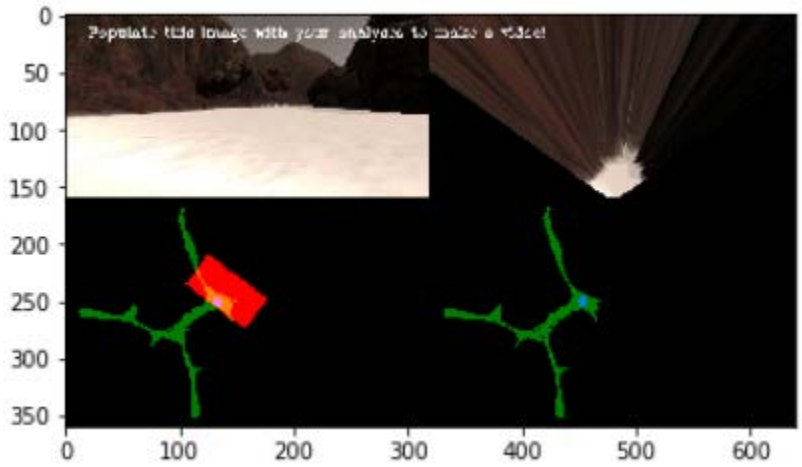
Write-up

Training / Calibration**

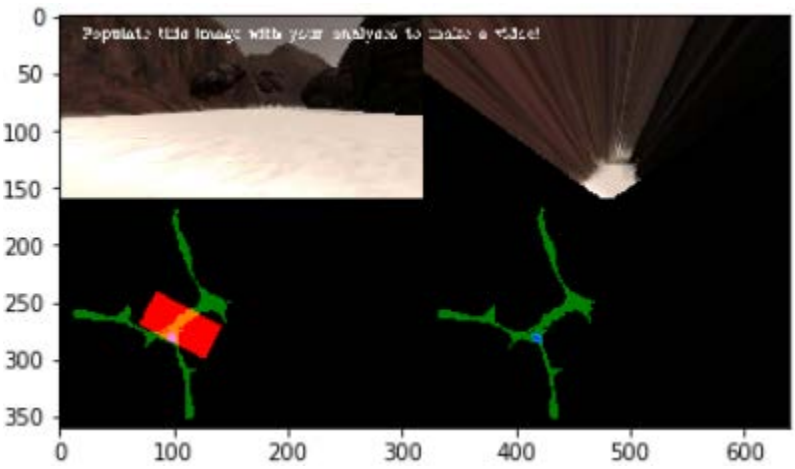
1. Run the functions provided in the notebook on test images (first with the test data provided, next on data recorded).

   Telemetry and Images were saved onto the local path in order to experiment with code using Jupyter Notebook. Path was added to path variable

   After a few unsuccessful runs, original functions were tested and video was viewable.



   After adding the recorded Data, PATH was changed and test re-run. Output was as expected.



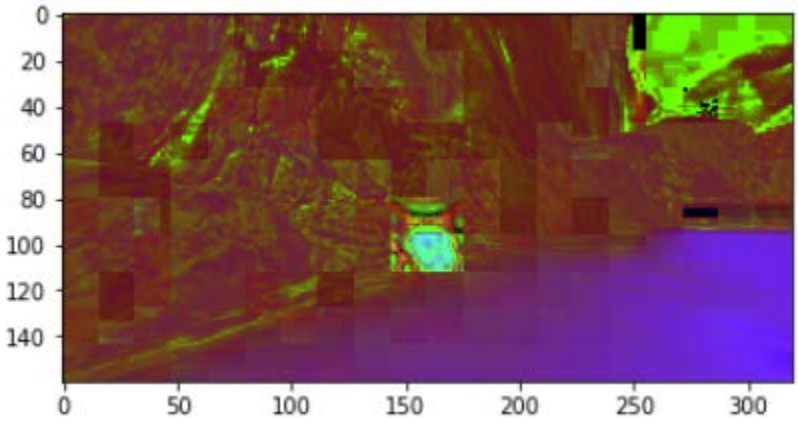
2. Add/modify functions to allow for color selection of obstacles and rock samples.

