1. Notebook Analysis

There is a function called '*target_filter(img, hsv_thresh_min, hsv_thresh_max)*' inside Rover_Project_Test_Notebook. This function accepts three arguments which are the image to be processed, minimum HSV threshold value and maximum HSV threshold value respectively. The function filters out only the pixel values in between the min and max HSV values. Finally, it returns the binary form of the filtered image. The images below show examples of filtered images with different HSV threshold values.

```
In [19]:
# Filter out only the navigale territorial
navigable = target_filter(rock_img, NAVIGABLE_LOW, NAVIGABLE_HIGH)

fig = plt.figure(figsize=(12,3))
# Plot
plt.subplot(121)
plt.imshow(rock_img)
plt.subplot(122)
plt.imshow(navigable, cmap='gray')
```

```
Out[19]: <matplotlib.image.AxesImage at 0x20dd1c252b0>
```

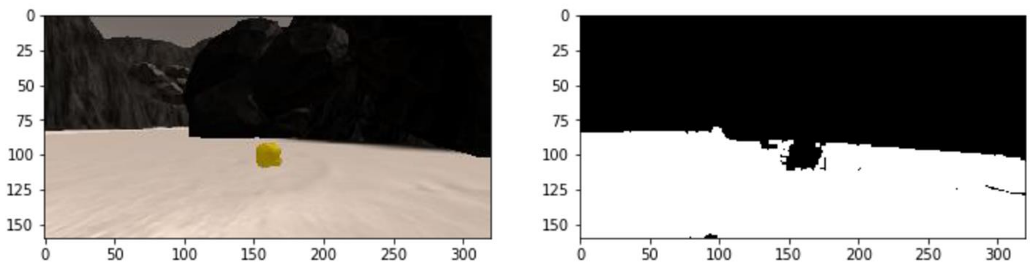


Figure 1: Filtering only navigable area.

```
In [22]:
# Filter out only the obstacle.
obstacle = target_filter(rock_img, OBSTACLE_LOW, OBSTACLE_HIGH)
# Plot
fig = plt.figure(figsize=(12,3))
plt.subplot(121)
plt.imshow(rock_img)
plt.subplot(122)
plt.imshow(obstacle, cmap='gray')
```

```
Out[22]: <matplotlib.image.AxesImage at 0x20dd1e752e8>
```

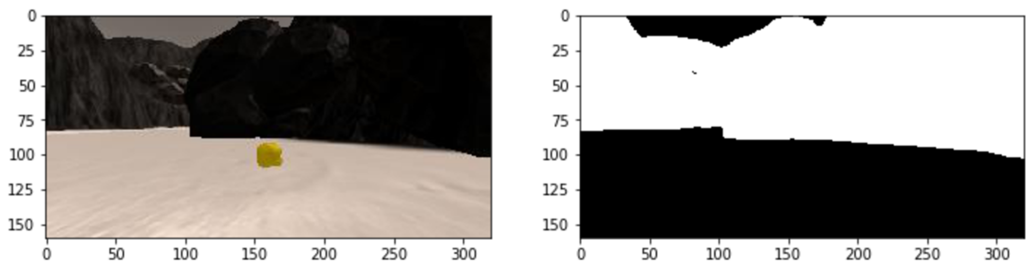
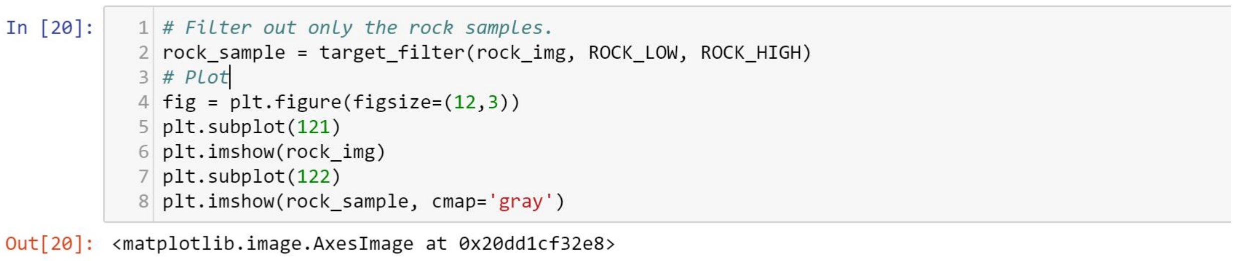


