CSE483 – Computer Vision

NASA Mars Sample & Return Rover

Phase One

Omar Mohamed Ibrahim Alsayed 19p7813

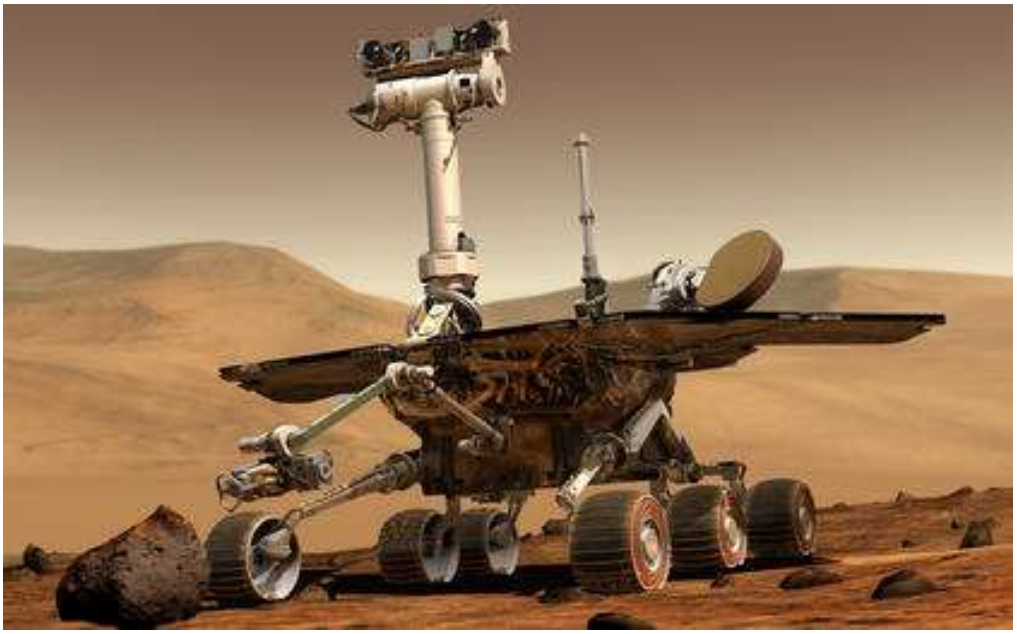
Mohamed Hatem Zakaraia Elafifi 19p7582

Karen Alber Farid Hanna 19p8948

Muhammad Amr Fathy Muhammad Nasef 20p5552

Table of Contents

Overview



Since 15th of July 1965 when Mariner 4 first flow nearby Mars, Human became interested in studying the planet. This is why one of our first objectives is to study the geology of the planet and map the planet. By the end of 1999 a mission called MSR where a rover to be sent to Mars to collect sampled to be studied by NASA, but unfortunately the mission was canceled, hence our main target is to build a MARS MSR-like program.

Github link: https://github.com/Omar-Mohamed-Ibrahim-Alsayed/MarsRoverProject/tree/master

Objectives

Using computer vision our main objective is

1. Mapping at least 40% of the environment with 60% fidelity
2. The map will be repainted to distinguish various elements such as (navigable terrain, obstacles, and rock samples)
3. Locate at least one rock (out of 6 rock samples) to be sent back home to Mother Earth
4. Building a Debugging Mode where each step in the pipeline in illustrated with vehicle operations

Installation

1. Create a Directory with command mkdir asu

2. Traverse to the directory using the command cd asu

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# cd asu
```

3. Clone the repository using the command git clone https://github.com/Omar-Mohamed-Ibrahim-Alsayed/MarsRoverProject.git

```
┌──(root㉿kali)-[/home/kali/Desktop/asu]
└─# git clone https://github.com/Omar-Mohamed-Ibrahim-Alsayed/MarsRoverProjec
t.git
Cloning into 'MarsRoverProject'...
remote: Enumerating objects: 374, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 374 (delta 0), reused 8 (delta 0), pack-reused 366
Receiving objects: 100% (374/374), 76.26 MiB | 3.12 MiB/s, done.
Resolving deltas: 100% (5/5), done.
```

4. Create a conda environment using the command conda create --name nasef --file cv1.txt