The following functions were added in order to detect a rock sample:

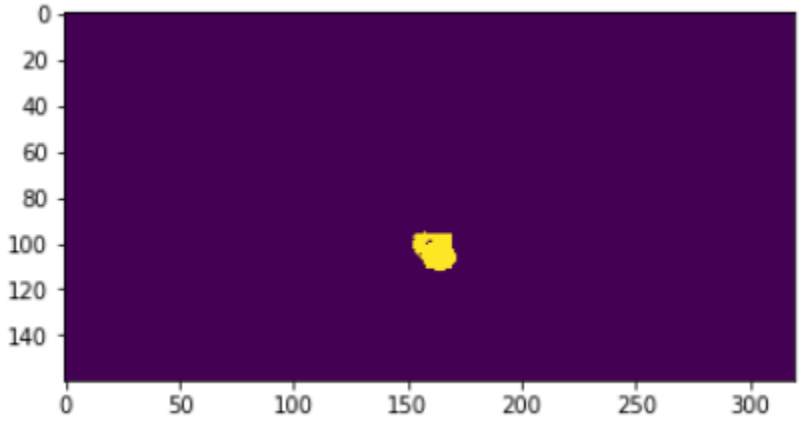
As suggested the Open CV was utilized to isolate colors

```
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
lower = np.array([100,100,0])
upper = np.array([255,255,75])
mask = cv2.inRange(img, lower, upper)
res = cv2.bitwise_or(hsv,hsv, mask = mask)
above_thresh = (res[:,:,0] > 10) \
        & (res[:,:,1] > 100) \
        & (res[:,:,2] > 100)
```

First a color threshold was created using an Open CV functions which created a range of colors



Once satisfied, a binary image was created and used to identify the samples

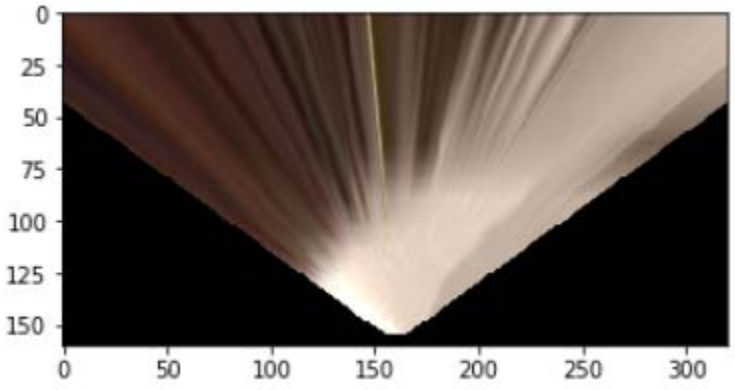


3. Populate the `process_image()` function with the appropriate analysis steps to map pixels identifying navigable terrain, obstacles and rock samples into a worldmap. Run `process_image()` on your test data using the `moviepy` functions provided to create video output of your result.

In order to achieve correct output image, the following steps were taken:

- Perspective transform was applied to the original image.

```
# provide source and destination coords
source = np.float32([[12,143 ],[118,93 ],[200 ,93 ],[305 ,143 ]])
destination = np.float32([[image.shape[1]/2-5,image.shape[0]],
                          [image.shape[1]/2-5,image.shape[0]-10],
                          [image.shape[1]/2+5, image.shape[0]-10],
                          [image.shape[1]/2+5,image.shape[0]]])
# Apply perspective transform
warped = perspect_transform(img, source, destination)
```



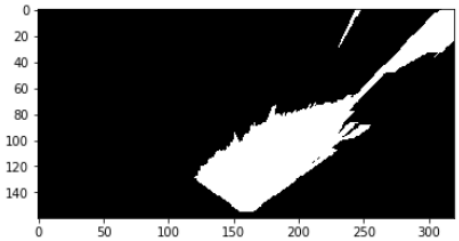
- Using color thresholds, navigable terrain, obstacles and rock samples were identified and binary images were created for further processing. In my case, I took an inverse of navigable to create the obstacle terrain.

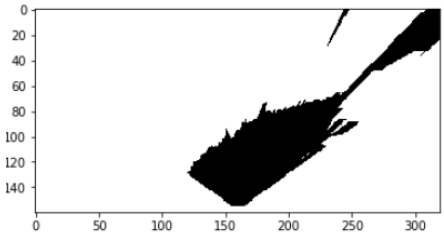
  Navigable terrain = standard code introduced in the project

  Obstacles = take inverse of Navigable

```
color_select_obst = np.invert(color_select)
```

