
# Project Report

## NASA Mars Sample & Return Rover (MARS MSR)

Team Members:

| Name | ID |
|---|---|
| **Zainb Maged Arafa Zahran** | **1901420** |
| **Reem Hassan Ahmed** | **1901677** |
| **Rana Mohamad Ahmad Abdel Salam** | **1901208** |
| **Nada Waleed Mohammed Ebed** | **1901544** |
| **Sara Ramadan Mohamed** | **1901309** |

# Phase (1) - Basic Operation:

**Perception.py file :**

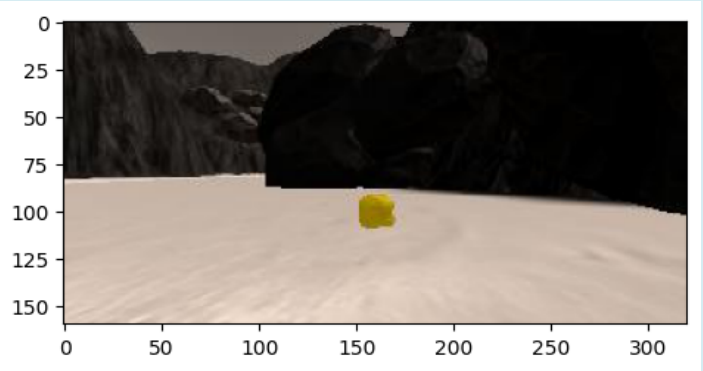
❖ Functions:

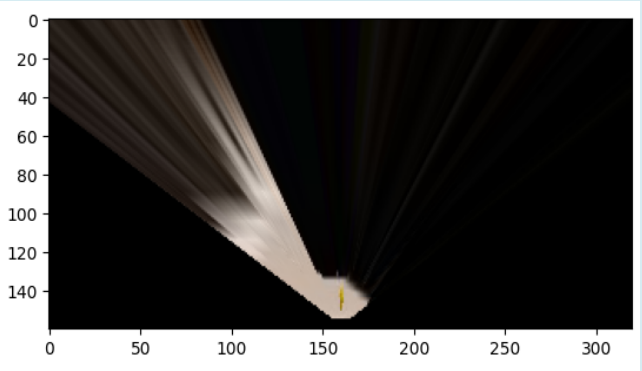
| | |
|---|---|
| **1** | **color_thresh** |
| **2** | **obstacles_tresh** |
| **3** | **rock_tresh** |
| **4** | **rover_coords** |
| **5** | **to_polar_coords** |
| **6** | **rotate_pix** |
| **7** | **translate_pix** |
| **8** | **pix_to_world** |
| **9** | **perspect_transform** |
| **10** | **perception_step** |
| **11** | **Decision_step** |

❖ Explanation of functions:

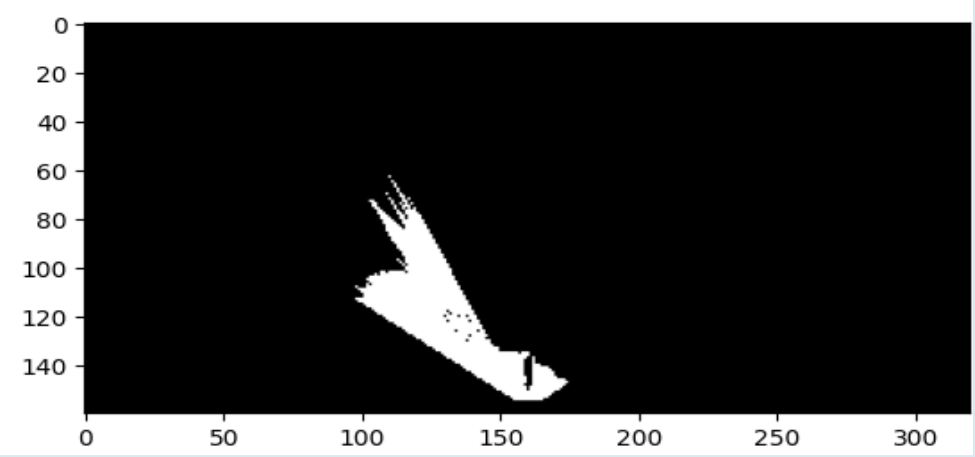
1) def color_thresh(img, rgb_thresh=(160, 160, 160)):

