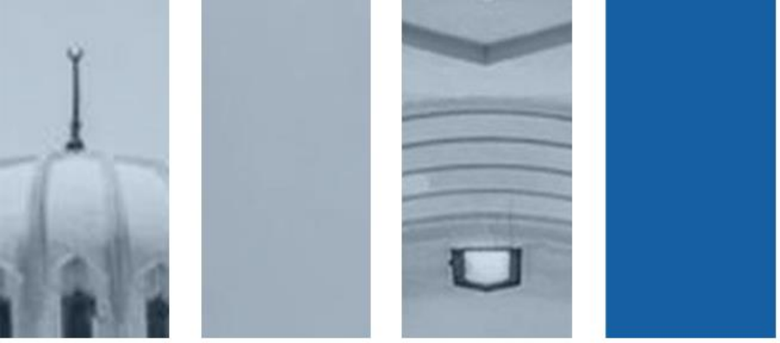


Mars Rover Project

Program: Computer and Systems Engineering

Course Code: CSE374s
Course Name: Digital Image Processing

Ain Shams University
Faculty of Engineering
Fall Semester – 2022

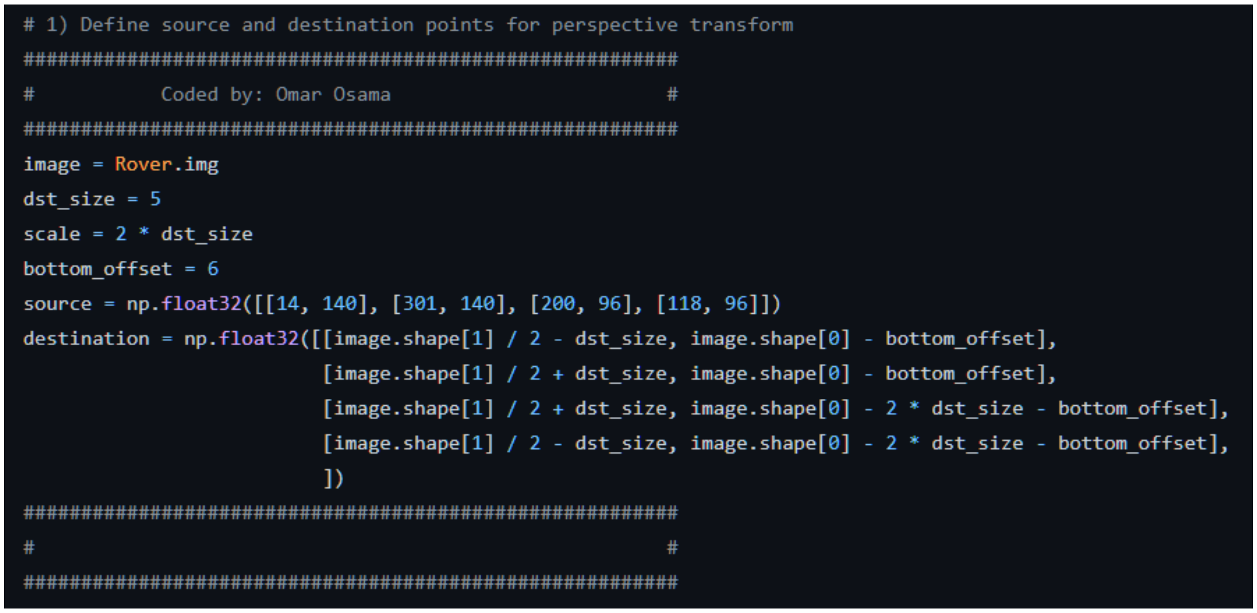
Student Personal Information

Student Names:	Student Codes:
Maram Ahmed Hussein Mostafa	1900050
Habiba Ahmed Alaa Eldin Mohamed	1900839
Shiry Ezzat Hakim Attalla	2001156
Omar Osama Abd ElMonem	2001754
Engy Mohamed Abdelmordy Negm	1900630

GitHub Repo: https://github.com/OmarAMonem/Mars_Rover.git

Perception steps updated Rover():