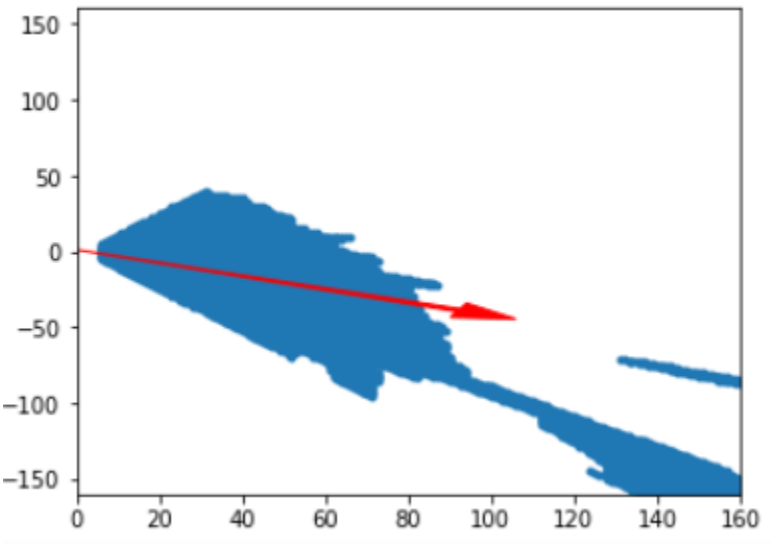
  Rock Samples = using OpenCV isolate range of Colors mentioned in step 2 of this write-up





- Using conversions, above images were then converted to rover-centric pixels

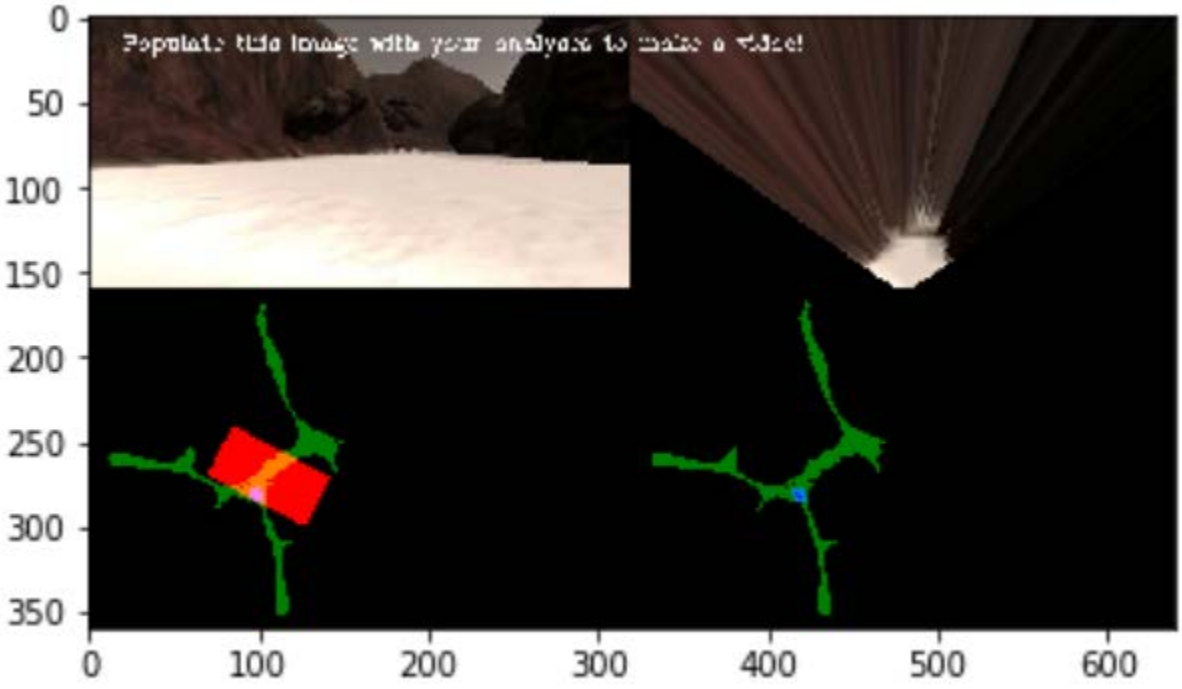
```
# converting navigable terran, obstacles and rock samples to rover-centric pixels
xpix, ypix = rover_coords(threshed)
xpixr, ypixr = rover_coords(threshrock)
xpixo, ypixo = rover_coords(threshobst)
```



- World map was updated with three overlays (navigable terrain, obstacles and rock samples)