Figure 2: Filtering only the obstacles(non-navigable area).



```
In [20]: # Filter out only the rock samples.
         rock_sample = target_filter(rock_img, ROCK_LOW, ROCK_HIGH)
         # Plot
         fig = plt.figure(figsize=(12,3))
         plt.subplot(121)
         plt.imshow(rock_img)
         plt.subplot(122)
         plt.imshow(rock_sample, cmap='gray')

Out[20]: <matplotlib.image.AxesImage at 0x20dd1cf32e8>
```

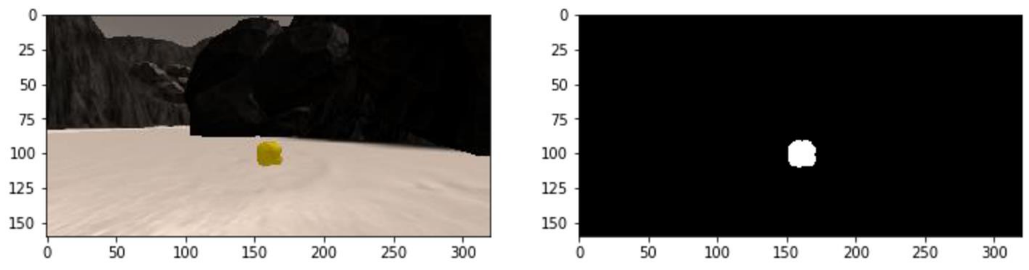
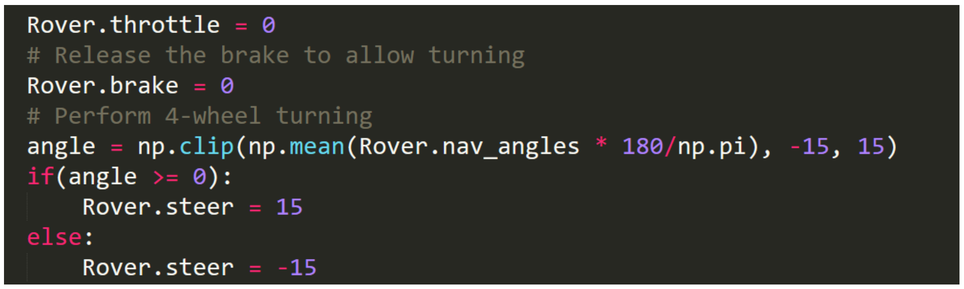


Figure 3: Filtering only the rock samples.

The function '*process_image()*' handles filtering the necessary parts of an image(using color thresholding), performing a perspective transform, converting thresholded image pixel values to rover-centric coords and also updating the world map as shown on this video.

The purpose of the function *'perception_step()'* is to process the image grabbed from the rover's camera. Firstly, perspective transform will be applied to the image using '*perspect_transform()*' function. After that, the navigable area, the obstacles and the rock samples will be filtered with the help of '*target_filter()*' function as shown in figure 1,2 and 3. After the filtration filtered image will be shown on the left side of the screen by updating '*Rover. vision_image*' instance. Then, the world coordinates will be calculated using the function '*pix_to_world()*' based on the rovers position, yaw angle and rover-centric coordinates. After that, the centric pixel positions will be converted into polar coordinates in order to update the navigation distance and navigation angle of the rover which helps to steer and control the speed of the rover. Finally, the world map will be updated if pitch and roll angles are near to zero(< 0.5).