```
(base) ┌──(root㉿kali)-[/home/kali/Desktop/MarsRoverProject-master/Repo]
└─# conda create --name nasef --file cv1.txt

Downloading and Extracting Packages
blas-1.0             | ############################################################################### | 100%
ca-certificates-2022 | ############################################################################### | 100%
tzdata-2022f         | ############################################################################### | 100%
c-ares-1.18.1        | ############################################################################### | 100%
charls-2.2.0         | ############################################################################### | 100%
eigen-3.3.7          | ############################################################################### | 100%
```

5. Install python-socketio version 4.6.1 using the command conda install python-socketio=4.6.1

```
(base) ┌──(root㉿kali)-[/home/kali/Desktop/MarsRoverProject-master/Repo]
└─# conda install python-socketio=4.6.1
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/kali/Downloads/ENTER

  added / updated specs:
    - python-socketio=4.6.1


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-22.11.1              |   py39h06a4308_3         938 KB
    python-engineio-4.1.0      |     pyhd3eb1b0_0          36 KB
    ruamel.yaml-0.17.21        |   py39h5eee18b_0         178 KB
    ruamel.yaml.clib-0.2.6     |   py39h5eee18b_1         140 KB
    ------------------------------------------------------------
                                           Total:         1.3 MB

The following NEW packages will be INSTALLED:

  python-engineio    pkgs/main/noarch::python-engineio-4.1.0-pyhd3eb1b0_0 None
  python-socketio    pkgs/main/noarch::python-socketio-4.6.1-pyhd3eb1b0_0 None
  ruamel.yaml        pkgs/main/linux-64::ruamel.yaml-0.17.21-py39h5eee18b_0 None
  ruamel.yaml.clib   pkgs/main/linux-64::ruamel.yaml.clib-0.2.6-py39h5eee18b_1 None

The following packages will be UPDATED:

  conda                              22.9.0-py39h06a4308_0 --> 22.11.1-py39h06a4308_3 None


Proceed ([y]/n)? y


Downloading and Extracting Packages
python-engineio-4.1. | 36 KB     | ######################################################################## | 100%
ruamel.yaml-0.17.21  | 178 KB    | ######################################################################## | 100%
ruamel.yaml.clib-0.2 | 140 KB    | ######################################################################## | 100%
conda-22.11.1        | 938 KB    | ######################################################################## | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Retrieving notices: ...working... done
```

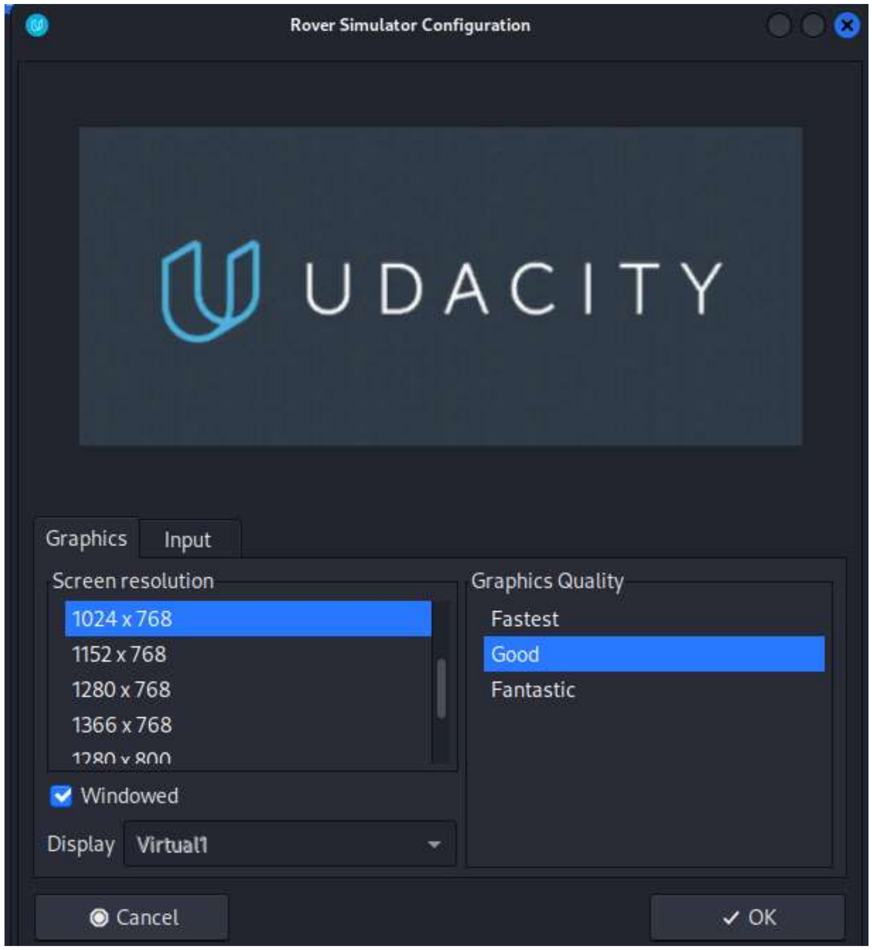
6. Active the environment using the command source activate nasef