1) Define source and destination points for perspective transform



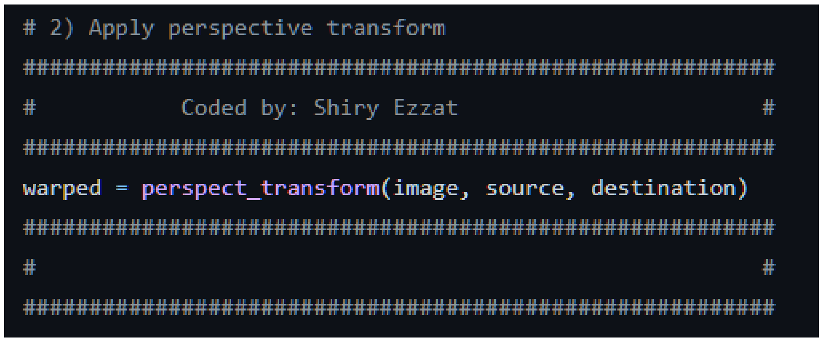
```
# 1) Define source and destination points for perspective transform
#########################################################
#           Coded by: Omar Osama                        #
#########################################################
image = Rover.img
dst_size = 5
scale = 2 * dst_size
bottom_offset = 6
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],
                          ])
#########################################################
#                                                       #
#########################################################
```

Define calibration box in source (actual) and destination (desired) coordinates

These source and destination points are defined to warp the image to a grid where each 10x10 pixel square represents 1 square meter.

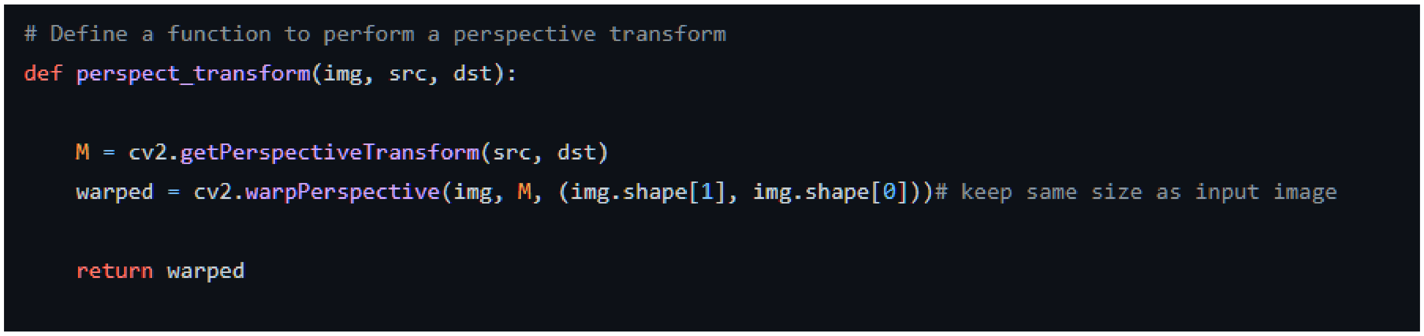
The destination box will be 2*dst_size on each side

2) Apply perspective transform



```
# 2) Apply perspective transform
#########################################################
#           Coded by: Shiry Ezzat                       #
#########################################################
warped = perspect_transform(image, source, destination)
#########################################################
#                                                       #
#########################################################
```

This function generates a filter to be applied on the source image and its output is the eyebird view the function generates the filter by giving it four points



```
# Define a function to perform a perspective transform
def perspect_transform(img, src, dst):

    M = cv2.getPerspectiveTransform(src, dst)
    warped = cv2.warpPerspective(img, M, (img.shape[1], img.shape[0]))# keep same size as input image

    return warped
```

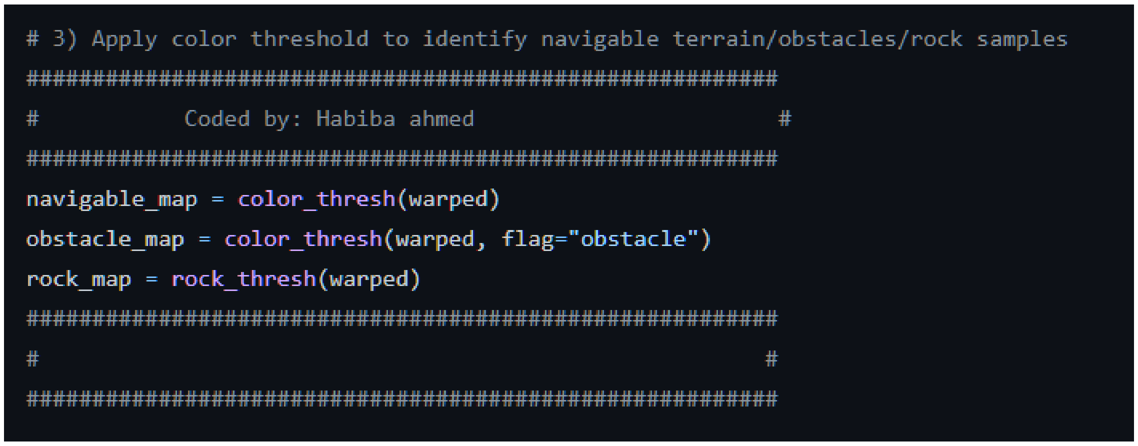
```
Syntax: def perspect_transform(img, src, dst)

Description: Transform prespective to eyebird view

Parameters (in): img -> original image

                 src -> four points I provided

                 dst -> four arbitrary points that must form a square shape

Return value : warped
```

