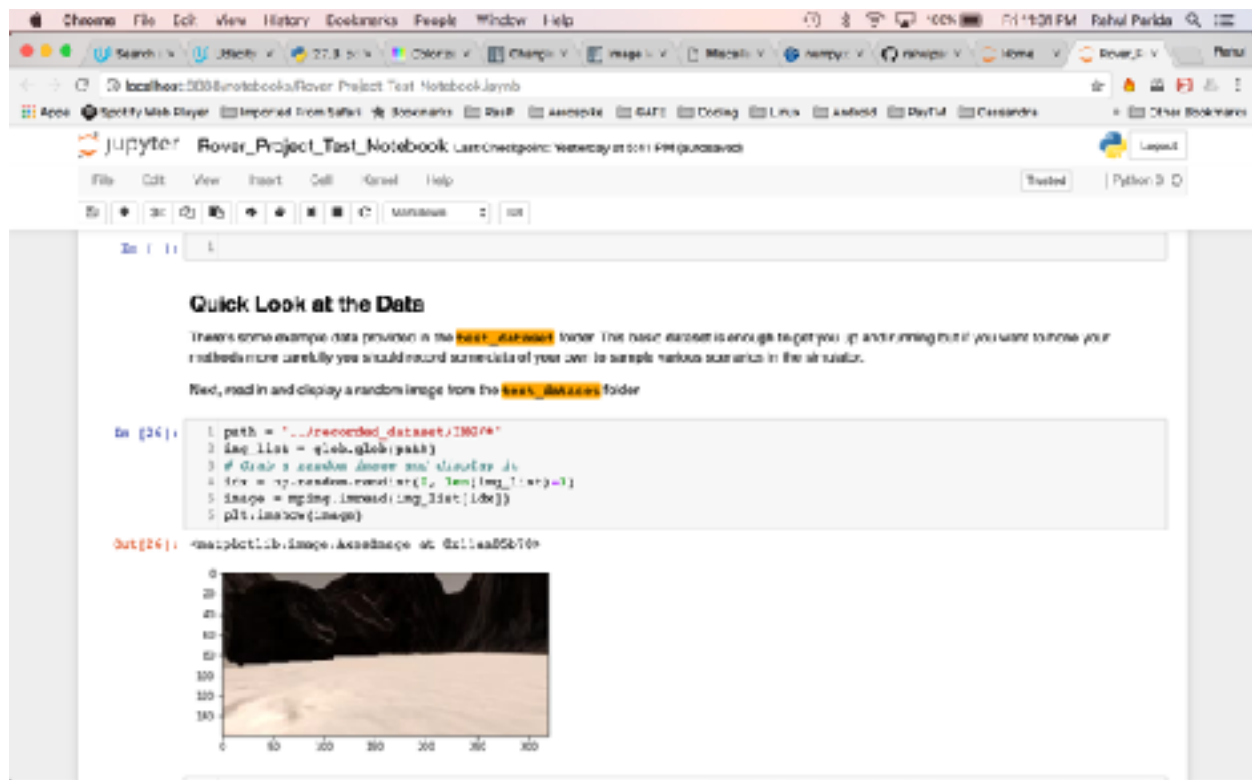


Search And Sample Return Project

Notebook Analysis

1. After recording data in training mode, I changed the path from which the images were being fetched i.e. changed *test_dataset* to *recorded_dataset* in the following code in the notebook.



2. Then I used both the calibration images *grid_img* and *rock_img* for finding out the approx coordinates at the edges of the grid square and finding the color range for the rock samples. The approx pixel coordinates were used for the perspective transform