This function takes RGB image of perspective transform (Fig. 2) and three threshold values for RGB channels as input,navigable train is thresholded with high values of RED GREEN BLUE **160, 160, 160** ( as image will be saved in single channel), any pixel above threshold in all RGB values is set to 1 (white) returning binary image where:(white == navigable train , black == else)( Fig .3)



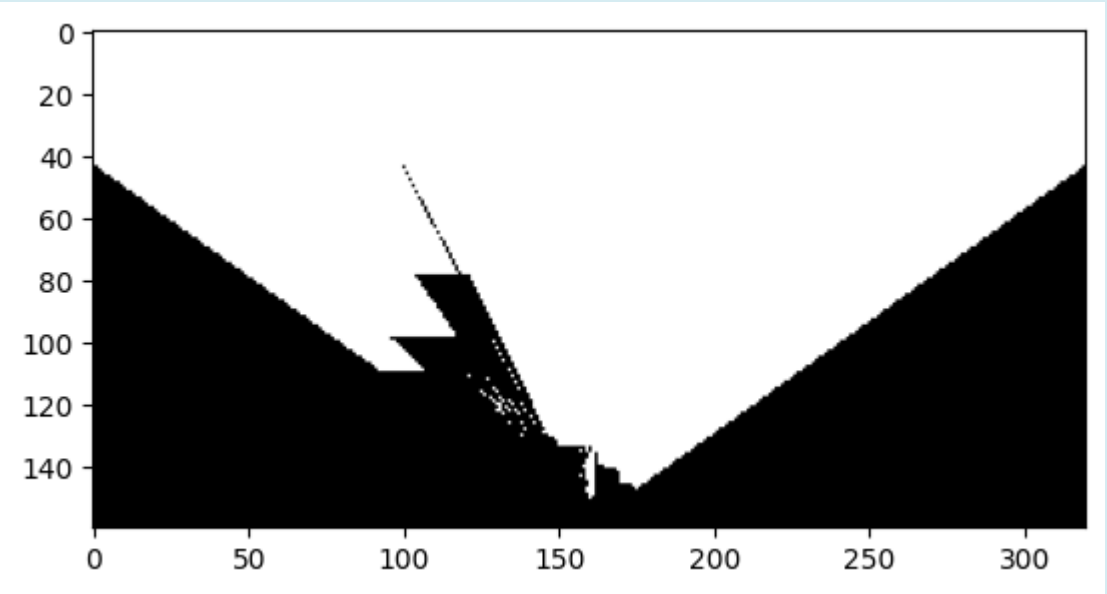

**Fig(1) original image** **Fig(2) bird_eye view**




**Fig(3) binary image**

2) obstacles_tresh :

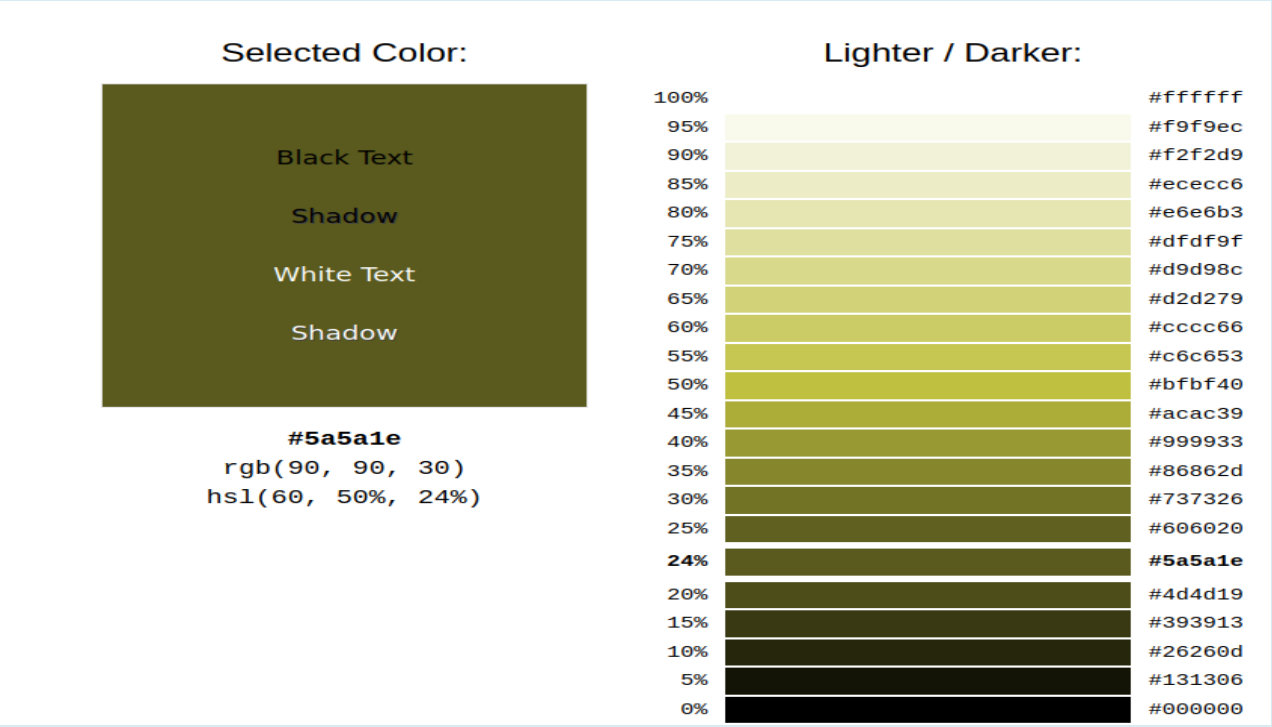
This function takes RGB image of perspective transform and three threshold values for RGB channels as input,navigable train is thresholded with high values of **rgb(160,160,160)**(Fig. 1)
( as image will be saved in single channel), obstacles are thresholded with lower than or equal 160 values of RED or GREEN orBLUE ( Fig .4)



