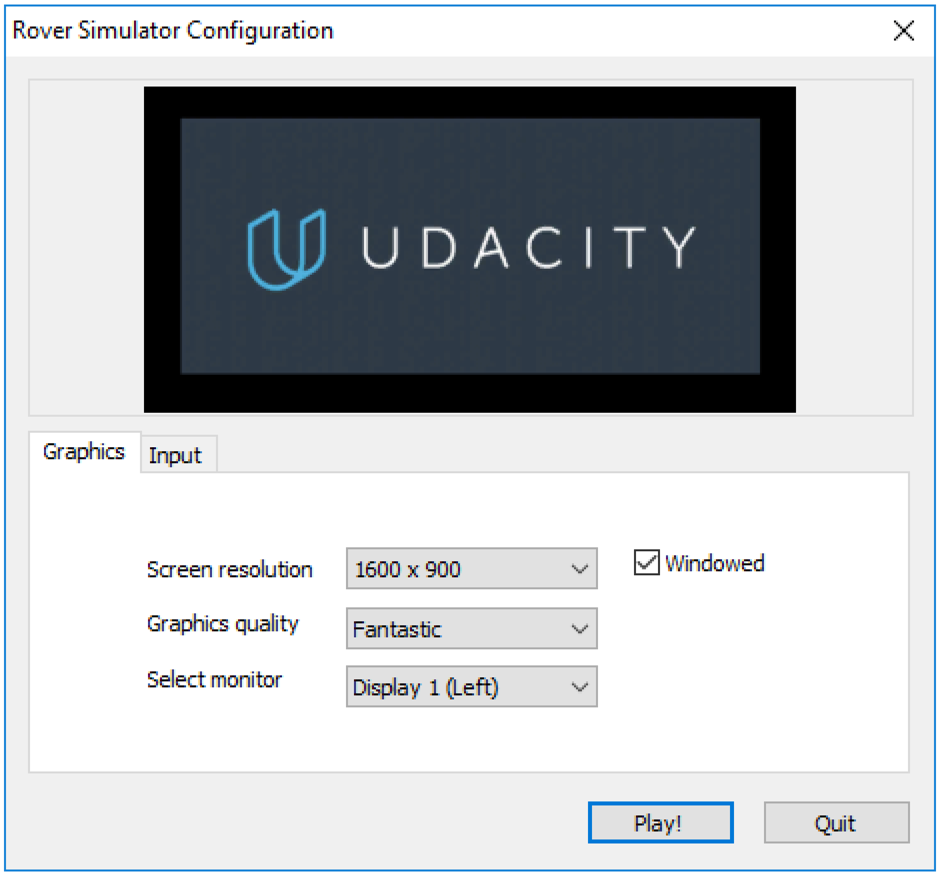
Using moviepy, the images are processed at 60 fps and turned into an mp4.

Please see the working directory '../output/test_mapping.mp4' to view the video.



Autonomous Navigation and Mapping

SETTINGS



Screen resolution: 1600 x 900
Windowed: [✓]
Graphics quality: Fantastic
Select monitor: Display 1 (Left)

On a Dell Precision 7710, maintaining about 35 FPS.

PERCEPTION

The perception.py file is run on each time step by the drive_rover.py script. At a high level the perception file uses the rover's camera to identify navigable terrain, obstacles, and rocks. The information gathered by perception is passed to the Rover.

The perception.py file added several color thresholding functions. These are described previously in the Jupyter notebook. Not every function in the file is used.

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

The Rover worldmap was set such that it will only update if Rover pitch and Rover roll are within a small window of values (plus or minus 3 degrees from nominal). The issue of distorted maps due to vehicle pitch and roll could be 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.

Available "navigable" terrain is passed into the Rover class in polar coordinates.

Perception Advances

With additional time, other processing techniques could be applied to the vision system. The camera could be "windowed" into zones such as left, center, right to help avoid obstacles.

Utilizing the roll and pitch angle of the vehicle, the transformation applied to the camera pixels could be updated to improve fidelity.

The accuracy of pixels analyzed at a distance are less than the accuracy of pixels near the machine. The vision mask could be shortened to limit the available data to improve fidelity. Additionally it may improve "wall following."

DECISION

Decision.py controls the vehicle with an assortment of logic. State machines may be easier to track, but with the limited time an assortment of if statements are used.

Depending on the available navigable pixels, a vehicle speed is selected. These speeds range from 1 to 3 m/s. More navigable pixels means higher vehicle speed. Fewer navigable pixel means lower vehicle speed.

Stuck Detection

The rover has a tendency of becoming stuck on rocks. Some simple concepts were tested to address this issue.

A check is done on a narrow range of vision, plus or minus five degrees of radial vision. If this range is small, the vehicle's "navigable" terrain is told to shift to the right, hopefully avoiding smaller rocks. This has had limited success. The tuning of "navigable" pixels likely needs more adjustment.

Yaw is measured over the previous 5 frames by constantly updating an array with the latest information. The variance of this is recorded to help identify if the vehicle is stuck.

The last vehicle position is measured every x seconds and compared to the current vehicle position. If this value is too small, the vehicle is considered "stuck."

When the vehicle is considered “stuck” the vehicle performs a motion very similar to the “stop” state, in which the rover comes to stop and completes a four wheel steering maneuver.

Rock Pick Up

When perception identifies a rock, it provides the direction and distance of those pixels instead of navigable pixels. The maximum vehicle speed is greatly reduced to 0.5 m/s. When Rover.near_sample is true, the vehicle comes to a stop and is sent a command to pick up the rock.

Driving

Generally the vehicle drives forward and in the mean direction of navigable terrain. A several degree bias is provided to encourage the machine to follow a wall.

Forward

Forward will increase and decrease throttle to maintain the desired vehicle speed.

Stop

Stop will stop the vehicle and spin the vehicle until enough navigable terrain is visible.

Stuck

Stuck will stop the vehicle and spin the vehicle randomly (150 frames).

Else

The previous samples require there to be defined Robot nav angles provided from perception. If no navigation angles are identified, a rover throttle is applied to start some vehicle motion. If the vehicle is near a sample however, it’ll stop the vehicle.

Decision Advances

The decision tree could be updated such that the vehicle would prefer to navigate terrain that is unexplored.

The x,y data of a rock could be saved and it could navigate based off of coordinates rather than visual perception.

Path planning could be introduced to improve the steering performance of the vehicle.

CONCLUSION

The grading rubric is used to lay out the order of the report. The rover is capable of navigating the world to the initial requirements of 40% mapping and 60% fidelity. However, the success is somewhat dependent on the random starting position of the rover. It can occasionally become stuck going in circles.

The course is a great introduction to Python language and libraries such as OpenCV. It represents a challenging topic with various levels of challenges for students of different skill levels.

As of my current progress, the machine navigates successfully (usually) and can even identify and pick up a rock. I am currently stuck in a position where the rover will not continue moving after picking up a rock. The Rover.pick_up variable does not seem to reset.

All information related to this project can be found under the github address: https://github.com/rukie/Udacity-RSE