M is the mask that openCV generats to transform from the four points I provided (src) to the four arbitary points (dst)

Note that: four points on (dst) must form a square shape

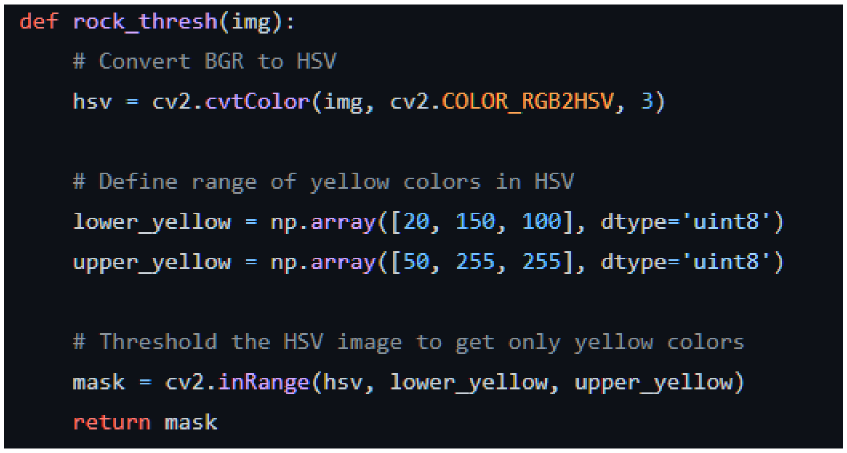
Then when appling the musk on any image the eyebird view will be generated

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



```
# 3) Apply color threshold to identify navigable terrain/obstacles/rock samples
#########################################################
#           Coded by: Habiba ahmed                      #
#########################################################
navigable_map = color_thresh(warped)
obstacle_map = color_thresh(warped, flag="obstacle")
rock_map = rock_thresh(warped)
#########################################################
#                                                       #
#########################################################
```

navigable_map: Identify the ground on which the rover can move, uses the default flag
obstacle_map: Identify the obstacles that the rover should avoid, uses the obstacle flag
rock_map = Identify the rocks that the rover should grip



```
def rock_thresh(img):
    # Convert BGR to HSV
    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV, 3)

    # Define range of yellow colors in HSV
    lower_yellow = np.array([20, 150, 100], dtype='uint8')
    upper_yellow = np.array([50, 255, 255], dtype='uint8')

    # Threshold the HSV image to get only yellow colors
    mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
    return mask
```

Syntax: def rock_thresh(img):