```
(base) ┌──(root㉿kali)-[/home/…/Desktop/MarsRoverProject-master/Repo/code]
└─# source activate nasef

(nasef) ┌──(root㉿kali)-[/home/…/Desktop/MarsRoverProject-master/Repo/code]
└─# 
```

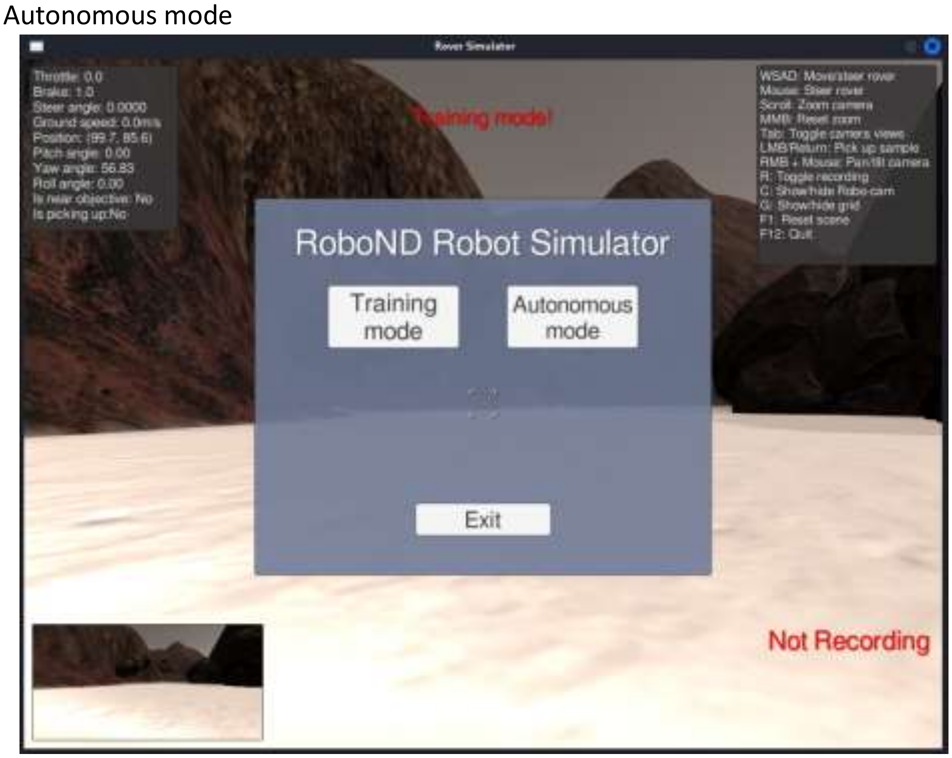
7. Run the environment using the command python driver_rover.py