3. For the color thresholding, I modified the function to accept extra parameters which applied threshold on the RGB values on the image according to the range provided in the parameters. I also defined two other functions *color_thresh_2* which applies the color thresholding using opencv and *rgb_to_bgr* to convert RGB to BGR channel config to apply to the previous function. The color values for shades of yellow are hardcoded (values calibrated manually).

```

In [22]:
1 # Similarly, please write the threshold
2 # Threshold of RGB = 194 since a value less of demitifying ground pixels only
3 def color_thresh(img, rgb_thresh_low=[140, 140, 140], rgb_thresh_high=[255, 255, 255]):
4     # Create an array of zeros same size as img, but single channel
5     mask = np.zeros_like(img, dtype=bool)
6     # Assume that each pixel is above all three threshold values so RGB
7     # above threshold will now create a boolean array with "True"
8     # where threshold was met
9     mask = (img[:, :, 0] > rgb_thresh_low[0]) & (img[:, :, 1] > rgb_thresh_low[1]) &
10         (img[:, :, 2] > rgb_thresh_low[2]) & (img[:, :, 0] < rgb_thresh_high[0]) &
11         (img[:, :, 1] < rgb_thresh_high[1]) & (img[:, :, 2] < rgb_thresh_high[2])
12     # Filter the array of zeros with the boolean array and set to 1
13     mask[mask == 0] = 1
14     # Return the binary image
15     return mask
16
17 def write_to_image(img):
18     img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
19     img_bar = cv.cvtColor(img, cv.COLOR_RGB2BGR)
20     return img_bar
21
22 def color_thresh_2(img):
23     img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
24     low = cv.cvtColor(img, cv.COLOR_RGB2BGR)
25     upper_low = np.array([140, 140, 140], dtype=int)
26     upper_high = np.array([255, 255, 255], dtype=int)
27     # Threshold the RGB image to get only blue pixels
28     mask = cv.inRange(low, upper_low, upper_high)
29     return mask
30
31 threshold = color_thresh(img)
32 plt.imshow(threshold, cmap='gray')
33 plt.show()
34 plt.imshow(threshold, cmap='gray')
35 plt.show()
36 plt.imshow(color_thresh_2(img), cmap='gray')
37 plt.show()

```

Out[22]:

```
43 # Rover vision_image[s, r, 0] = color_thresh(image)
44 # Rover vision_image[s, r, 1] = color_thresh(image, sph_thresh_low=133.75, 179.75, sph_thresh_high=185, 281,
45 # Rover vision_image[s, r, 2] = 1 - Rover.vision_image[:, :, 0]
46
47 # # Convert map image pixel values to rover-centric coords
48 # # Convert rover-centric pixel values to world coordinates
49 # # Update Rover velocity (to be displayed as robot state of screen)
50 # Example: Rover.worldmap(obstacle_y_world, obstacle_x_world, 0) == 1
51 # Rover.worldmap(rover_y_world, rock_x_world, 0) == 1
52 # Rover.worldmap(naviable_y_world, naviable_x_world, 0) == 1
53
54 obstacle_rover_coords = rover_coords[0] - color_thresh(image)
55 rock_rover_coords = rover_coords[color_thresh(image, sph_thresh_low=133.75, 179.75, sph_thresh_high=185, 281,
56 naviable_rover_coords = rover_coords[color_thresh(image)
57
58 obstacle_x_world, obstacle_y_world = pix_to_world(obstacle_rover_coords[0], obstacle_rover_coords[1])
59
60 rock_x_world, rock_y_world = pix_to_world(rock_rover_coords[0], rock_rover_coords[1], Rover.ypos[data.count], 0)
61
62 naviable_x_world, naviable_y_world = pix_to_world(naviable_rover_coords[0], naviable_rover_coords[1], Rover.ypos[data.count], 0)
63
64 Rover.worldmap(obstacle_y_world, obstacle_x_world, 0) == 1
65 Rover.worldmap(rock_y_world, rock_x_world, 0) == 1
66 Rover.worldmap(naviable_y_world, naviable_x_world, 0) == 1
67
68 # # Convert rover-centric pixel positions to polar coordinates
69 # Update Rover pixel distances and angles
70 # Rover.cur_dist = rover_center pixel distance
71 # Rover.cur_angle = rover_center angle
72
73 Rover.cur_dist, Rover.cur_angle = to_polar_coordinates(rover_coords[0], naviable_rover_coords[1])
74
75 # # Make a moving image, below is some example code
76 # Plot using a black image (no in distance, show red line)
```


Decision Step

I have modified and added several pieces of code in the *decision_step* function for picking up rocks and for detecting and recovering from a state where the rover is stuck.

1. The logic when the rover remain in *stop* mode remains the same, however, I have moved the function to the top in *if-else* as *stop* mode now also serves as a way to recover from the state when the rover is stuck and unable to move.