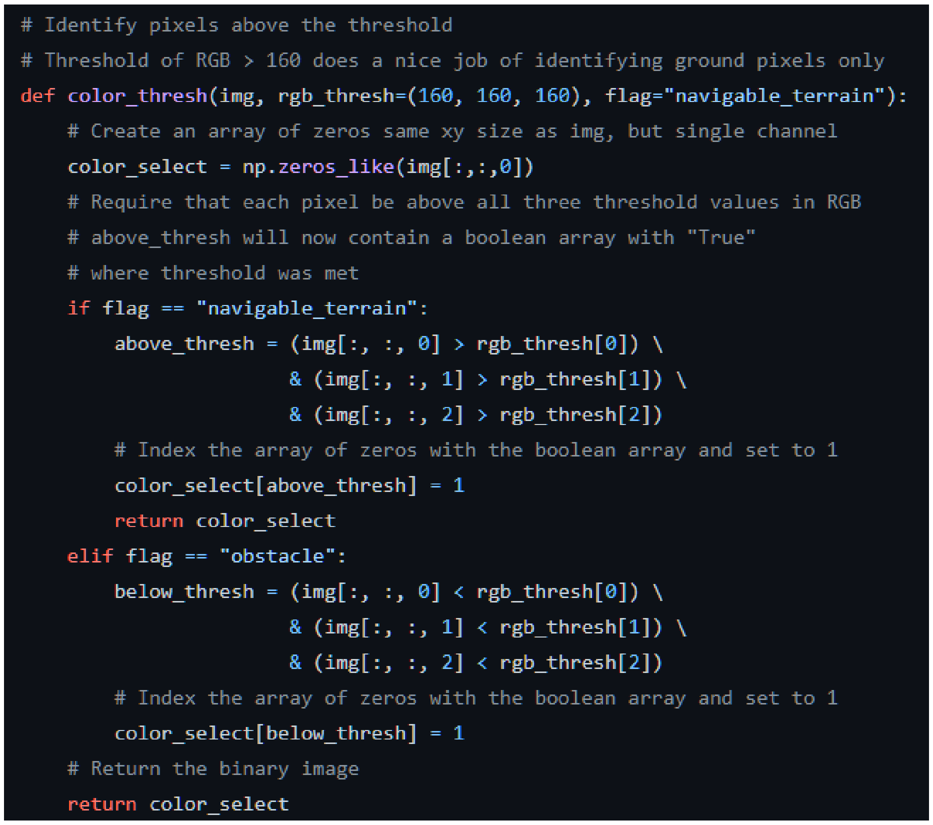
Description: Identify pixels above the threshold to detect the rocks

Parameters (in): img -> input image in eyebird view

Return value : mask

lower_yellow and upper_yellow define the range of the yellow/gold colors to identify the rock.

The mask converts the RGB values to HSV vales to detect rocks more accurately.
HSV based color space is more accurate compared to RGB color space in autonomous system.



```python
# Identify pixels above the threshold
# Threshold of RGB > 160 does a nice job of identifying ground pixels only
def color_thresh(img, rgb_thresh=(160, 160, 160), flag="navigable_terrain"):
    # Create an array of zeros same xy size as img, but single channel
    color_select = np.zeros_like(img[:,:,0])
    # Require that each pixel be above all three threshold values in RGB
    # above_thresh will now contain a boolean array with "True"
    # where threshold was met
    if flag == "navigable_terrain":
        above_thresh = (img[:, :, 0] > rgb_thresh[0]) \
                       & (img[:, :, 1] > rgb_thresh[1]) \
                       & (img[:, :, 2] > rgb_thresh[2])
        # Index the array of zeros with the boolean array and set to 1
        color_select[above_thresh] = 1
        return color_select
    elif flag == "obstacle":
        below_thresh = (img[:, :, 0] < rgb_thresh[0]) \
                       & (img[:, :, 1] < rgb_thresh[1]) \
                       & (img[:, :, 2] < rgb_thresh[2])
        # Index the array of zeros with the boolean array and set to 1
        color_select[below_thresh] = 1
    # Return the binary image
    return color_select
```

Syntax: def color_thresh(img, rgb_thresh=(160, 160, 160),flag="navigable_terrain")

Description: Identify pixels above the threshold to detect the ground

Parameters (in): img -> input image in eyebird view

rgb_thresh=(160, 160, 160) -> Threshold of RGB > 160 does a nice job of identifying ground pixels only

flag="navigable_terrain"-> A flag that has a default value “drive”