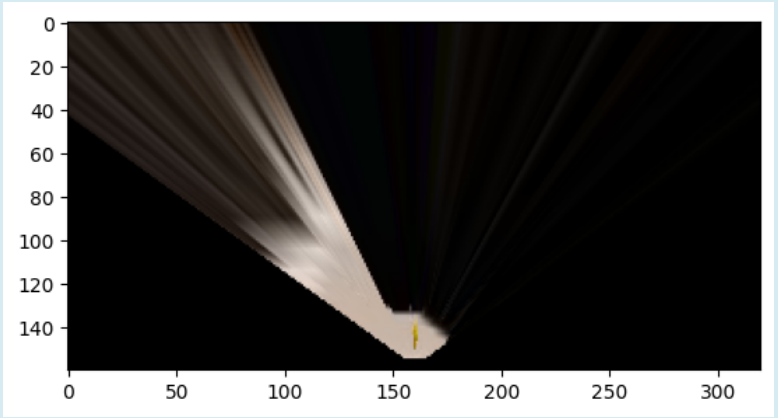

**Fig(4)**

3) rock_tresh:

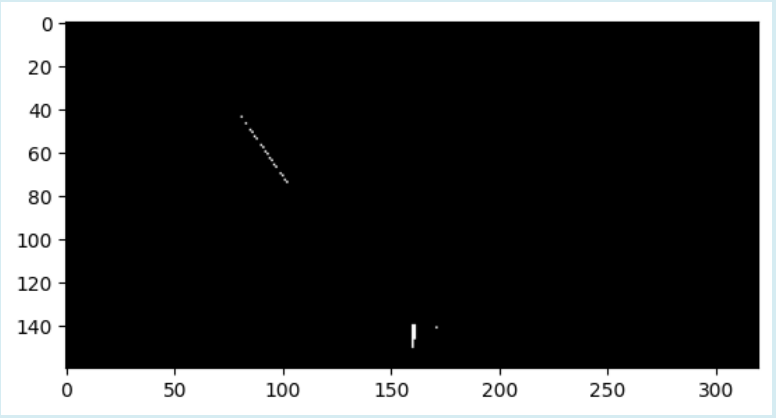
This function takes RGB image of perspective transform (Fig. 5) and three threshold values for RGB channels as input,rocks are thresholded with high values of RED & GREEN with much less BLUE **90, 90, 30**(shadowed yellow color is our threshold for rocks) (Fig. 6)( as image will be saved in single channel), (white == rock , black == else)( Fig .7)




(Fig. 6)




(Fig. 5)




( Fig .7)

