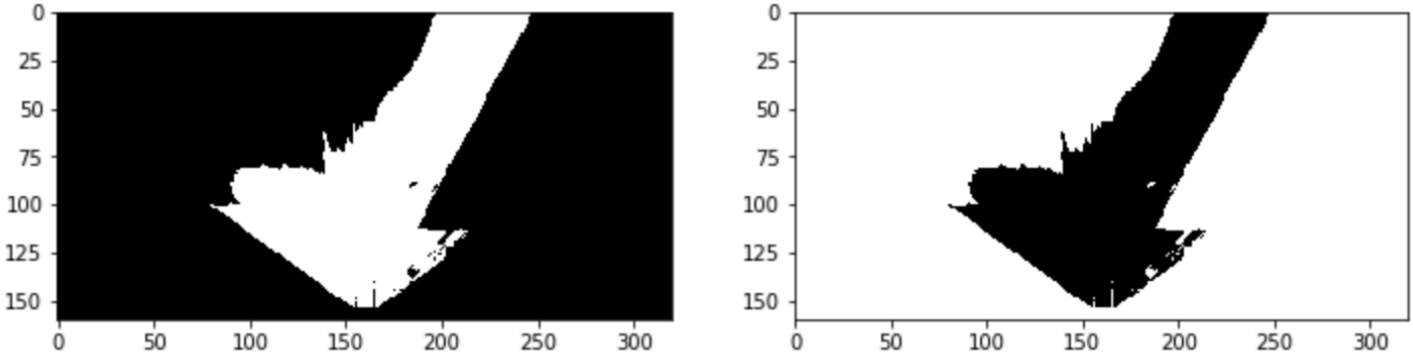
Project: Search and Sample Return

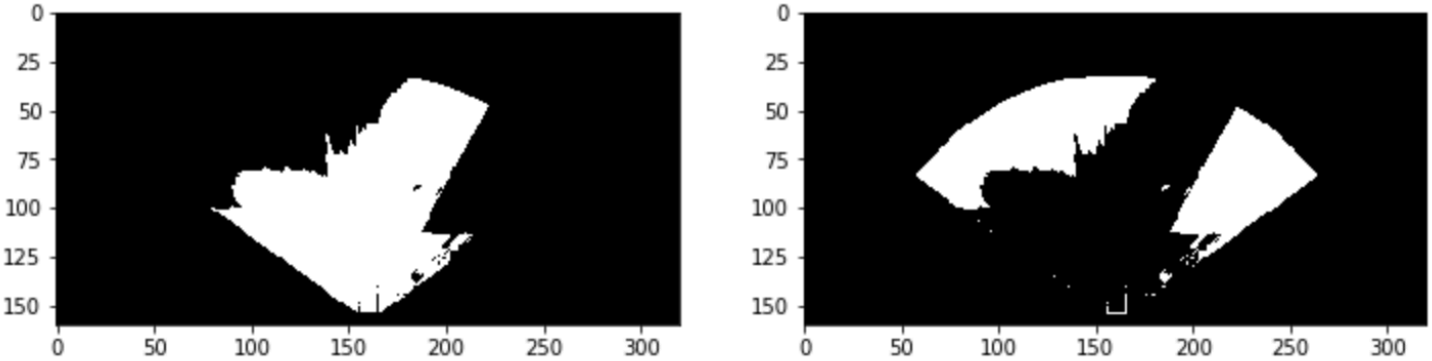
Notebook Analysis

Quest 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 original function `color_thresh()` provided by the sample code was able to distinguish terrian from the obstacles in the imported image. A simple operation that extracting the terrain from the full vision image can be used to output the obstacles. The picture below shows the rover-center view of the terrain and obstacles sorround the rover. White part in the left picture indicates terrain and in the right picture shows the obstacle.



However, since the camera mounted on the rover has only limited vision range (it cannon see behind the rover itself), a musk was suggested to filter out the areas that are unseen by the rover camera. In addition, the mask is also blocking out the objects that are beyond a fixed distance from the camera. By doing so, we can focus on the closer objects in the later operation. The left picture below shows the terrain (white color) and the right one shows the obstacle (white color).