```
(nasef) ┌──(root㉿kali)-[/home/…/Desktop/MarsRoverProject-master/Repo/code]
└─# python drive_rover.py
/home/kali/Desktop/MarsRoverProject-master/Repo/code/drive_rover.py:36: DeprecationWarning: `np.float` is a depreca
ted alias for the builtin `float`. To silence this warning, use `float` by itself. Doing this will not modify any b
ehavior and is safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprec
ations
  ground_truth_3d = np.dstack((ground_truth*0, ground_truth*255, ground_truth*0)).astype(np.float)
/home/kali/Desktop/MarsRoverProject-master/Repo/code/drive_rover.py:68: DeprecationWarning: `np.float` is a depreca
ted alias for the builtin `float`. To silence this warning, use `float` by itself. Doing this will not modify any b
ehavior and is safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprec
ations
  self.vision_image = np.zeros((160, 320, 3), dtype=np.float)
/home/kali/Desktop/MarsRoverProject-master/Repo/code/drive_rover.py:72: DeprecationWarning: `np.float` is a depreca
ted alias for the builtin `float`. To silence this warning, use `float` by itself. Doing this will not modify any b
ehavior and is safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprec
ations
  self.worldmap = np.zeros((200, 200, 3), dtype=np.float)
NOT recording this run ...
(26351) wsgi starting up on http://0.0.0.0:4567
```

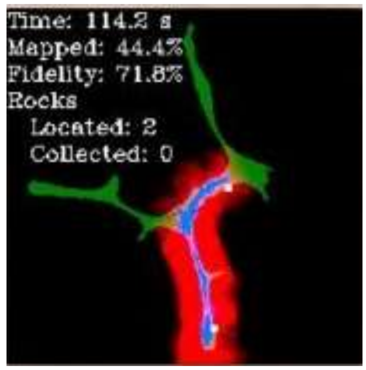
8. Run Roversim then click ok



9. Choose Autonomous mode



Map



1. Reds are non-navigable terrain (obstacles)
2. Blues are navigable terrain (path)
3. Whites are rocks to be collected

Results

During the simulations we got the following readings which fulfill the objectives of phase One

Iterations	40%	time	50%	time	60%	time	success
1	76%	59	64%	150	60%	160	succeeded
2	stuck	stuck	stuck	stuck	stuck	stuck	stuck
3	78%	53	63%	146	60%	161	succeeded
4	74%	56	64%	145	stuck	stuck	succeeded
5	67%	54	62%	147	62%	160	succeeded
6	71%	52	53%	173	54%	185	succeeded
7	70%	55	65%	144	63%	166	succeeded
8	77%	54	62%	145	61%	163	succeeded
9	77%	110	67%	120	66%	140	succeeded
10	77%	110	70%	117	66%	139	succeeded

Step By Step Intuition for the Pipeline

Our project follows the following step by step to achieve the desired outcomes. These steps are not a steps that gets executed once then the program ends. These steps is a closed-loop feedback system or simply we can call it a pipeline where these steps are executed over and over again. The output of the system in iteration n is the input for the system in iteration n+1.

1. As the rover navigates the perspective will be different, for example things may be titled to the right or to the left. This is why we created a perspective transform to obtain a flat image in a top-down view we can work on and then processed it.
2. After obtaining a very good view using the perspective transform, we need to differentiate things we can walk on (navigable terrain), things that prevents us from walking (obstacles), and things we want to gather (rocks). This why we used the color thresholding to differentiate between all of them, where ground pixels are detected, sky is just the invert of ground pixels, and rocks is defined within a range.
3. For a rover to navigate, it must have some kind of a navigation criterion. This why we first converted the image coordinates to rover coordinates then converted again to polar coordinates which will be used as a parameter for the decision tree in the autonomous mode
4. Then a set of geometric transformations such as rotations, scaling and clipping is applied to output an image
5. The identifiable objects image, the decisions and the map will be combined as an input to the simulator which will in turn take a new decision to make us go again to point 1 in a closed-feedback loop
6. Then a debugging mode is implemented where each step of the pipeline is illustrated.

Methods

color_thresh

Description

This function is used to detect and identify pixels that exceeds a given threshold.

Code