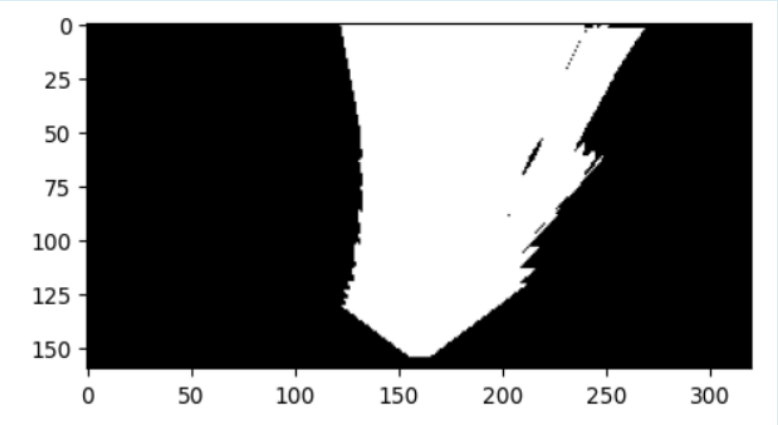
## 4)rover coords

This function convert
Takes binary image ,1st Identify nonzero pixels of the input image and gets(xpos,ypos) positions of x&y ,2nd reversing: where x in image coords is y in rover coords and  y in image coords is x in rover coords by using this equation

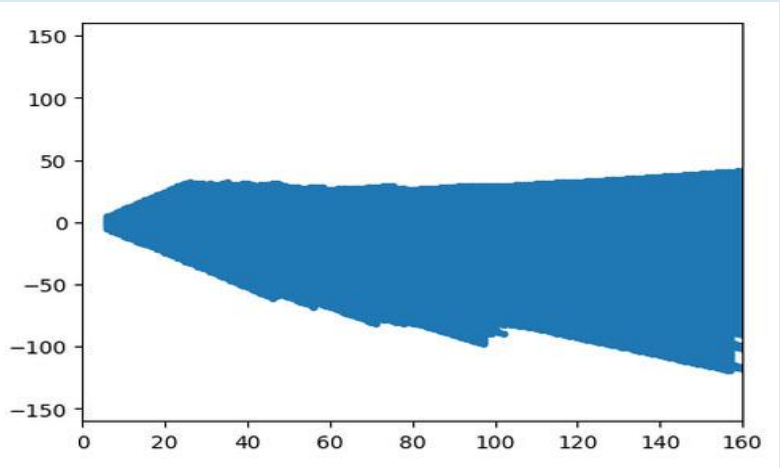
```
x_pixel = -(ypos - binary_img.shape[0])
y_pixel = -(xpos - binary_img.shape[1]/2 )
binary_img.shape[0]>>height(vertical)
binary_img.shape[1]>>width(horizontal)
```

Then return (x_pixel,y_pixel)

**Ex input image plot**




**Output image plot**




## 5)to_poler_coords

This function convert from radial coordinates to polar coordinates
Take (x,y) && return [distance (d),angle($\theta$)]
d=r √x^2 +y^2
$\theta = \tan^{-1}(y\ x)$

## 6)rotate_pix:

This function rotate the pixels in 2D that
takes( (x,y)and Θ)
and return (x',y')
Which (x,y) pixels before rotation and Θ is the angle of rotation
And the function return (x',y') pixels after applying the rotation
We use 2d rotation matrix to rotate pixels

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Which results in the following two equations where (x,y) are the cartesian coordinates of a point before applying the rotation, (x',y') are the cartesian coordinates of this point after applying the rotation and Θ is the angle of rotation.

$$x' = x.cos(\theta) - y.sin(\theta)$$

$$y' = x.sin(\theta) + y.cos(\theta)$$

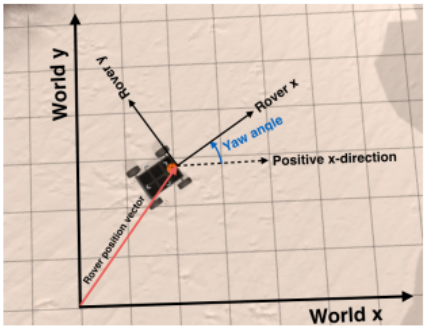
## 7) translate_pix :

This function ( takes pixels after applying the rotation and xpos, ypos, scale) and return( the pixels after scaling and a translation) the pixels will become at origin after applying translate_pix function .Which is xpos, ypos is the location of the robot .

8) pix_to_world :

The function parameters are the pixels of the rover (xpix,ypix) ,the world map pixels(xpos,ypos), its yaw angle , world map size and the scale.

This function is used to mapping to world map coordinates by applying rotation on rover axes by angle yaw to be parallel with world map axes ,then translate it to be centralized at origin as shown in the following picture and clipping its values to fit the world map image .The function returns pixels of x and y related to world map axes.




9) perspective_transform:

The perspective transform is a function that converts the original image to the bird eye view. It takes three parameters. The image we want to process, the source pixels of the original image and the destination pixels.