[illegible]

2. The next condition checks whether the Rover is stuck and if it is, it invokes a function called *launch_recovery*. In my code, I detect if a Rover is stuck by analyzing the *Rover.vel* and *Rover.throttle* values. If the *Rover.vel* value is below the threshold 0.01 and *Rover.throttle* is set

```

# Press F for help
44 Rover.brake = Rover.brake_set/2
45 else:
46     Rover.brake = 0
47     Rover.stuck = False
48     print('OK!')
49     # If Rover mode is 'forward'
50     if Rover.mode == 'forward':
51         print('***** Forward Mode *****')
52         # Check the extent of navigable terrain
53         if len(Rover.nav_angles) == Rover.stop_forward:
54             # If mode is forward, navigable terrain looks good
55             # and velocity is below max, then throttle
56             if Rover.vel < Rover.max_vel:
57                 # set throttle value to throttle setting
58                 Rover.throttle = Rover.throttle_set
59             else:
60                 # Fine tune
61                 Rover.throttle = 0
62                 Rover.brake = 0
63                 # Set steering to average angle clipped to the range +/- 15
64                 Rover.steer = np.clip(np.mean(Rover.nav_angles) * 180/np.pi, -15, 15)
65                 # If there's a lack of navigable terrain ahead then go to 'stop' mode
66                 # If there's a lack of navigable terrain ahead then go to 'stop' mode
67                 if len(Rover.nav_angles) < Rover.nav_thresh:
68                     launch_stuck_recovery()
69             # Don't make the rover go spinning
70             # even if no modifications have been made to the code
71             else:
72                 Rover.throttle = Rover.throttle_set
73                 Rover.steer = 0
74                 Rover.brake = 0
75                 # If Rover mode is 'stop' and Rover.vel is 0.01
76                 if Rover.mode == 'stop' and Rover.vel == 0.01:
77                     # If it is a state where we need to pick up a rock send pickup command
78                     if Rover.pickup == 0 and Rover.pickup == 0.2 and not Rover.picking_up:
79                         Rover.send_pickup = True
80                 return Rover
81     # Launch stuck recovery screen
82     print('LAUNCH STUCK RECOVERY SCREEN')
83     print('***** Launch stuck recovery *****')
84     Rover.throttle = 0
85     Rover.brake = Rover.brake_set
86     Rover.steer = 0
87     Rover.mode = 'stop'
88     # Check stuck recovery
89     print('CHECK STUCK RECOVERY')
90     if Rover.vel < 0.01:
91         Rover.throttle = 0
92     else:
93         # If Rover mode is 'stop'
94         if Rover.mode == 'stop':
95             if Rover.vel < 0.01 and Rover.throttle == 0.0:
96                 return True
97     return False

```

to max 1.0. Then I assume that the Rover is stuck and therefore launch stuck recovery. In stuck recovery all I do is to apply the brakes and set *Rover.throttle* and *Rover.steer* values to 0 and change the *Rover.mode* to *stop*. The same set of operations which were applied when there was a dearth of navigable terrain in *forward* mode. The *stop* mode handles the rest.