```
def color_thresh(img, rgb_thresh=(160, 160, 160)):
    color_select = np.zeros_like(img[:,:,0])
    above_thresh = (img[:,:,0] > rgb_thresh[0]) \
                & (img[:,:,1] > rgb_thresh[1]) \
                & (img[:,:,2] > rgb_thresh[2])
    color_select[above_thresh] = 1
    return color_select
```

The code works as follows

1. An image and a desired threshold is given to the function.
2. It uses numpy to create a 0 initialized array where the detected element will be marked on
3. Then it will iterate on the three channels of the given image comparing it to the threshold
4. If the given threshold is met, then it will be marked on the color_select which will be returned later

rover_coords

Description

Both the generated image and actual rover coordinates are different, this why we need to translate the image coordinates to the rover coordinates using this function

Code

```
def rover_coords(binary_img):
    ypos, xpos = binary_img.nonzero()
    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
```

The code works as follows

1. The function takes the binary image then extract the nonzero pixel from it
2. Subtracting the y coordinates of the image from the rovers y position then invert it
3. Subtracting half of the x coordinates of the image from the rovers x position then invert it
4. Then return the float values of both x and y coordinates.

to_polar_coords

Description

Having a cartesian coordinate may be useful for a specific case but having a polar coordinate will help a lot. This why this function converts from cartesian coordinates to polar coordinates.

Code

```
def to_polar_coords(x_pixel, y_pixel):
    dist = np.sqrt(x_pixel**2 + y_pixel**2)
    angles = np.arctan2(y_pixel, x_pixel)
    return dist, angles
```

The code works as follows

1. It takes x and y coordinates
2. It calculates the distance by taking the square root of the squared coordinates
3. It takes the arctan of the coordinates to calculate the angles
4. Then it returns both destinations and angles

rotate_pix

Description

The rotate_pix function is used to map the rover space to the world space

Code

```
def rotate_pix(xpix, ypix, yaw):
    yaw_rad = yaw * np.pi / 180
    xpix_rotated = (xpix * np.cos(yaw_rad)) - (ypix *
np.sin(yaw_rad))
    ypix_rotated = (xpix * np.sin(yaw_rad)) + (ypix *
np.cos(yaw_rad))
    return xpix_rotated, ypix_rotated
```

The code works as follows

1. The function takes x,y and yaw axis as a parameters
2. Then converts the yaw into radiant
3. Then rotates the x and y coordinates by using the converted yaw radiant
4. Then it returns the rotated coordinates

translate_pix

Description

The function applies both translation and scaling on any given coordinates

Code

```
def translate_pix(xpix_rot, ypix_rot, xpos, ypos,scale):
    xpix_translated = (xpix_rot / scale) + xpos
    ypix_translated = (ypix_rot / scale) + ypos
    return xpix_translated, ypix_translated
```

The code works as follows

1. X and y coordinates, the amount of translation in x and y coordinates, and the scaling factor is a parameters of the function
2. The scaling is a division/multiplication operation, and the translation is plus/minus operations.
3. Then it returns the translated and scaled coordinates.

pix_to_world

Description

The pix_to_world function is a function that applies different geometric transformations to output the final world map image. It also ties the previous functions together

Code

