Search and Mapping Report

Machine: 2016 MacBook Pro

Simulation Version: Latest

Project Version: Latest

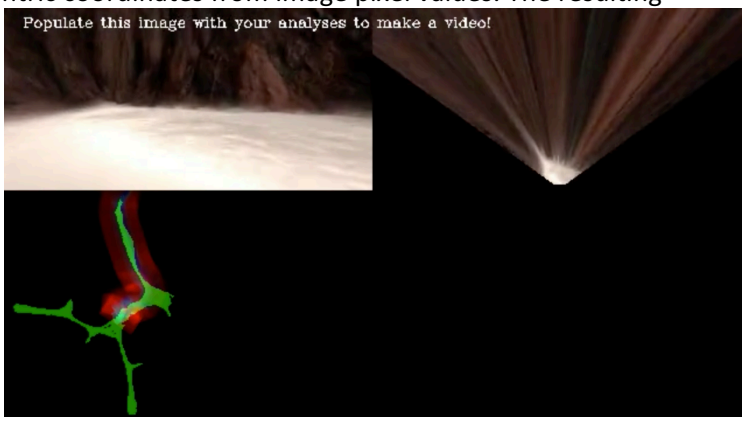
Screen Resolution: 640 x 480

Graphics Quality: Good

FPS: 38

1 NOTEBOOK ANALYSIS

1. To identify other objects other than the navigable terrain, I modified the color_thresh function to allow identification of objects that are below threshold. This creates greater flexibility, in addition to getting the best RGB combination to identify objects. I added another input parameter for the color_thresh function called above_thresh. By default, above_thresh is set to true. In the body of the function a conditional statement was created to differentiate between identifying objects that are above or below thresholds. The function then returns an output with the desired results.
2. To create a worldmap that includes the navigable terrain, obstacles, and rock samples, I populated the function with what I learned in class and modified the color_thresh to make this possible. The first step to initiate was to gather the source and destination points for the perspect_transform function. After applying the transform, we take the resulting image and apply the color threshold three times for the terrain, obstacles, and rock samples. The result will create three binary images. Each image is entered in as a parameter in the rover_coords function to obtain the rover-centric coordinates from image pixel values. The resulting coordinates are then fed into the pix_to_world function along with the rover's global position and heading, world size, and a scale to map the objects appropriately in the map. Lastly, the map is updated for every call to the process_image function.



AUTONOMOUS NAVIGATION AND MAPPING

1. The perception step in the project was implemented very similar to the notebook analysis. Just like in the notebook analysis, the color_thresh was modified to properly identify objects. In addition, the Rover vision is updated constantly using the binary images and obtaining the rover's navigation angles and distances using the to_polar_coords function to help determine the steering angle. One major addition to this script was creating a threshold when updating the world map to increase map fidelity:

```
if (Rover.roll <= 1.0 and Rover.pitch <= 1.0):
    Rover.worldmap[rock_y_world, rock_x_world, 1] += 1
    Rover.worldmap[navigable_y_world, navigable_x_world, 2] += 1
```

   In the decision step, I split the body of the conditionals to separate functions called move_forward and stop_moving both taking a Rover object as an input.

   a. For the move_forward function the following additions were made:
      i. To reduce understeer when moving, the condition Rover.roll > 10 then coast was added to slow the rover down:

```
if Rover.roll > 10 and Rover.vel > 1:
    print ('TOO FAST AND TOO MUCH ROLL...GETTING OF THE GAS...')
    Rover.throttle = 0
    if Rover.vel > 1.5:
        print ('WAY TOO FAST, BRAKING...')
        Rover.brake = 0.15
```

      ii. To reduce invalid images when moving, the steering angle for Rover.steer was reduced from 15 to 10 if the ground velocity was greater than 1.5. In turn, this reduces the pitch and roll enabling the rover to create valid images:

```
steering_angle = 15 if Rover.vel < 1.5 else 12
```

      iii. In an attempt to get unstuck from obstacles, a new mode was implemented to reverse when the rover gets stuck, which is when the rover has plenty of navigable terrain and the rover is accelerating, but the rover is not moving. This mode reverses when the rover and only stops when the distance between the rover and the object meets a certain threshold and it surpasses the number of angles threshold.

```
if Rover.vel == 0 and Rover.throttle > 0:
    print ('STUCK...')
    Rover.throttle = 0
    Rover.mode = 'reverse'
```

      This mode is called from move_forward when the conditions are met of getting stuck.

iv. When the rover's mean distance is reduced to a certain threshold, the rover slows down so it can brake in time instead of crashing into an obstacle:

```
if np.mean(Rover.nav_dists) < 20:
    print ('LIMITED DISTANCE FORWARD')
    if Rover.vel > 1:
        print ('LIMITED VIEW FORWARD: BRAKING...')
        Rover.throttle = 0
        Rover.brake = 0.1
    else:
        print ('LIMITED VIEW FORWARD: COASTING...')
        Rover.throttle = 0
```

v. Finally, when the near sample flag is raised, the rover brakes immediately and stops to attempt to pick up the sample

```
if Rover.near_sample:
    print ('NEAR SAMPLE: STOPPED')
    Rover.throttle = 0
    Rover.brake = 100
    Rover.mode = 'stop'
```

b. For the stop_moving function, the only addition is to the rover modes when the rover is fully stopped. When the rover had finished turning, the rover mode is set to stop so the rover can assess the situation and keep turning if need be.

2. After launching in autonomous mode, the rover accomplished the passing requirements but could be optimized more. These optimizations include:
    a. The rover steering preference should head toward unexplored areas to create a more efficient search or avoiding areas that have already been visited. This will yield to a faster search time and having the whole map explored.
        i. To accomplish this task, I would cache positions in the map to where I can check those positions and steer away from them or maybe just check the worldmap to see positions that have already been marked and stay away from those places
    b. Hugging the wall to where the rover does not get stuck often. Hugging the wall leads to a completely explored map, however, the more daunting task in doing this is to figure a way to be close enough to the wall to be able to pick rock samples while avoiding getting stuck.
        i. For this I would, pick a side of the rover and steer towards that side until I get close enough to a wall/obstacle and create a distance threshold between the wall/obstacle and the rover that if it goes below a certain distance it would steer the opposite way. The color threshold maybe optimized further by getting a better RGB threshold to distinguish between obstacle and terrain.