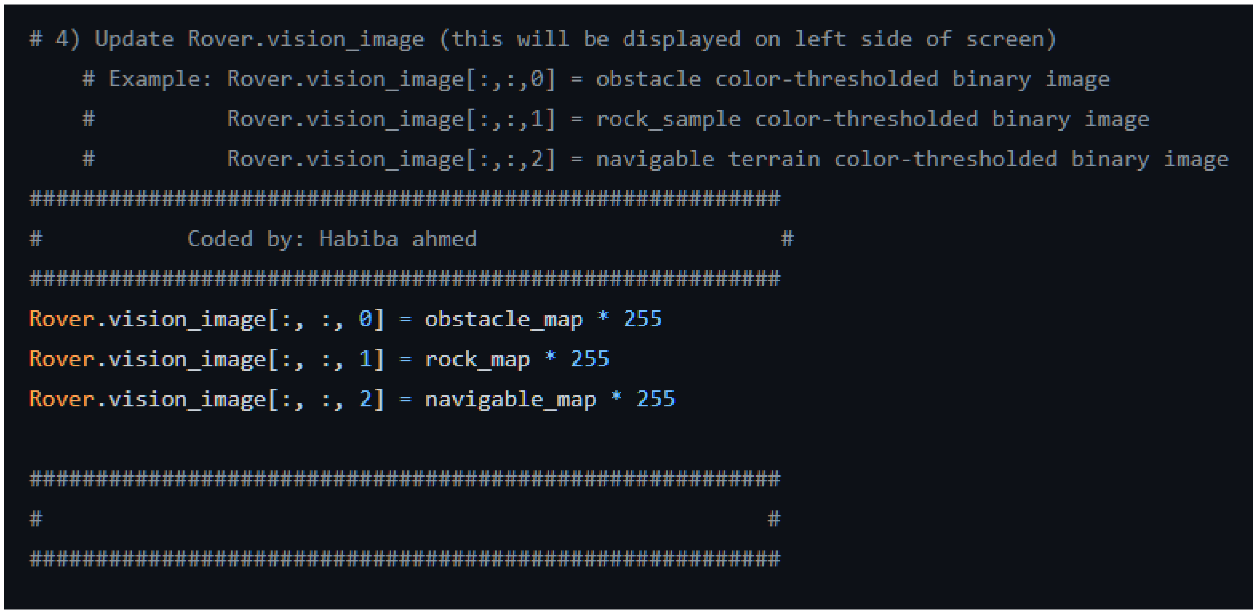
Return value : color_select

The variable “above_thresh” loops around the image to check if each pixel is above the threshold or not.

If the flag equals “drive”, “above_thresh” returns ones in place of the pixels whose values are above the threshold to identify the ground pixels

If the flag equals “obstacle”, “above_thresh” returns ones in place of the pixels whose values are below the threshold to identify the obstacles.

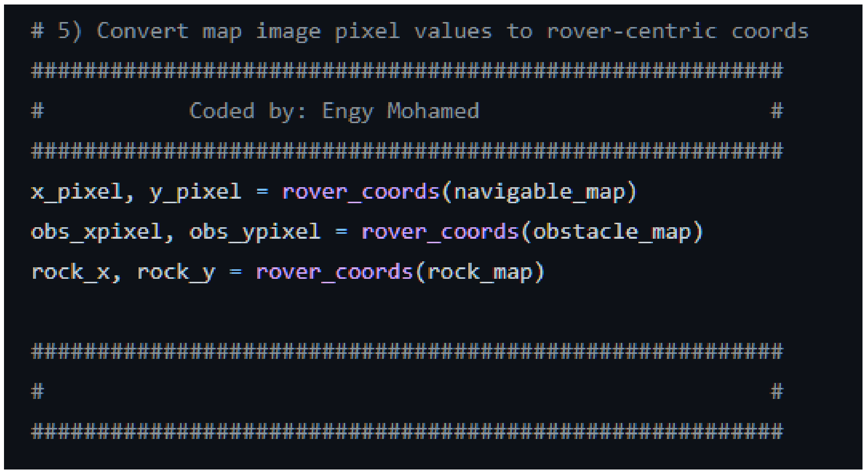
4) Update Rover.vision_image



```
# 4) Update Rover.vision_image (this will be displayed on left side of screen)
    # Example: Rover.vision_image[:,:,0] = obstacle color-thresholded binary image
    #          Rover.vision_image[:,:,1] = rock_sample color-thresholded binary image
    #          Rover.vision_image[:,:,2] = navigable terrain color-thresholded binary image
#########################################################
#           Coded by: Habiba ahmed                      #
#########################################################
Rover.vision_image[:, :, 0] = obstacle_map * 255
Rover.vision_image[:, :, 1] = rock_map * 255
Rover.vision_image[:, :, 2] = navigable_map * 255

#########################################################
#                                                       #
#########################################################
```

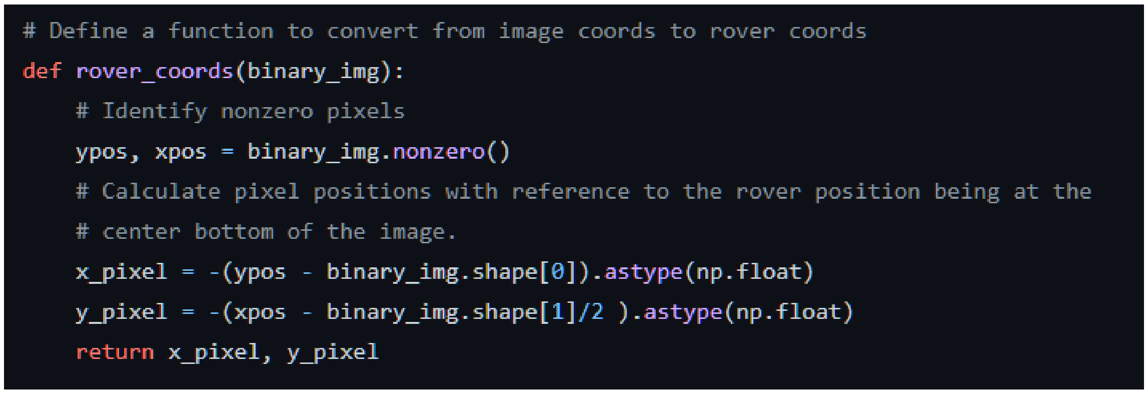
Make	Obstacle color	-> RED
	Rock color	-> GREEN
	Navigable color	-> BLUE