```
def pix_to_world(xpix, ypix, xpos, ypos, yaw,world_size, scale):
    xpix_rot, ypix_rot = rotate_pix(xpix, ypix, yaw)
    xpix_tran, ypix_tran = translate_pix(xpix_rot, ypix_rot, xpos, ypos, scale)
    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 x_pix_world, y_pix_world
```

The code works as follows

1. First it rotates the x and y coordinates
2. Then it translates and scale them using the given value
3. Then at the end it clip the unwanted values to only have the wanted values
4. It returns the final x and y coordinates for the map image

perspect_transform

Description

As we have mentioned in the step-by-step guide. A perspective transformed image have to be generated to have a flattened top-down view of the field of view to differentiate between navigable and non-navigable terrain. This function creates the perepective transformation

Code

```
def perspect_transform(img, src, dst):

    M = cv2.getPerspectiveTransform(src, dst)
    warped = cv2.warpPerspective(img, M, (img.shape[1],
img.shape[0]))
    mask =
cv2.warpPerspective(np.ones_like(img[:,:,0]),M,
(img.shape[1], img.shape[0]))

    return warped , mask
```

The code works as follows

1. The function takes the image, coordinates in the source image, and coordinates in the output image
2. Then it generates a transformation matrix and store it at M
3. Then it uses the transformation matrix with the image to apply the perspective transformation to the image
4. Then we create a mask which will have a values of 1s for navigable pixels and 0s for non-navigable pixels
5. Then return both the mask and the transformed image

find_rocks

Description

The find_rocks function is similar to the color_tresh function as it uses thresholding to identify the rocks but with minor modifications

Code

```
def find_rocks(img, thresh = (110,110,50)):
    rock_pixels = ((img[:,:,0]>thresh[0])\
                  &(img[:,:,1]>thresh[1])\
                  &(img[:,:,2]<thresh[2]))
    colored_pixels = np.zeros_like(img[:,:,0])
    colored_pixels[rock_pixels] = 1
    return colored_pixels
```

The code works as follows

1. It takes the image and the wanted threshold
2. It compares the image three channels to the given threshold and store the values to rock_pixels
3. A zero array is generated
4. The identified pixels that met the conditions will be used as an index for the zero array generated and every found rock will be equal to 1
5. The zero array is returned

perception_step

Description

This function is the function that tie the previous functions together to create a better perception of the world and achieve the objectives we want

Code

```
def perception_step(Rover):
    dst_size = 10
    bottom_offset = 5
    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],
                    ])
    warped, mask = perspect_transform(Rover.img, source, destination)
    threshed = color_thresh(warped)
    obs_map = np.absolute(np.float32(threshed) - 1)*mask
    Rover.vision_image[:,:,2] = threshed *255
    Rover.vision_image[:,:,0] = obs_map *255
    xpix, ypix = rover_coords(threshed)
    dist, angles = to_polar_coords(xpix, ypix)
    mean_dir = np.mean(angles)
    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(obsxpix,obsypix,Rover.pos[0],Rover.pos[1],R
over.yaw,world_size,scale)
    if(Rover.pitch< 1.6):
        if(Rover.roll<5):
            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,(110,110,50))
    if rock_map.any():
        rock_xpix , rock_ypix = rover_coords(rock_map)

        rock_xpix_world , rock_ypix_world
=  pix_to_world(rock_xpix, rock_ypix, Rover.pos[0],
Rover.pos[1], Rover.yaw, world_size, scale)