The filter M takes the source and destination pixels.

The function warp_prespective takes the following parameters, the image, the filter M,  the y axis of the image first through img.shape[1] then the x axis of the image through img.shape[0]. This function returns the warped image.

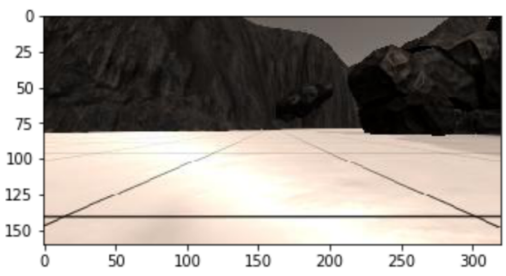



Figure 1: The original Image

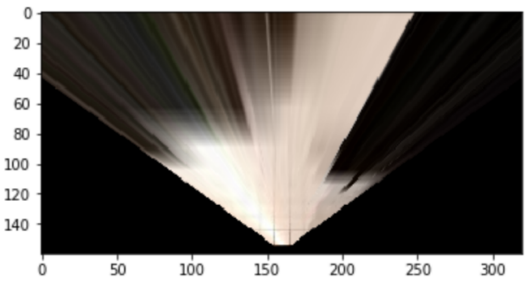



Figure 2: Bird eye view

## 10) perception_step

**# 1  Define source and destination points for perspective transform**

The source and destination points consist of 2D arrays. Where the points are taken in anticlockwise direction.We want the destination points to be at the middle of the axis therefore we used image.shape[1]/2 **+** dst_size for the y coordinate. Also we wanted to keep the edge of the image away from the bottom of the axis, hence we used image.shape[0] **-** bottom_offset, were the offset is 6 pixels. Like shown in the above figure.

**# 2  Apply perspective transform**

For the original  rover input image we apply perspective transform then find navegable terrain and threshold it to obtain binary image

```
warped = perspect_transform(Rover.img, source, destination)
thresh = color_thresh(warped)
```

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

For the original  rover input image we find rock samples and threshold them using rock_tresh function then we apply perspective transform on obtained binary image as follow:

*rocks = perspect_transform(rock_tresh(Rover.img), source, destination)*

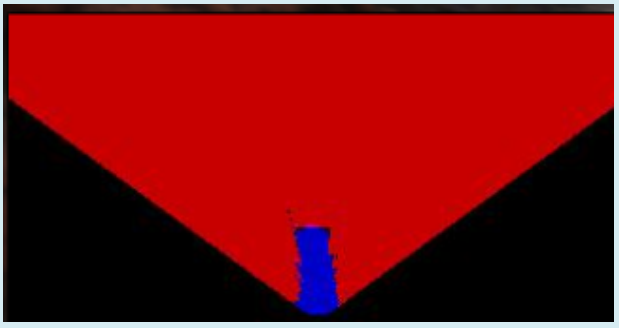
For the obstacles we simply do the same but using obstacles threshold as follow:

*obstacles = perspect_transform(obstacles_tresh(Rover.img), source, destination)*

**# 4) Update Rover.vision_image (this will be displayed on left side of screen) our image  is 200 x200 pixels**

Obstacles are given in Red channel , rocks are in Green and navigable train is in Blue as follow:

```
Rover.vision_image[:,:,2] = thresh*200 #navigable train set to Blue
Rover.vision_image[:,:,1] = rocks *200# rocks Set to GREEN
Rover.vision_image[:,:,0] = obstacles*200#obstacles set to Red
```



**#5** Convert map image(road,obstacles,rocks) pixel values to rover-centric coords & Calculate pixel values in rover-centric coords and distance/angle to all pixels using `(rover_coords function )`

**#6 We convert rover pixel values (including rock , obstacles ) to world coordinates by scaling 25**

```
worldmap = Rover.worldmap
scale = 25
obstacle_x_world, obstacle_y_world
=pix_to_world(obxpix,obypix,Rover.pos[0],Rover.pos[1],Rover.ya
w,worldmap.shape[0],scale)
rock_x_world, rock_y_world =
pix_to_world(roxpix,roypix,Rover.pos[0],Rover.pos[1],Rover.yaw
,worldmap.shape[0],scale)
navigable_x_world, navigable_y_world =
pix_to_world(xpix,ypix,Rover.pos[0],Rover.pos[1],Rover.yaw,wor
ldmap.shape[0],scale)
```