5) Convert map image pixel values to rover-centric cords



```
# 5) Convert map image pixel values to rover-centric coords
#########################################################
#           Coded by: Engy Mohamed                      #
#########################################################
x_pixel, y_pixel = rover_coords(navigable_map)
obs_xpixel, obs_ypixel = rover_coords(obstacle_map)
rock_x, rock_y = rover_coords(rock_map)

#########################################################
#                                                       #
#########################################################
```

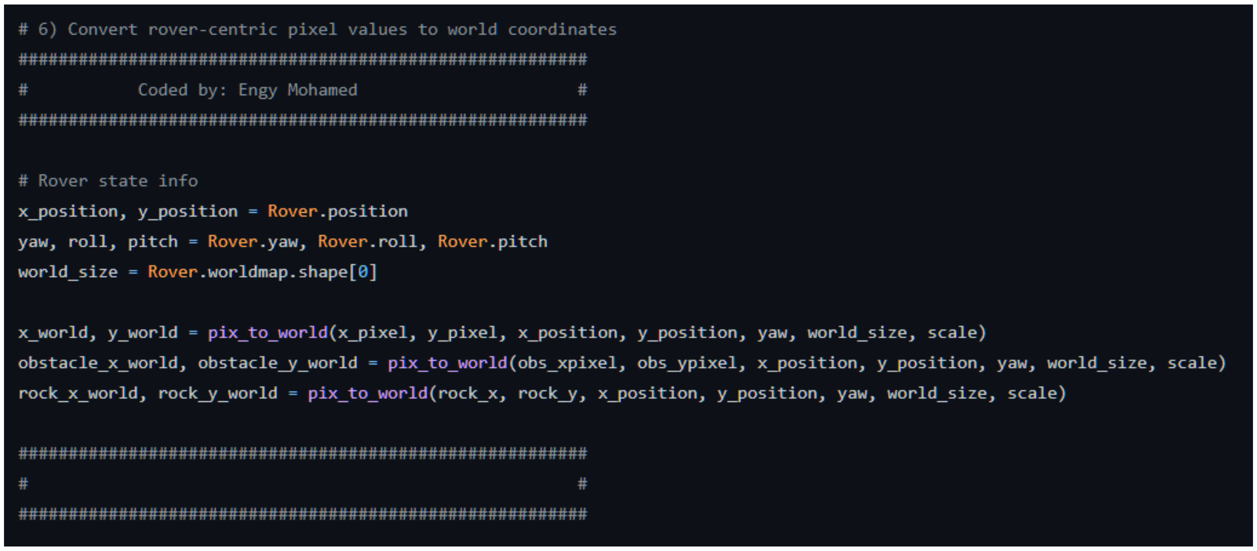
This Part use the function “rover coords” which take a binary image “image coords” and convert it to the rover position after changing the origin of the image returning x_pixel and y_pixel after editing



```
# Define a function to convert from image coords to rover coords
def rover_coords(binary_img):
    # Identify nonzero pixels
    ypos, xpos = binary_img.nonzero()
    # Calculate pixel positions with reference to the rover position being at the
    # center bottom of the image.
    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
```

