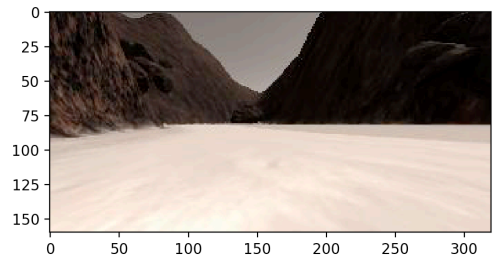
I recommend setting rover to 1024*768

the goal for this project is to have a rover drive around without any piloting autonomously
steps in the project

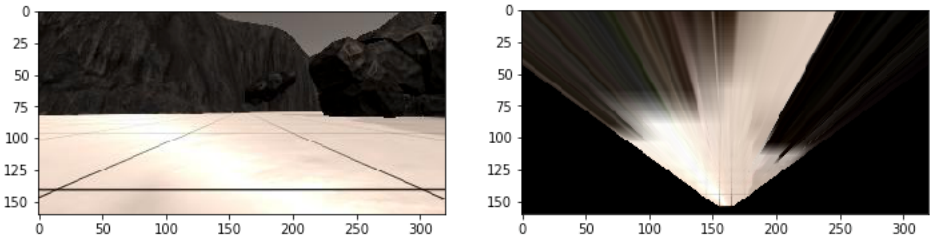
step 1) reading image from our camera
the first step is to read image from our camera. which is similar to image below
in the picture there is area where we can drive and areas where we cant drive for example the mountains and the sky.by plotting the image we could read the rgb(red,green,blue) values on the picture which we would use later to detect areas where the road exists and we can drive.



these are the x,y point for the mouse and the [red,green,blue] values for pixel where our mouse is located.

```
x=108.653     y=74.1501     [56, 41, 36]
```

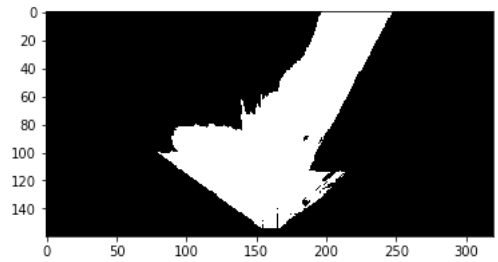
step 2) perspective transform
In this step we would use cv2 library to transform our picture to a bird view. basically, we are transforming our original view to a top down view.
using cv2.getPerspectiveTransform(src, dst) we calculate a perspective transform from four pairs of the corresponding points.
For example in the below picture we can see the corresponding points to 1 square is close to following coordinates ([14, 140], [301 ,140],[200, 96], [118, 96]) which is our src value in the function. Dst is corresponding to 10x10 square which we would see in birdview graph.
Basically perspect_transform function would transform the first picture to second.



Step 3) color threshold
In this step we would use the value for red,green,blue we found in step 1 to detects areas where we could drive vs mountains. `x=108.653  y=74.1501  [56, 41, 36]` I have a function called color_thresh() which would read the birdview image and convert it into a simpler black and white map. At this point img[:,:,0] would correspond to red pixels in our image and img[:,:,1] would correspond to green, and img[:,:,2] correspond to blue. By comparing the values to the

basic value we define in the whole image we could create a simpler map where white areas would be drivable areas and black would be mountains and areas that our car would not be able to drive. Basically in code below we decided that 160 color threshold would be for ground pixels and anything above 160 (rgb) value would be set to 1 and any pixel below would be set to 0.

```
# Threshold of RGB > 160 does a nice job of identifying ground pixels only
def color_thresh(img, rgb_thresh=(160, 160, 160)):
    # Create an array of zeros same xy size as img, but single channel
    color_select = np.zeros_like(img[:,:,0])
    # Require that each pixel be above all three threshold values in RGB
    # above_thresh will now contain a boolean array with "True"
    # where threshold was met
    above_thresh = (img[:,:,0] > rgb_thresh[0]) \
                & (img[:,:,1] > rgb_thresh[1]) \
                & (img[:,:,2] > rgb_thresh[2])
    # Index the array of zeros with the boolean array and set to 1
    color_select[above_thresh] = 1
    # Return the binary image
    return color_select
```

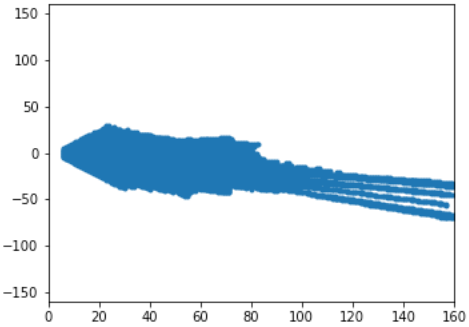


Step 4) finding out the drivable direction

In this step we would want to find out the direction that our car would need to take using the drivable area detected in the step before and also we would need to move our coordination to rover coordination so we could find the direction later.

```
def rover_coords(binary_img):
    # Identify nonzero pixels
    ypos, xpos = binary_img.nonzero()
    # Calculate pixel positions with reference to the rover position being at the
    # center bottom of the image.
    x_pixel = -(ypos - binary_img.shape[0]).astype(np.float)
    y_pixel = -(xpos - binary_img.shape[1]/2 ).astype(np.float)
    return x_pixel, y_pixel
```

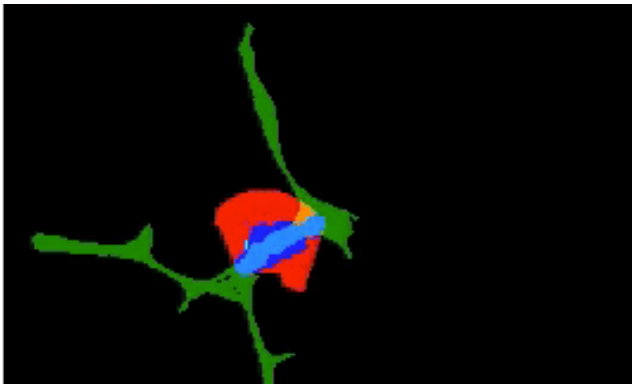
As you can see now our path is moved to point 0,0 which would basically move our origin to our car.



Step 5) map the smaller image to the world map
Now we need to detect where we are on the map which we can achieve by using pix_to_world function.

```
# Define a function to apply rotation and translation (and clipping)
# Once you define the two functions above this function should work
def pix_to_world(xpix, ypix, xpos, ypos, yaw, world_size, scale):
    # Apply rotation
    xpix_rot, ypix_rot = rotate_pix(xpix, ypix, yaw)
    # Apply translation
    xpix_tran, ypix_tran = translate_pix(xpix_rot, ypix_rot, xpos, ypos, scale)
    # Perform rotation, translation and clipping all at once
    x_pix_world = np.clip(np.int_(xpix_tran), 0, world_size - 1)
    y_pix_world = np.clip(np.int_(ypix_tran), 0, world_size - 1)
    # Return the result
    return x_pix_world, y_pix_world
```

Above function takes yaw which is our angle and using rotate and translate functions would turn and move our image to correspond to the section on the world map.



The green section is the worldmap that was provided to us, and blue section is the area where we are driving.

Step 6) to drive autonomously
We have a python script driver_rover.py which uses the flask to read values from our rover.
And using the steps above we would basically
A)Read the image
B) run the perspect_transform on the image
C) run the color_thresh function
D)