- A. If the location of the rock is present out of the bounds in which the Rover can turn and the Rover is moving at a high speed (> 0.2 threshold), I apply the brakes and set *throttle* to 0. Or if the Rover's velocity is lower than the above threshold, I just set the *brakes* and *throttle* values to zero. This enables the Rover to stop and align itself towards the rock sample.
- B. If the location of the rock is present within the bounds in which the Rover can turn, I try to maintain $Rover.vel < 1$ by braking when it exceeds the threshold. I also set the *throttle* value to be the inverse of the mean rock distance from the Rover (rock is closer implies that the throttle should be low).

```

12 # Turn angle is +/- 15 degrees, when stopped the next line will induce 4-wheel turning
13 Rover steer = -15 # Could be more clever here about which way to turn
14 # If we're stopped but are sufficient navigable terrain in front then go!
15 elif (Rover.nav_angle) <= Rover.go_forward:
16     # Set throttle back to normal value
17     Rover.throttle = Rover.throttle_max
18     # Release the brake
19     Rover.brake = 0
20     # Set steer to max angle
21     Rover.steer = np.clip(np.mean(Rover.nav_angle + 100*np.pi, -35, 35)
22     Rover.mode = 'forward'
23
24     # If stuck(Rover)
25     return State(Rover)
26
27 # If stuck(Rover, nav_angle) == 1
28 elif (Rover.nav_angle) <= 1:
29     # If stuck(Rover, nav_angle) == 1
30     # If stuck(Rover, nav_angle) == 1
31     # If stuck(Rover, nav_angle) == 1
32     # If stuck(Rover, nav_angle) == 1
33     # If stuck(Rover, nav_angle) == 1
34     # If stuck(Rover, nav_angle) == 1
35     # If stuck(Rover, nav_angle) == 1
36     # If stuck(Rover, nav_angle) == 1
37     # If stuck(Rover, nav_angle) == 1
38     # If stuck(Rover, nav_angle) == 1
39     # If stuck(Rover, nav_angle) == 1
40     # If stuck(Rover, nav_angle) == 1
41     # If stuck(Rover, nav_angle) == 1
42     # If stuck(Rover, nav_angle) == 1
43     # If stuck(Rover, nav_angle) == 1
44     # If stuck(Rover, nav_angle) == 1
45     # If stuck(Rover, nav_angle) == 1
46     # If stuck(Rover, nav_angle) == 1
47     # If stuck(Rover, nav_angle) == 1
48     # If stuck(Rover, nav_angle) == 1
49     # If stuck(Rover, nav_angle) == 1
50     # If stuck(Rover, nav_angle) == 1
51     # If stuck(Rover, nav_angle) == 1
52     # If stuck(Rover, nav_angle) == 1
53     # If stuck(Rover, nav_angle) == 1
54     # If stuck(Rover, nav_angle) == 1
55     # If stuck(Rover, nav_angle) == 1
56     # If stuck(Rover, nav_angle) == 1
57     # If stuck(Rover, nav_angle) == 1
58     # If stuck(Rover, nav_angle) == 1
59     # If stuck(Rover, nav_angle) == 1
60     # If stuck(Rover, nav_angle) == 1
61     # If stuck(Rover, nav_angle) == 1
62     # If stuck(Rover, nav_angle) == 1
63     # If stuck(Rover, nav_angle) == 1
64     # If stuck(Rover, nav_angle) == 1
65     # If stuck(Rover, nav_angle) == 1
66     # If stuck(Rover, nav_angle) == 1
67     # If stuck(Rover, nav_angle) == 1
68     # If stuck(Rover, nav_angle) == 1
69     # If stuck(Rover, nav_angle) == 1
70     # If stuck(Rover, nav_angle) == 1
71     # If stuck(Rover, nav_angle) == 1
72     # If stuck(Rover, nav_angle) == 1
73     # If stuck(Rover, nav_angle) == 1
74     # If stuck(Rover, nav_angle) == 1
75     # If stuck(Rover, nav_angle) == 1
76     # If stuck(Rover, nav_angle) == 1
77     # If stuck(Rover, nav_angle) == 1
78     # If stuck(Rover, nav_angle) == 1
79     # If stuck(Rover, nav_angle) == 1
80     # If stuck(Rover, nav_angle) == 1
81     # If stuck(Rover, nav_angle) == 1
82     # If stuck(Rover, nav_angle) == 1
83     # If stuck(Rover, nav_angle) == 1
84     # If stuck(Rover, nav_angle) == 1
85     # If stuck(Rover, nav_angle) == 1
86     # If stuck(Rover, nav_angle) == 1
87     # If stuck(Rover, nav_angle) == 1
88     # If stuck(Rover, nav_angle) == 1
89     # If stuck(Rover, nav_angle) == 1
90     # If stuck(Rover, nav_angle) == 1
91     # If stuck(Rover, nav_angle) == 1
92     # If stuck(Rover, nav_angle) == 1
93     # If stuck(Rover, nav_angle) == 1
94     # If stuck(Rover, nav_angle) == 1
95     # If stuck(Rover, nav_angle) == 1
96     # If stuck(Rover, nav_angle) == 1
97     # If stuck(Rover, nav_angle) == 1
98     # If stuck(Rover, nav_angle) == 1
99     # If stuck(Rover, nav_angle) == 1
100    # If stuck(Rover, nav_angle) == 1

```

5. I have also added a small snippet at the end of the *decision_step* which brakes hard when the Rover is near a rock sample and has a very high velocity.

All the above changes enable the Rover to recover maximum rock samples without compromising fidelity of mapping.