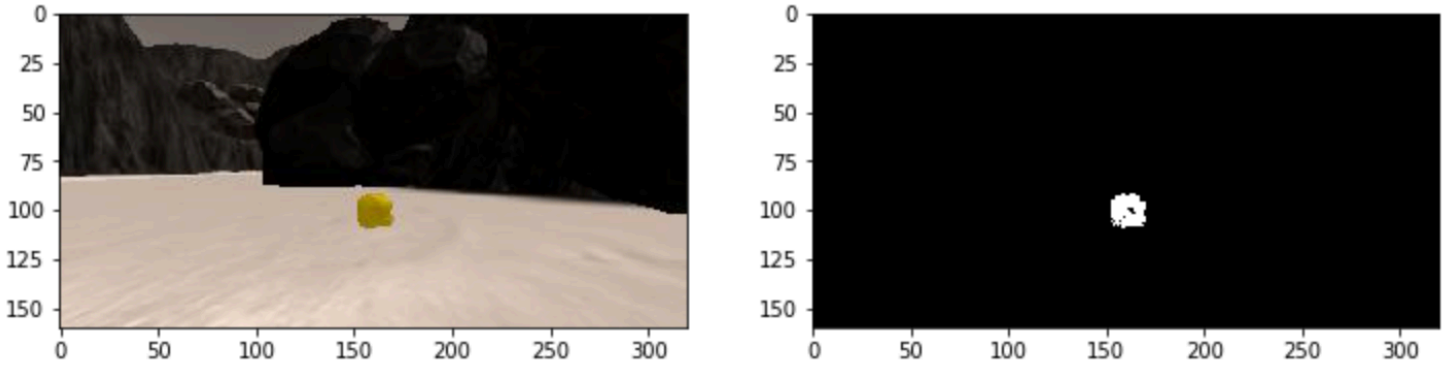


To idenfy the rock samples from the imported images, a new function `find_rock()` was added based on

the original function `color_thresh()`. The only change was on the threshold critaria. Since the rocks usually has color around [110, 100, 50], so the threshold on blue color was set to less than the inputed value.

```
(img[:,:,2] < rgb_thresh[2])
```

Picture below shows the image with a rock and the output result after the `find_rock()` filter.



Quest 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.

Implemented the `prcess_image()` by following the suggested steps:

1) Define source and destination points for perspective transform

Simply use the suggested values in the code.

2) Apply perspective transform

Key changes is adding the mask variable as the second output from the `perspect_transform()` function.
`warped, mask = perspect_transform(img, source, destination)`

3) Apply color threshold to identify navigable terrain/obstacles/rock samples

Use the `color_thresh()` function and `find_rock()` function to generate the pixel maps for each types.

4) Convert thresholded image pixel values to rover-centric coords

Use the `rover_coords()` function to convert each maps.

5) Convert rover-centric pixel values to world coords

Use the `pix_to_world()` function to convert each maps to world coordinates.

6) Update worldmap (to be displayed on right side of screen)

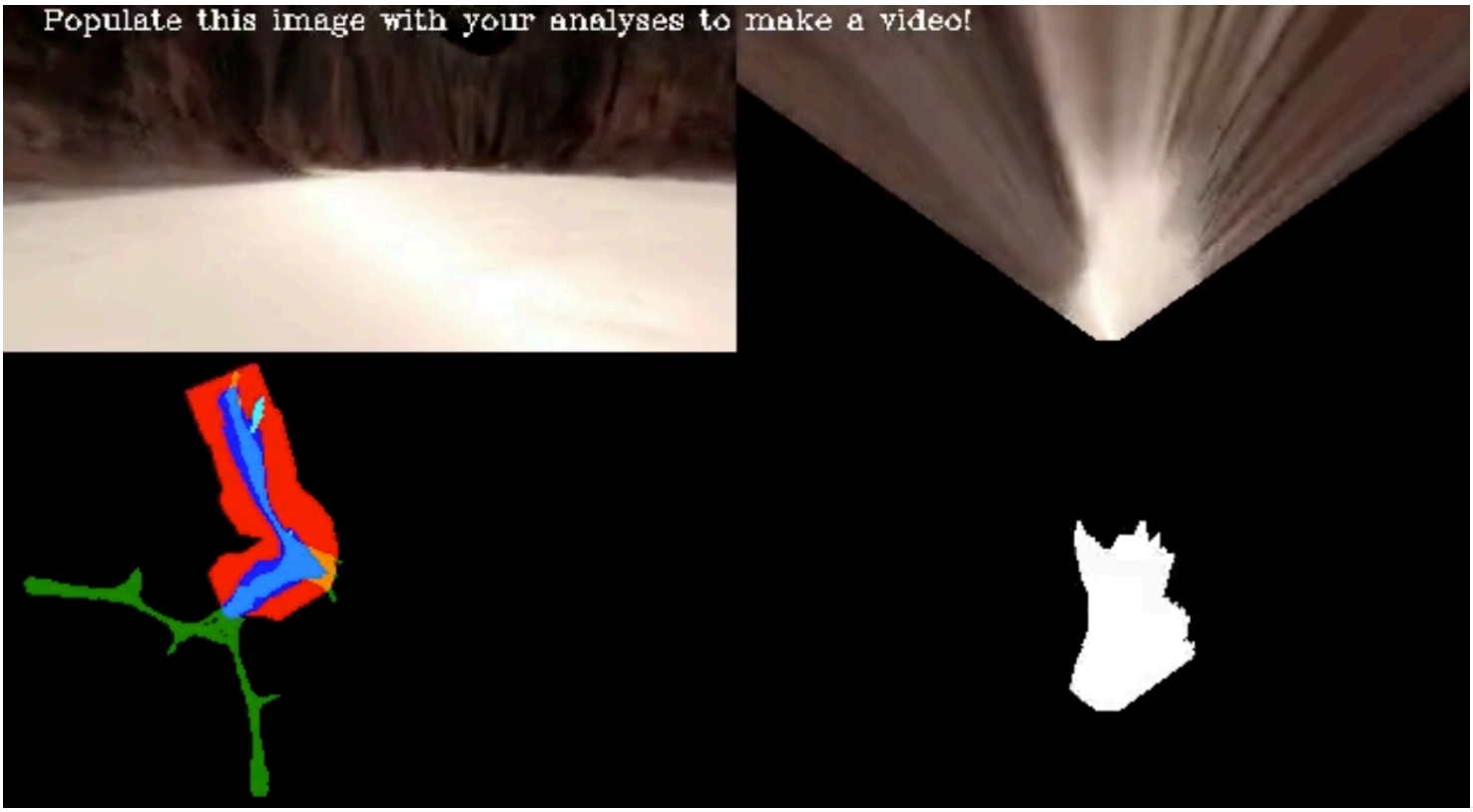
Before updating the worldmap, the rover's roll and pitch angles are used to block out some abnormal image results. To do so, the roll and pitch angles are mapped to [-180, 180) range, and then converted to absolute values. If the absolute angles are less than 5 degrees, continue to update the map. Otherwise, the following steps will be skipped.

