**Notebook Analysis:**

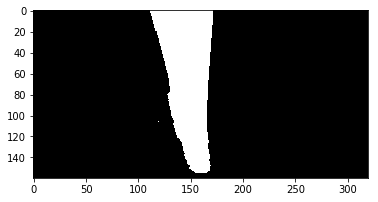
1. **Add obstacle and rock sample identification:**

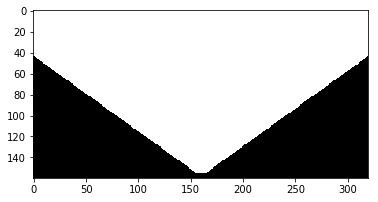
To identify the obstacle, it is quite intuitive to just inverse the binary mapping of the navigable terrain. But this will also change the setting of the area on the map that is not part of the real mapping; thus, we need a mask to hide those regions; namely, apply the following definition to get a map for the obstacles:

obs = abs(np.float32(pas)-1)\*mask

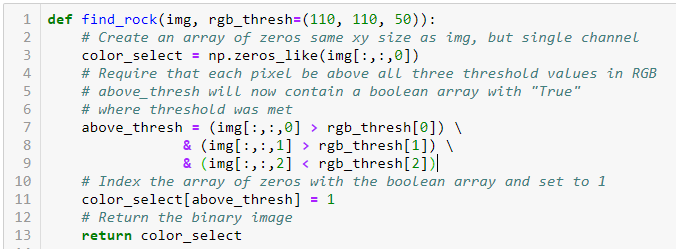
Mask can be achieved using the following definition:

mask = cv2.warpPerspective(np.ones\_like(img[:,:,0]), M, (img.shape[1], img.shape[0]))

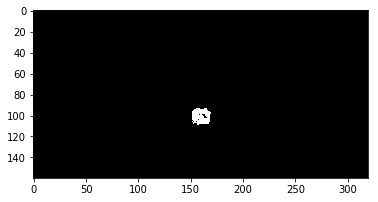
 



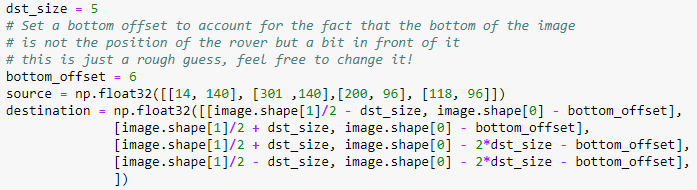
To find the yellow rock, a certain color threshold must be satisfied, and the following function is created.



The following figure shows the binary rock map.



1. **Modification on *process\_image()* function:**
2. Define source and destination points for perspective transform



1. Apply perspective transform



1. Apply color threshold to identify navigable terrain/obstacles/rock samples



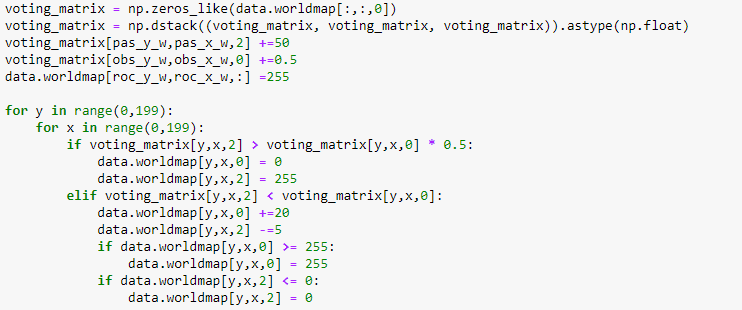
1. Convert thresholded image pixel values to rover-centric cords



1. Convert rover-centric pixel values to world cords



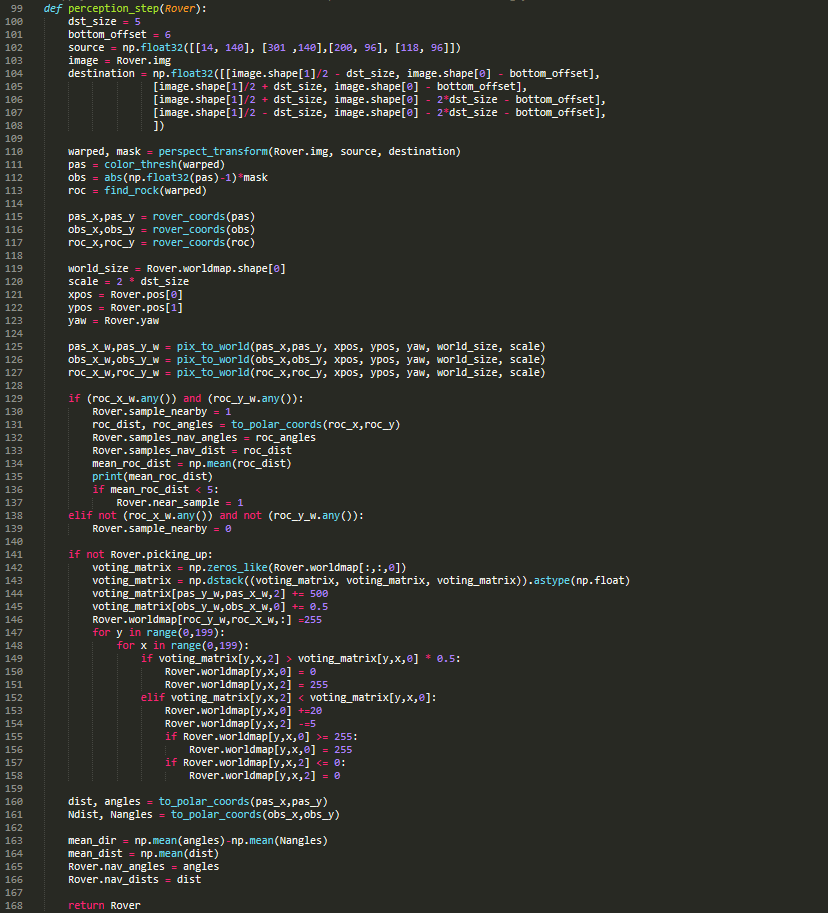
1. Update worldmap (to be displayed on right side of screen)



A voting matrix is created to decide the RGB value for each pixel. Every time the navigable terrain is mapped, the blue channels of the mapped region in voting matrix will increase by 50; and every time the obstacle is mapped, the red channels of the mapped region in voting matrix will increase by 0.5. The value of red channel and blue channel in each pixel are compared. If the blue channel has a larger value, then that pixel on the worldmap will be changed to completely blue, and vice versa. Whenever the rock is found, the mapped pixels by the rock will be changed to white (255,255,255).

**Autonomous Navigation and Mapping:**

1. Modification in *perception\_step()* function



It is the similar thing as in the process\_image() function, one of the differences is the worldmap. In the Jupiter notebook, we use data bucket, and in the simulator, it changes to the object ‘Rover’. Other than that, we need to send out the steering angle to the decision.py. To do that, both the navigable terrain and the obstacle is changed from the rover coordinate to polar coordinate, and the output is the angles and distance of terrain and obstacles. After we take the average of both, we should use the mean angle of the terrain to subtract the mean angle of the obstacle so that we can get a direction to move. Other things remain the same.

1. Modification in *decision\_step()* function

There is no change in the decision\_step function.

The following figure is the best results I got with 26 fps