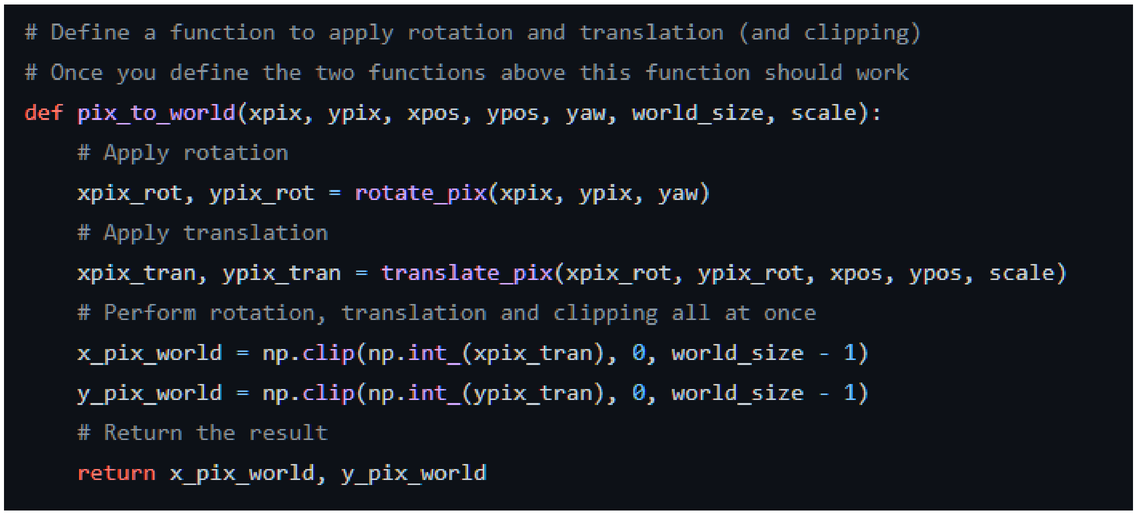
```
Syntax: def rover_coords(binary_img)

Description: Convert from image coords to rover coords

Parameters (in): binary_img

Return value : x_pixel, y_pixel
```

6) Convert rover-centric pixel values to world coordinates



```
# 6) Convert rover-centric pixel values to world coordinates
#########################################################
#           Coded by: Engy Mohamed                     #
#########################################################

# Rover state info
x_position, y_position = Rover.position
yaw, roll, pitch = Rover.yaw, Rover.roll, Rover.pitch
world_size = Rover.worldmap.shape[0]

x_world, y_world = pix_to_world(x_pixel, y_pixel, x_position, y_position, yaw, world_size, scale)
obstacle_x_world, obstacle_y_world = pix_to_world(obs_xpixel, obs_ypixel, x_position, y_position, yaw, world_size, scale)
rock_x_world, rock_y_world = pix_to_world(rock_x, rock_y, x_position, y_position, yaw, world_size, scale)

#########################################################
#                                                       #
#########################################################
```

This part use the function “pix_to_world” which apply the rotation, translation and clipping to the rover coordinates by the yaw angle and return the pixels after rotating (to be projected on the world map)



```
# Define a function to apply rotation and translation (and clipping)
# Once you define the two functions above this function should work
def pix_to_world(xpix, ypix, xpos, ypos, yaw, world_size, scale):
    # Apply rotation
    xpix_rot, ypix_rot = rotate_pix(xpix, ypix, yaw)
    # Apply translation
    xpix_tran, ypix_tran = translate_pix(xpix_rot, ypix_rot, xpos, ypos, scale)
    # Perform rotation, translation and clipping all at once
    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 the result
    return x_pix_world, y_pix_world
```

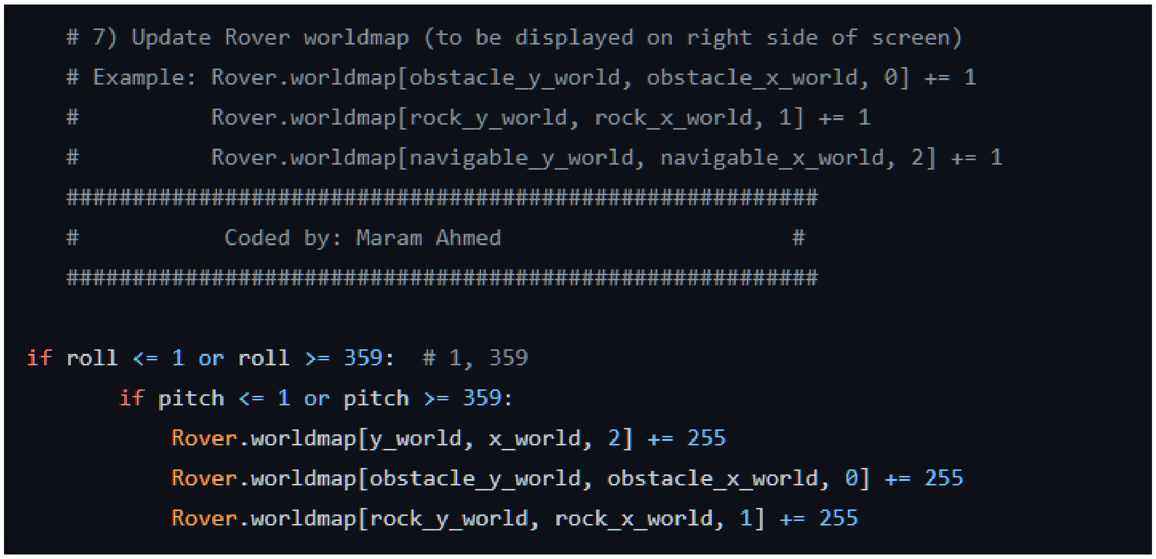
`Syntax: def pix_to_world(xpix, ypix, xpos, ypos, yaw, world_size, scale)`

