ISRAILOV SARDOR

ReadMe:

I used the simulator in the version **1024x768. Windows 10.**

Jupiter Notebook:

First of all, I added the **mask in** perspect_transform. The mask allows us to know where are real obstacles after the perspective transform. So in the cv2.warpPerspective, instead of the image pixels we use the np.ones_like function to obtain the black/white picture in the end. The black area means the limitation of our perspective_transform(camera).

Afterwards, I added the **find_rocks** function which basically uses color thresholding to find rocks. The color sample choosen is levels=(110,110,48), and we do an if statement to see if the red channel is greater than 110, blue channel is less than 48.

Afterwards, we use the pandas library to import the robot_log.csv and initialize the databucket, we is basically the class with data.

After I completed the process_image function which receives the image as a parameter and does the analyses of the image to map the world_map correctly.

process_image(img)

First of all, we are doing the perspective transform with the image source and destination points. We implement the color thresholholding above (160, 160, 160) to know where is the navigable terrain. After, we use **obs_map=np.absolute(np.float32(threshed)-1)*mask,** to obtain the location of pixels everywhere where the thresholded image was zero in the area of ones of the mask, which corresponds to the pixels where are the obstacles are. Then, we convert them to obs_x_world,obs_y_world, and navigable terrain pixels to world_coordinates(pix_to_world()) after finding the rover coordinates(rover_coords(threshed)). Whenever we have a navigable terrain we put a blue color on the map, and whenever we have obstacles we have the red color. So there are times, where we find for the same pixels that it's navigable terrain and the obstacle.

The basic solution is to tell that wherever we see the pixels of the navigable terrain, we put the red channel of the image to zero.

Suggestion to improvement:

In Jupiter processimage() we can implement the algorithm with counter for navigable terrain and obstacles outside of the function since we want it ti be initialized only once:

cNavigable=np.zeros((rock_img.shape[0],rock_img.shape[1])).astype(np.int)

cObstacle=np.zeros((rock_img.shape[0],rock_img.shape[1])).astype(np.int)

Inside process_image: inside the

cNavigable[y_world,x_world]+=1; #wherever we have navigable

cObstacle[obs_y_world,obs_x_world]+=1; #wherever we have an obstacle

ISRAILOV SARDOR

```
for i in range (1,rock_img.shape[0]):
    for j in range (1,rock_img.shape[1]):
        if(cNavigable[i,j]>cObstacle[i,j]):
            data.worldmap[i,j,0]=0
            data.worldmap[i,j,2]=255
```

ISRAILOV SARDOR

Autonomous Navigation and Mapping

`perception_step()`

The basic functionality is the same to the one of process_image. Nevertheless, we use the Rover class that have some other attributes. We use the vision_image (Image output from perception step that contains our analysis) instead:

Rover.vision_image[:,:,2]=threshed*255

Rover.vision_image[:,:,0]=obs_map*255

We use to_polar_coords to find navigable angles of the rover.

Red for obstacles, and blue for navigable terrain.

In rock_map function we find center of the rock, and color it.

decision_step(Rover):

I changed the fist line which basically checks if I have a vision data to work with:

if (Rover.nav_angles is not None) & (Rover.vision_image is not None):

the functionality:

there are 2 modes: forward or backward. If the navigable terrain angle is large enough(>50) and the velocity is below max than we put it on max, else we steer; if the navigable terrain isn't large enough, we stop the rover.

In the stop mode, if we are moving, start breaking and put the throttle to 0. We steer by (-15) every time we see that our navigable terrain isn't large enough, and when the nav_angle is greater than threshold(500) we continue to accelerate.

if (Rover.nav_angles is not None) & (Rover.vision_image is not None):

Annexes:

def perception_step(Rover):

```python
dst_size = 5
# Set a bottom offset to account for the fact that the bottom of the image
# is not the position of the rover but a bit in front of it
bottom_offset = 6
image=Rover.img
source = np.float32([[14, 140], [301 ,140],[200, 96], [118, 96]])
destination = np.float32([[image.shape[1]/2 - dst_size, image.shape[0] - bottom_offset],
                  [image.shape[1]/2 + dst_size, image.shape[0] - bottom_offset],
                  [image.shape[1]/2 + dst_size, image.shape[0] - 2*dst_size - bottom_offset],
                  [image.shape[1]/2 - dst_size, image.shape[0] - 2*dst_size - bottom_offset],
                  ])


# 3) Apply color threshold to identify navigable terrain/obstacles/rock samples

warped, mask = perspect_transform(Rover.img, source, destination)
threshed=color_thresh(warped)
obs_map=np.absolute(np.float32(threshed)-1)*mask#make obstacle map
xpix,ypix=rover_coords(threshed)

Rover.vision_image[:,:,2]=threshed*255
Rover.vision_image[:,:,0]=obs_map*255

world_size=Rover.worldmap.shape[0]
scale=2*dst_size


x_world,y_world=pix_to_world(xpix,ypix,Rover.pos[0],Rover.pos[1],Rover.yaw,world_size,scale)
obsxpix, obsypix=rover_coords(obs_map)
obs_x_world,obs_y_world= pix_to_world(xpix,ypix,Rover.pos[0],Rover.pos[1],Rover.yaw,world_size,scale)


6) Update worldmap (to be displayed on right side of screen)
Rover.worldmap[y_world,x_world,2]+=10
Rover.worldmap[obs_y_world,obs_x_world,0]+=1

dist,angles=to_polar_coords(xpix,ypix)
Rover.nav_angles=angles


rock_map=find_rocks(warped, levels=(110,110, 50))
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, Rover.pos[0],Rover.pos[1],Rover.yaw, world_size, scale)

    rock_dist,rock_ang=to_polar_coords(rock_x,rock_y)
    rock_idx=np.argmin(rock_dist)
    rock_xcen=rock_x_world[rock_idx]
    rock_ycen=rock_y_world[rock_idx]

    Rover.worldmap[rock_ycen,rock_xcen,1]=255;
    Rover.vision_image[:,:,1]=rock_map*255;
    # Add the warped image in the upper right hand corner
else:
    Rover.vision_image[:,:,1]=0

return Rover
```