To update the map, simply assign the red channel to 255 for the obstacles, blue channel to 255 for the terrain. If they are overlapped, force changing it to terrain and reset its red channel to 0.

Note: that this is not a good method to distinguish the terrian and obstacle. It will be changed in the later autonomous navigation program.

The code will automatically generate a video to show the image process result. A new video `.\output\test_mapping1.mp4` was generated based on the recorded images.

The following picture shows the final screenshot of the video. The top left picture is the original camera image, and the lower left is the terrain, obstacles, and rock mapped on the world map. On the right hand side, top picture shows the present image mapped to the rover-centric coordinate, and the lower picture shows the terrain identified in the present image.



Autonomous Navigation and Mapping

Quest 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.

The task is to complete and adjust the `perception_step()` function in the `perception.py` and `decision_step()` function in the `decision.py`. In addtion to those, I also changed the `Rover` class defined in the `drive_rover.py` file to have a few properties used in the code.

In the `percetion.py`, I mainly implemented the `perception_step()` function based on the same process done in the notebook to process the imported image and update the worldmap. I also added the `fine_rock()` function and changed the `perspect_transform()` function to accommodate the `mask` used to block out the unseen area from the camera.

A key change in the `preception_step()` that is different from it in the notebook is at updating the Rover worldmap. Instead forcing the overlapped obstacle and terrain to terrain, I added another attribute `colormap` in the Rover class to stored the color scores at each location. Only when the terrain score is higher than the obstacle, the position is assigined to terrain. Vice versa. This method helped to improve the fidality during autonomous mapping.

```
        nav_pix = 5 * Rover.colormap[:, :, 2] > Rover.colormap[:, :, 0]
        obs_pix = 5 * Rover.colormap[:, :, 2] < Rover.colormap[:, :, 0]
        Rover.worldmap[nav_pix, 0] = 0
        Rover.worldmap[obs_pix, 2] = 0
```

In the `decision_step()` function, a thrid mode, `idel` was implemeted to try to rescue the rover when it was stuck by smaller obstacles and could not move forward. An `idel_time` variable is used to record when long the rovoer has been stuck and after it is over a threthold, the rover will stop and move backward with a tiny turn.

```
        # If the rover is stuck at some point, drive backward and turn
        elif Rover.mode == 'idel':
            if Rover.idel_time > 0:
                Rover.throttle = -0.5 * Rover.throttle_set
                Rover.steer = -5
                Rover.idel_time -= 1
            else:
                Rover.throttle = Rover.throttle_set
                # Release the brake
                Rover.brake = 0
                # Set steer to mean angle
                Rover.steer = np.clip(np.mean(Rover.nav_angles * 180/np.pi), -15, 15)
                Rover.idel_time = 0
                Rover.mode = 'forward'
```

Another minor change in the `decision_step()` function is adding another speed level in `forward` mode. The speed will be reduced to half of the maximum speed when the detected navigation distance is shorter. By doing so, I was trying to prevent the rover ran into obstacles with full speed and causes it rolls over.

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

The picture below shows a screenshut when the rover running automactically. A video `.\output\autonomous_drive` was recorded to show the program execution.

The program is able to map 60% of the world with 90% fadility rounghly in 3 minutes with the default speed. A few improvements might be taken in the future are:

1) Implement new mapping and decision stragy to explore the unknown area.

A quick idea is to add another layer of the map in rover class so it can record the positions it has gone through. Then, when converting terrain position heading angleadd, add weighting factors on the positions based on the position history so the rover could head to unknown area.

Ultimately, intelligent autonomous driving algorithm such as A* method to find optimal path could be implemeted to efficienly explore the entire worldmap.

2) Implement better rover driving stragy so it can fast and smoothly run through the terrain.

Currently the rover could not agilely avoid the small rocks on the road. Also the rover 'shakes' it heading direction pretty large in response to the terrain change even though the obstacles are not directly in front of the rover. A better throttle control and filter with respect to the non-significant object could improve the rover motion.

3) Added a function to display current rover location on the worldmap.