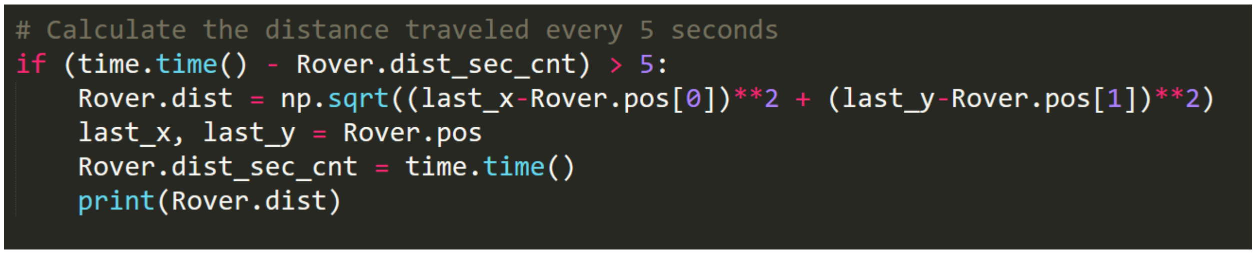
The other thing is making decisions to navigate the rover autonomously. This task is done using '*decision_step()*' function. Basically, the rover has four driving modes which are 'forward', 'stop', 'rock_nav' and 'got_stuck'. As the name of the mode implies, when the mode is 'forward' the rover tries to move forward while avoiding obstacles. And if the mode is 'stop' the rover will perform a 4-wheel turn to the left or right based on the length of the angles.



```
Rover.throttle = 0
# Release the brake to allow turning
Rover.brake = 0
# Perform 4-wheel turning
angle = np.clip(np.mean(Rover.nav_angles * 180/np.pi), -15, 15)
if(angle >= 0):
    Rover.steer = 15
else:
    Rover.steer = -15
```

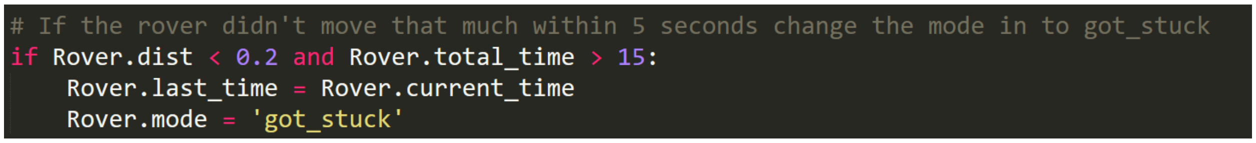
Figure 4: Steering angle decision making during 4 wheel turn.

Every 5 seconds the distance travelled is calculated inside the '*drive_rover.py*' script in order to know when the rover gets stuck



```python
# Calculate the distance traveled every 5 seconds
if (time.time() - Rover.dist_sec_cnt) > 5:
    Rover.dist = np.sqrt((last_x-Rover.pos[0])**2 + (last_y-Rover.pos[1])**2)
    last_x, last_y = Rover.pos
    Rover.dist_sec_cnt = time.time()
    print(Rover.dist)
```

Figure 5: Calculating the distance travelled every 5 seconds.

