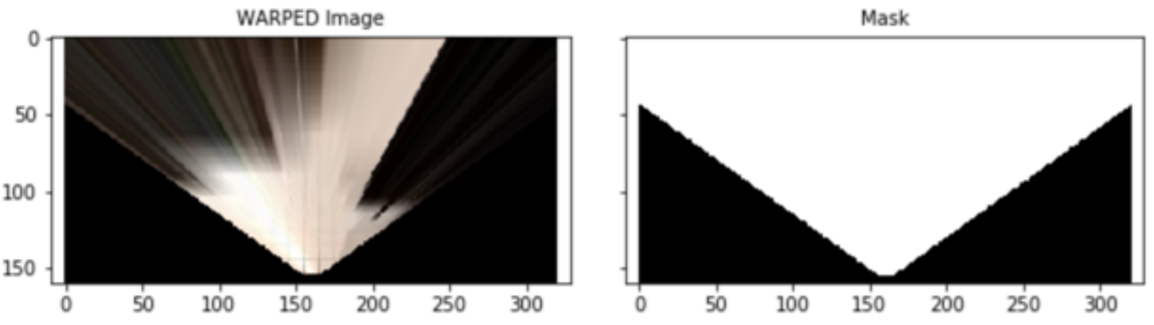
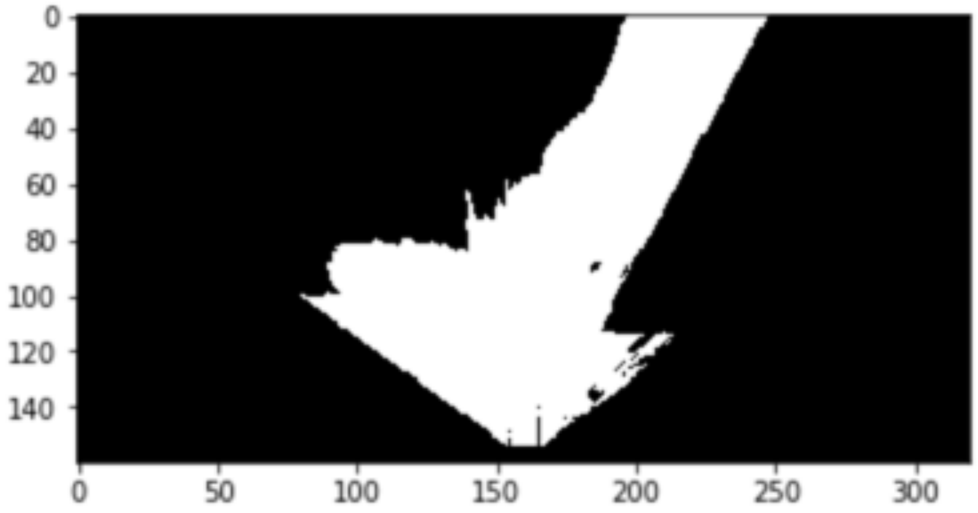
Notebook Analysis

1. **Run the functions provided in the notebook on test images (first with the test data provided, next on data you have recorded). Add/modify functions to allow for color selection of obstacles and rock samples.**

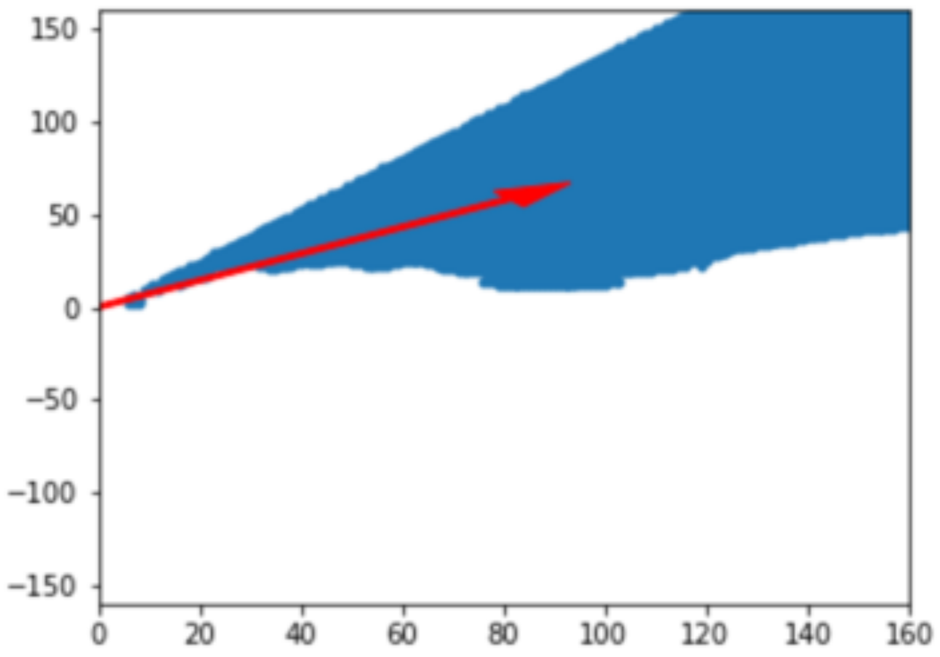
We have a camera in front of the rover. We need to transform the perspective of the images streaming from camera of the rover and shift them to one where you are looking on the world above. We have used a reference grid and transform image accordingly. Also, Front camera has a seeing angle, we mask out all the rest of the image.



After perspective transform and masking, we can apply threshold to find out navigable terrain .



After thresholding, we can convert from image coordinates to rover coordinates. We can calculate the mean of the navigable terrain. The red arrow is pointing the possible rover direction.