**#7 Update Rover world map (to be displayed on right side of screen)Updating the world map depending on condition of having ( pitch angle of rover < 1 or > 359) and (roll angle of rover < 1 or > 359)**

```
if ((Rover.pitch < 1 or Rover.pitch > 359) and (Rover.roll < 1
or Rover.roll > 359)):
Rover.worldmap[obstacle_y_world, obstacle_x_world, 0] = 200
Rover.worldmap[navigable_y_world, navigable_x_world, 0] = 0
Rover.worldmap[rock_y_world, rock_x_world, 1] = 200
Rover.worldmap[navigable_y_world, navigable_x_world, 2] = 200
```

```
# 8) Convert rover-centric pixel positions to polar coordinates
    Rover.nav_dists = dist
    Rover.nav_angles = angles
```

**descion.py file :**

```
##finding rocks

 # Check if there are rocks
        if Rover.rock_angle is not None and len(Rover.rock_angle) > 0
:
        Rover.mode = 'forward'
        Rover.steer = np.clip(np.mean(Rover.rock_angle * 180/np.pi),
-15, 15)
        # Move towards the rock slowly
        if not Rover.near_sample:
        if Rover.vel < Rover.max_vel/2:
                Rover.brake = 0
                Rover.throttle = 0.1
        else:
                Rover.throttle = 0
                Rover.brake = 1
        # Stop when close to a rock.
        else:
        Rover.throttle = 0
        Rover.brake = Rover.brake_set
        if(Rover.picking_up):
        Rover.brake = 0
        Rover.throttle = -0.2

 # Check the extent of navigable terrain
        if len(Rover.nav_angles) >= Rover.stop_forward :
 # the following conditions are added to stop rover from looping in
#same region in case navigable terrain allows by setting
#Rover.mode = 'stop'  and changing rover steer

                if Rover.steer > 14.5 and Rover.vel > 2:
                Rover.steer = 13
                Rover.mode = 'stop'
                if Rover.steer < -14.5 and Rover.vel > 2:
```

*Rover.steer = -13*
*Rover.mode = 'stop'*

**drive_rover.py file :**

Max velocity is set to 2.5
self.max_vel = 2.5

## Functions:

### 1) Return Home:

```
If Rover.samples_collected <= 5
  print("GO TO START")
  Dist_start = np.sqrt((Rover.pos[0] - Rover.start_pos[0])**2 + (Rover.pos[1] -
Rover.start_pos[1])**2)
if dist_start < 10.0 :
  print("10m From start")
  Rover.mode = 'stop'
  Rover.throttle = 0
  Rover.brake = Rover.brake_set
  return Rover
  print(Rover.samples_collected)
```

**An if condition that checks the number of rocks picked, if 5 or more rocks are picked up, it estimates the distance between the rover and the starting point. Then the rover heads back to the starting point.**

### 2) Avoid Obstacle:

```
if (len(Rover.ob_dist)<=40 or len(Rover.ob_angle)is not None) and
Rover.rock_angle is None :
          if Rover.vel !=0:
             Rover.brake=10
          Rover.steer=-15
          Rover.throttle=0
          Rover.brake=0
          if nav_area>650 :
             Rover.mode='forward'
          if nav_area<650 :
             Rover.mode='stop'
```

An if condition that checks the distance between the rover and the obstacle. If it is less than 40 pixels, the rover stops and moves backwards until there is enough area in front of it (> 650) to move forward again without hitting an obstacle. If there are no obstacles in the way, the rover moves forward.

## 3) Rock detection and pick up:

```
if Rover.rock_angle is not None and len(Rover.rock_angle) > 0:
  Rover.mode = 'forward'
  Rover.steer = np.clip(np.mean(Rover.rock_angle * 180/np.pi), -15, 15)
  if not Rover.near_sample:
     if Rover.vel < Rover.max_vel/2:
         Rover.brake = 0
         Rover.throttle = 0.1
     else:
         Rover.throttle = 0
         Rover.brake = 1
  Else:
     Rover.throttle = 0
     Rover.brake = Rover.brake_set
```

If a rock is detected in the map and the rover is still far away from it then the rover moves towards it and slows down. If the rover needs to throttle, it can't do that unless its velocity is at least half the maximum value. Also the steering of the rover is limited to the range [-15,15] any value greater than + 15 or smaller than - 15 is set to 15 or -15 respectively. If the rover is close to the sample it stops and picks it up

**LINk to repo:**
**https://github.com/zainbmaged/Image-Processing-Project.git**