5) Convert rover-centric pixel values to world coords

```
#convert passable area to world map
x_world, y_world = pix_to_world(xpix, ypix, data.ypos[data.count],
                    data.xpos[data.count], data.yaw[data.count],
                    data.worldmap.shape[0], 5)
#convert Rock pixels to world map
x_worldr, y_worldr = pix_to_world(xpixr, ypixr, data.ypos[data.count],
                    data.xpos[data.count], data.yaw[data.count],
                    data.worldmapr.shape[0], 5)
#convert Obstacle pixels to world map
x_worldo, y_worldo = pix_to_world(xpixo, ypixo, data.ypos[data.count],
                    data.xpos[data.count], data.yaw[data.count],
                    data.worldmapr.shape[0], 5)
# Add pixel positions to worldmap
# 6) Update worldmap (to be displayed on right side of screen)
data.worldmap[x_world, y_world,2] += 1
data.worldmap[x_worldr, y_worldr,1] += 1
data.worldmap[x_worldo, y_worldo,0] += 1
```



Autonomous Navigation / Mapping

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.

   Similar to Notebook, perception.py function was modified to include all of the data on navigable terrain, obstacles as well as the rock samples.

   Some ideas included:

   - Increasing thresholds when pitch and roll angles exceed certain value

     ```
     # adjust mean angle and turn to create erratic pattern
     Rover.steer = np.clip(np.mean(Rover.nav_angles/2 * 180/np.pi), -15, 12)
     ```

   - Adjusting mapped pixels when the rover experiences pitch/roll threshold.

     ```
     if (1.2<Rover.pitch<358.8) or (1.0<Rover.roll<358.9):
      x_world = np.zeros_like(x_world)
      y_world = np.zeros_like(y_world)
     ```

   Last solution seem to yield more fruit as every time rover hits a bump, perception is distorted and adjusting good data increased fidelity.

2. Fill in the `decision_step()` function within the `decision.py` script with conditional statements that take into consideration the outputs of the `perception_step()` in deciding how to issue throttle, brake and steering commands.

   Modifications to decision.py included the use of additional logic to ensure a rover will not continue forward when navigable terrain exists but rover is unable to move forward due to obstacle.

   ```
       # if rover hits and obstacle while angles are above threshold
   if (Rover.vel < 0.01 and Rover.vel>-0.01) and Rover.throttle > 0.4:
      Rover.throttle = -0.9
      Rover.brakes = 10
      Rover.throttle = 0
      Rover.steer = 15
      Rover.brakes=0
   ```

Additional code was added but not implemented on rock pickup because there was an issue when picking up the rock, rover seemed to freeze in place.

```
if Rover.near_sample == 0:
##       Rover.send_pickup = False
##       Rover.picking_up = 0
##
##     if not Rover.nav_angles is None:
##       print("rover mode", Rover.mode )
##       if rock_world_pos[0].any():
##          Rover.mode = 'rocks'

        # Check for Rover.mode status
##       if Rover.mode == 'rocks':
##          if Rover.vel < Rover.max_vel:
##             # Set throttle value to throttle setting
##             Rover.throttle = Rover.throttle_set
##          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_anglesr*2 * 180/np.pi)
```

Rover_nav_anglesr is the nav angle toward a rock, when rock pixels are detected, the rover was instructed to drive toward the rock. Once the rover is within the 3m mark, rover was tasked to pick up the rock by setting Rover.send_pickup = True in supporting functions.

3. Iterate on your perception and decision function until your rover does a reasonable (need to define metric) job of navigating and mapping.

   Fidelity is good but could use some improvements, currently it is in the high 80s depending on map location it traverses. The decision to have a mean angle reduced the left/right sway making the dive more efficient.

   Ability to pick and store rocks was successful however I was not able to resolve issue with the rover freezing.

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

   Given ore time, some other improvements which I will seek to develop:

   - Ability for the rover to adjust thresholds and transforms according to pitches and rolls in order to eliminate the false positives of navigable terrain on the ground truth map, this should improve the fidelity
   - Ability for the rover to detect rocks better. I was able to detect rocks ad drive toward them, but rover would miss some and drive past them. One way to resolve the issue was the mentioned Wall Crawler. Another would be to periodically stop and look around.
   - Ability for the rover to be able to navigate through the whole map. This would be accomplishable by:
     - Wall crawler technique, where the rover only hugs the right or left hand side of the wall
     - Comparison to truth map and logic which would use that notion to weight the direction of travel.