        rock_dist,rock_ang =
to_polar_coords(rock_xpix,rock_ypix)
        rock_idx = np.argmin(rock_dist)
        rock_xcen = rock_xpix_world[rock_idx]
        rock_ycen = rock_ypix_world[rock_idx]
        Rover.worldmap[rock_ycen, rock_xcen,1] = 255
        Rover.vision_image[:,:,1] = rock_map *255
    else:
        Rover.vision_image[:,:,1]=0
    image2 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    cv2.imshow('camera',image2)
    warped2 = cv2.cvtColor(warped, cv2.COLOR_BGR2RGB)
```

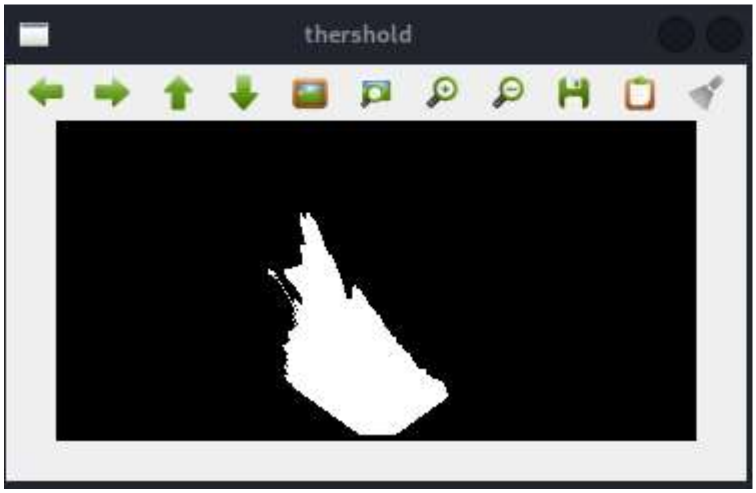
```
    cv2.imshow('precpective transform',warped2)
    mask2 = mask*255
    cv2.imshow('precpective mask',mask2)
    cv2.imshow('obstacle',obs_map)
    threshed2 = threshed * 255
    cv2.imshow('thershold',threshed2)
    rock_map2 = rock_map*255
    cv2.imshow('rock',rock_map2)
    arrow_length=100
    x_arrow = arrow_length * np.cos(mean_dir)
    y_arrow = arrow_length * np.sin(mean_dir)
    if( (x_arrow == x_arrow) and  (y_arrow == y_arrow)
):
        color = (0, 0, 255)
        thickness = 2
        view = image = cv2.rotate(threshed2,
cv2.ROTATE_90_COUNTERCLOCKWISE)
        start_point =
(int(view.shape[1]),int(view.shape[0]/2))
        end_point=(int(x_arrow),int(y_arrow)+int(view.sh
ape[0]/2))
        direction = cv2.arrowedLine(view,start_point,
end_point, color,thickness)
        cv2.imshow('Direction', direction)
    cv2.waitKey(5)