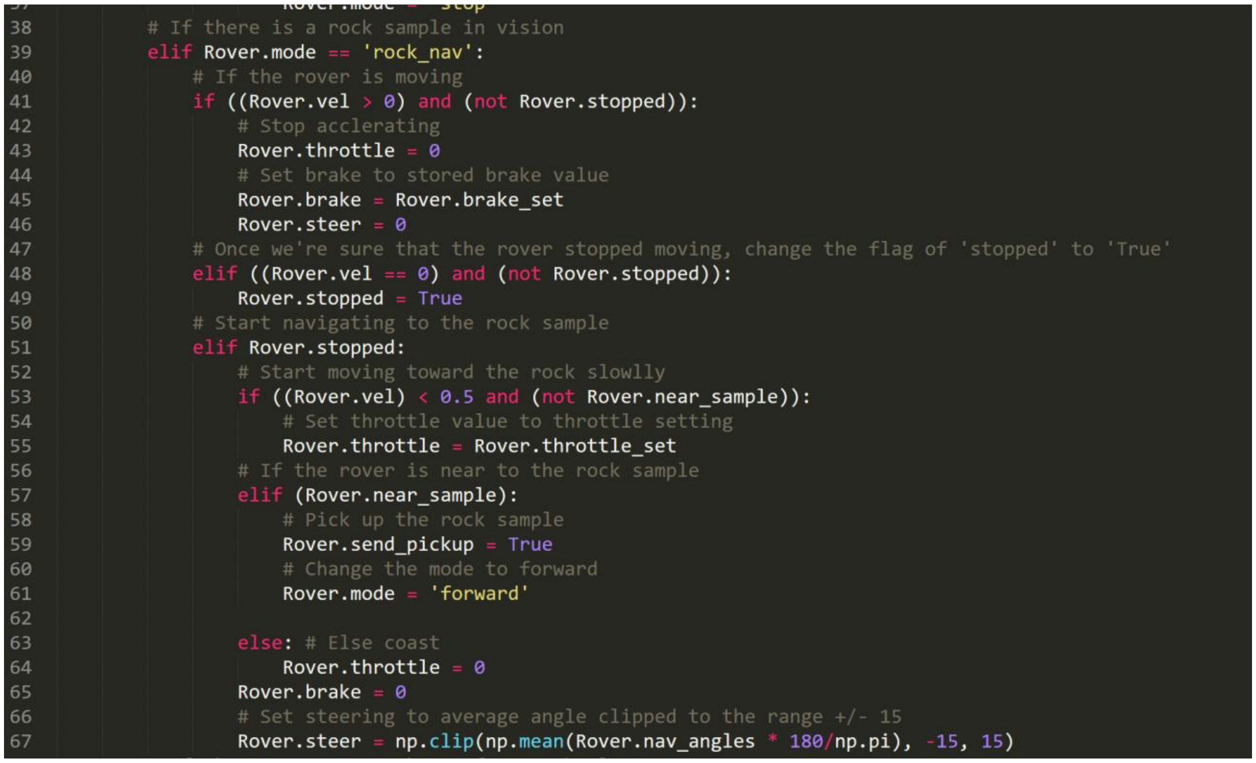


```python
# If the rover didn't move that much within 5 seconds change the mode in to got_stuck
if Rover.dist < 0.2 and Rover.total_time > 15:
    Rover.last_time = Rover.current_time
    Rover.mode = 'got_stuck'
```

Figure 6: Checking whether the rover got stuck or not.

If the rover stopped moving for 5 seconds or the total distance travelled within 5 seconds is less than 0.2 the mode will change to 'got_tuck'. This mode first makes sure the rover is not moving then it will perform 4-whell turn and finally move forward.

The other mode is 'rock_nav' which gets triggered whenever there is a rock sample in vision. This mode stops the rover at the beginning and then starts stirring the rover towards the rock sample with a maximum speed of 0.5m/s. Once the rover is sure that it's near enough to the rock sample it'll pick up the rock sample and change the mode to forward again.



```python
        # If there is a rock sample in vision
        elif Rover.mode == 'rock_nav':
            # If the rover is moving
            if ((Rover.vel > 0) and (not Rover.stopped)):
                # Stop acclerating
                Rover.throttle = 0
                # Set brake to stored brake value
                Rover.brake = Rover.brake_set
                Rover.steer = 0
            # Once we're sure that the rover stopped moving, change the flag of 'stopped' to 'True'
            elif ((Rover.vel == 0) and (not Rover.stopped)):
                Rover.stopped = True
            # Start navigating to the rock sample
            elif Rover.stopped:
                # Start moving toward the rock slowlly
                if ((Rover.vel) < 0.5 and (not Rover.near_sample)):
                    # Set throttle value to throttle setting
                    Rover.throttle = Rover.throttle_set
                # If the rover is near to the rock sample
                elif (Rover.near_sample):
                    # Pick up the rock sample
                    Rover.send_pickup = True
                    # Change the mode to forward
                    Rover.mode = 'forward'

                else: # Else coast
                    Rover.throttle = 0
                Rover.brake = 0
                # Set steering to average angle clipped to the range +/- 15
                Rover.steer = np.clip(np.mean(Rover.nav_angles * 180/np.pi), -15, 15)
```

Figure 7: Navigating the rover towards to a rock sample.

.

2. Autonomous Navigation and Mapping

By running drive_rover.py and launching the simulator in autonomous mode the rover does map 59.4% of the environment at 81.4% of fidelity. Additionally, the rover was able to collect 4 of the rock samples during this autonomous ride as shown on this video.

Note: The simulator was running with the following configuration

Screen Resolution: 1024 x 768

Graphics Quality: Fantastic