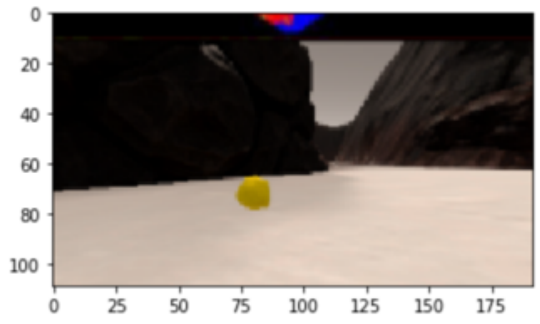


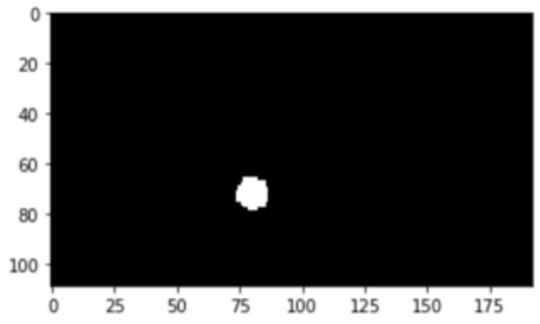
Below given method is used to identify the golden rocks

```
def find_rocks(img, levels=(110, 110, 50)):
    rockpix = ( (img[:,:,0] > levels[0]) \
                & (img[:,:,1] > levels[1]) \
                & (img[:,:,2] < levels[2]) )
    color_select = np.zeros_like(img[:,:,0])
    color_select[rockpix] = 1

    return color_select
```

Gold 's colors are different from other obstacles. We can apply a color filter to detect the gold and detect a gold if it is exists in an image as shown below.





2. 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.

Process_image implemeted and tested. You can find details in;

code/Rover_Project_Test_Notebook.ipynb

Autonomous Navigation and 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.

Prior to perception_step function, we I made some changes on percept_transform function. Front camera has a seeing angle, we mask out all the rest of the image.

Also, I added find_rocks method to find the golden rocks on the map.

In perception_step, I have applied similar steps where I had already applied in jupyter notebook process_image method.

First, we have marked the floor as grid and we have marked them as source and destination in order to make a perspective transform. Apply threshold to identify the navigable trains. Also, generate an obstacle maps.

Rover.vision_image has three channel. put thresholded map and obstacle map to channel of vision_image after multiplying them with 255.

```
x_world, y_world = pix_to_world(xpix, ypix, xpos, ypos, yaw, world_size, scale)
```

Find the coordinated of the rover in the world.

```
obs_x_world, obs_y_world = pix_to_world(obsxpix, obsypix, xpos, ypos, yaw, world_size, scale)
```

Find the obstacle coordinates in the world

```
rock_map = find_rocks(warped, levels=(100, 100, 40))
    if rock_map.any():
        rock_x, rock_y = rover_coords(rock_map)
        rock_x_world, rock_y_world = pix_to_world(rock_x, rock_y, xpos, \
                                        ypos, yaw, world_size, scale)
        data.worldmap[rock_y_world, rock_x_world, :] = 255
```

Find the golden rocks on the map and mark their position on the world.

You can find whole implementation below:

```
def process_image(img):
    warped, masked = perspect_transform(img, source, destination)
    threshed = color_thresh(warped)
    obs_map = np.absolute(np.float32(threshed) -1) # * mask
    x_pix, y_pix = rover_coords(threshed)

    world_size = data.worldmap.shape[0]
    scale = 2 * dst_size
    xpos = data.xpos[data.count]
    ypos = data.ypos[data.count]
    yaw = data.yaw[data.count]
    x_world, y_world = pix_to_world(xpix, ypix, xpos, ypos, yaw, world_size, scale)

    obsxpix, obsypix = rover_coords(obs_map)
    obs_x_world, obs_y_world = pix_to_world(obsxpix, obsypix, xpos, ypos,
                                            yaw, world_size, scale)


    data.worldmap[y_world, x_world, 2] = 255
    data.worldmap[obs_y_world, obs_x_world, 0] = 255
    nav_pix = data.worldmap[:,:,2] > 0

    data.worldmap[nav_pix, 0] = 0

    rock_map = find_rocks(warped, levels=(100, 100, 40))
    if rock_map.any():
        rock_x, rock_y = rover_coords(rock_map)
        rock_x_world, rock_y_world = pix_to_world(rock_x, rock_y, xpos, \
                                        ypos, yaw, world_size, scale)
        data.worldmap[rock_y_world, rock_x_world, :] = 255

    output_image = np.zeros((img.shape[0] + data.worldmap.shape[0], img.shape[1]*2, 3))

    output_image[0:img.shape[0], 0:img.shape[1]] = img

        # Let's create more images to add to the mosaic, first a warped image
    #warped, masked = perspect_transform(img, source, destination)
        # Add the warped image in the upper right hand corner
    output_image[0:img.shape[0], img.shape[1]:] = warped

        # Overlay worldmap with ground truth map
    map_add = cv2.addWeighted(data.worldmap, 1, data.ground_truth, 0.5, 0)
        # Flip map overlay so y-axis points upward and add to output_image
    output_image[img.shape[0]:, 0:data.worldmap.shape[1]] = np.flipud(map_add)

    cv2.putText(output_image,"Populate this image with your analyses to make a video!", (20, 20),
            cv2.FONT_HERSHEY_COMPLEX, 0.4, (255, 255, 255), 1)
    if data.count < len(data.images) - 1:
        data.count += 1 # Keep track of the index in the Databucket()

    return output_image
```

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

First, we use a very simple decision tree for decision making. We might use better approach. We might use reinforcement learning so the system might improve itself after each decision.

We detect navigable terrains and obstacles just with their colors. Also, Gold is detected only by color. In a more realistic scenario we might need more features to identify the objects.