    return Rover
```
The code works as follows

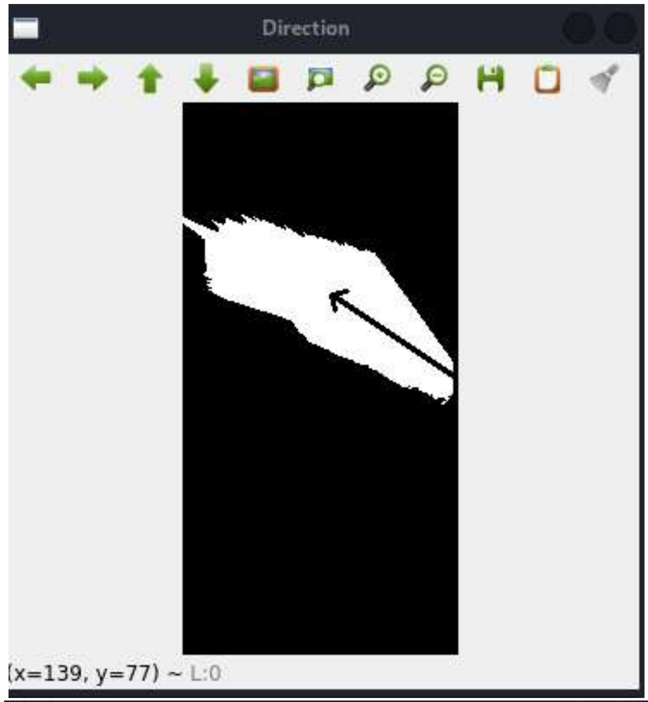
1. Define the dst_size which is the destination size
2. Define bottom_offset which is just an offset the gives us a buffer by moving the position 6 units to the front
3. Define the image, source, and destinations points for the perspective transformation
4. It will generate a perspective transformation and return the transformed image and the mask

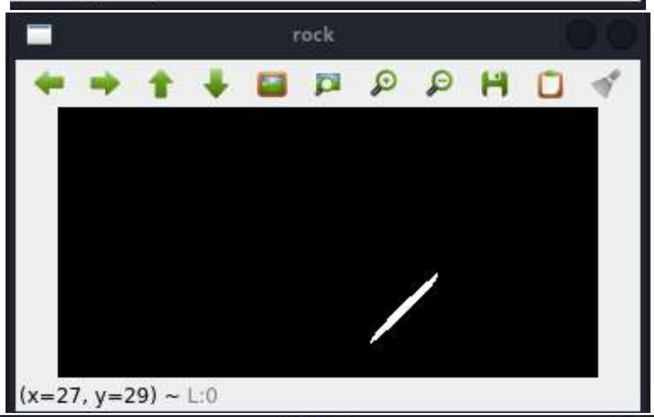
5. Then it will apply color thresholding to identify navigable terrain, non-navigable terrain and rocks
6. Then it will generate obstacle maps in obs_map by multiplying the mask with threshold -1 so we get only the things in the field of view
7. Using the threshold and the obstacle map we modify the vision image for both the obstacles and the navigable terrain
8. Then It will convert the image coordinates to rover coordinates
9. It will define variables related to the scale and world size which will be used for the next step
10. Then it will convert the rover coordinates to world coordinates for both the obstacles and the walkable world
11. An updated rover worldmap is generated to be on the right where steps 12 and 13 happens and it prevents the rotation unless a certain pitch and roll is satisfied
12. The navigable terrain becomes blue given the world coordinates
13. The obstacles become red given the obstacles coordinates
14. Then the world is converted from rover coordinates to polar coordinates
15. The find_rocks function is used to find the rocks
16. If rocks are identified, then it will convert the rock position to rover coordinates then to world coordinates then to polar coordinate
17. Then color the rock with white on the map
18. After that come the debugging part where we first going to show the camera, precpective transform, the mask, the obstacle, threshold, and the rocks for debugging purposes
19. To build the direction vector we had to get the values of x_ arrow and y_ arrow using the mean dir of xpix and ypix
20. Then check if the value is not null (NaN) to avoid being stuck when facing a wall
21. If it’s not equal null then it will rotate the threshold image by 90 degrees
22. Sepcifying the starting point which is x= width of threshold, y= half the length of the threshold
23. An Arrow is Drawn from the starting point to the endpoint of the threshold image
24. Direction window is shown

Appendix A

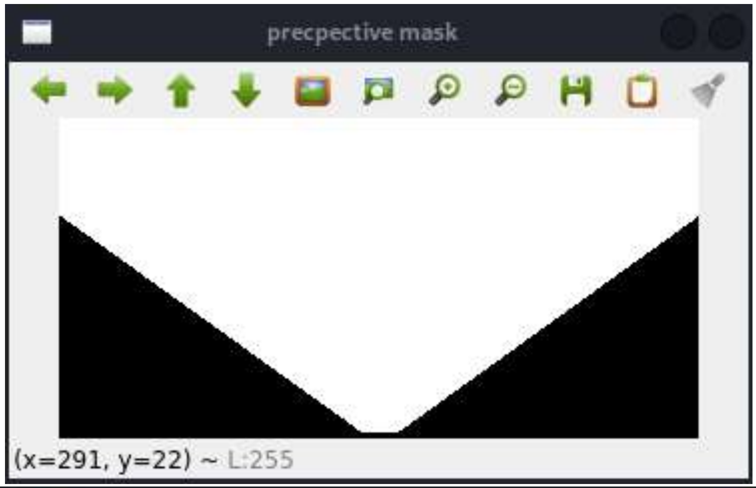
Appendix A is a collection of a screenshoots f




Direction
(x=139, y=77) ~ L:0


rock
(x=27, y=29) ~ L:0


obstacle


precpective mask
(x=291, y=22) ~ L:255


camera