`Description: Apply rotation and translation (and clipping)`

`Parameters (in): xpix, ypix, xpos, ypos, yaw, world_size, scale`

`Return value : x_pix_world, y_pix_world`

7) Update Rover worldmap (to be displayed on right side of screen)



```
   # 7) Update Rover worldmap (to be displayed on right side of screen)
   # Example: Rover.worldmap[obstacle_y_world, obstacle_x_world, 0] += 1
   #          Rover.worldmap[rock_y_world, rock_x_world, 1] += 1
   #          Rover.worldmap[navigable_y_world, navigable_x_world, 2] += 1
   #########################################################
   #           Coded by: Maram Ahmed                       #
   #########################################################

if roll <= 1 or roll >= 359:  # 1, 359
       if pitch <= 1 or pitch >= 359:
           Rover.worldmap[y_world, x_world, 2] += 255
           Rover.worldmap[obstacle_y_world, obstacle_x_world, 0] += 255
           Rover.worldmap[rock_y_world, rock_x_world, 1] += 255
```

Checks if	Roll is in the range[1:359]
	Pitch is in the range[1:359]

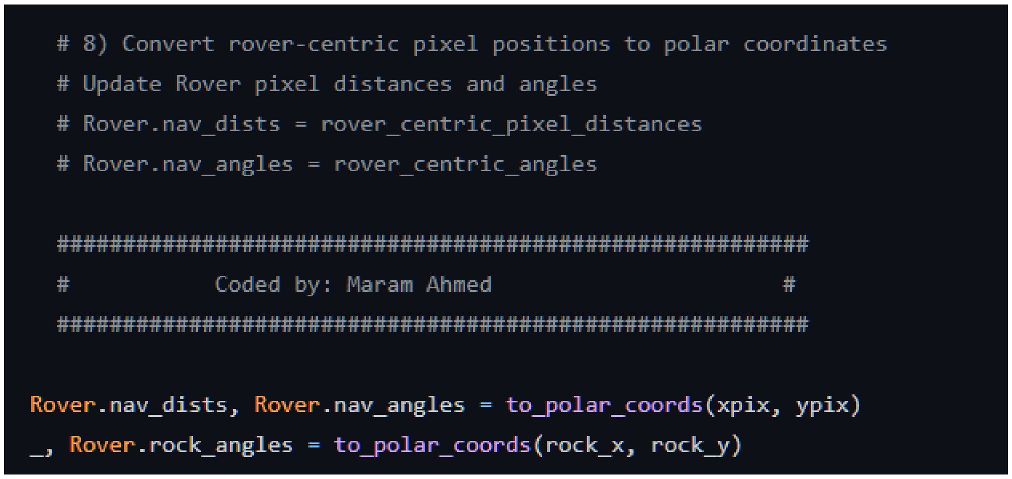
If both conditions are True draw the x and y coordinates on the map

Rock -> white

Obstacles -> RED

Road -> BLUE

8) Convert rover-centric pixel positions to polar coordinates



```
  # 8) Convert rover-centric pixel positions to polar coordinates
  # Update Rover pixel distances and angles
  # Rover.nav_dists = rover_centric_pixel_distances
  # Rover.nav_angles = rover_centric_angles

  #########################################################
  #           Coded by: Maram Ahmed                       #
  #########################################################

Rover.nav_dists, Rover.nav_angles = to_polar_coords(xpix, ypix)
_, Rover.rock_angles = to_polar_coords(rock_x, rock_y)
```

Convert x and y rover coordinates to get distance and angles using polar coordinates.

Ignore distances for rocks, convert x and y rover coordinates to polar coordinates to get rock angles.