[Date]

Computer vision phase 1 document

Team 20

19p7095	ziad ashraf ahmed ahmed kasem
19p9610	Gannah Allah Mohamed GAber MAhmoud
19p9597	Yassin Khaled Mostafa Attia Mahgub
19p4388	Adham Ehab Salman Selim
17P8053	Mohamed Mahmoud Sadek Mahmoud Rehab

At first we need some supporting libraries to import to help us dealing with images and other process

Example for these libraries

- CV2
- Numpy
- Matplotlib.image
- Matplotlib.pyplot
- Scipy.misc
- glob
- imageio

===

Color Thresholding

We have 3 objects needed to be mapped in out word

1- our navigable road
2- the Infront obstacles
3- The position of the rocks

For the navigable road we implemented **navigable_thresh function** in perception.py

First it create array of zeros with size = the image size but as single channel

Second apply the selected threshold on each pixel

Third set the value of the pixel = 1 in the array if it is above the applied threshold.

Fourth : return that array

For the Infront obstacle we implemented **obstacle_thresh function** in perception.py

It has the same concept of navigable_thresh function but this time pixel considered as obstacle if it is below the selected threshold.

as obstacle at our map tends to be the dark objects.

For the Rock position we implemented **rock_thresh function** in perception.py

First :We know that our rocks are yellow color

We used OpenCV to get the needed must to detect our rock

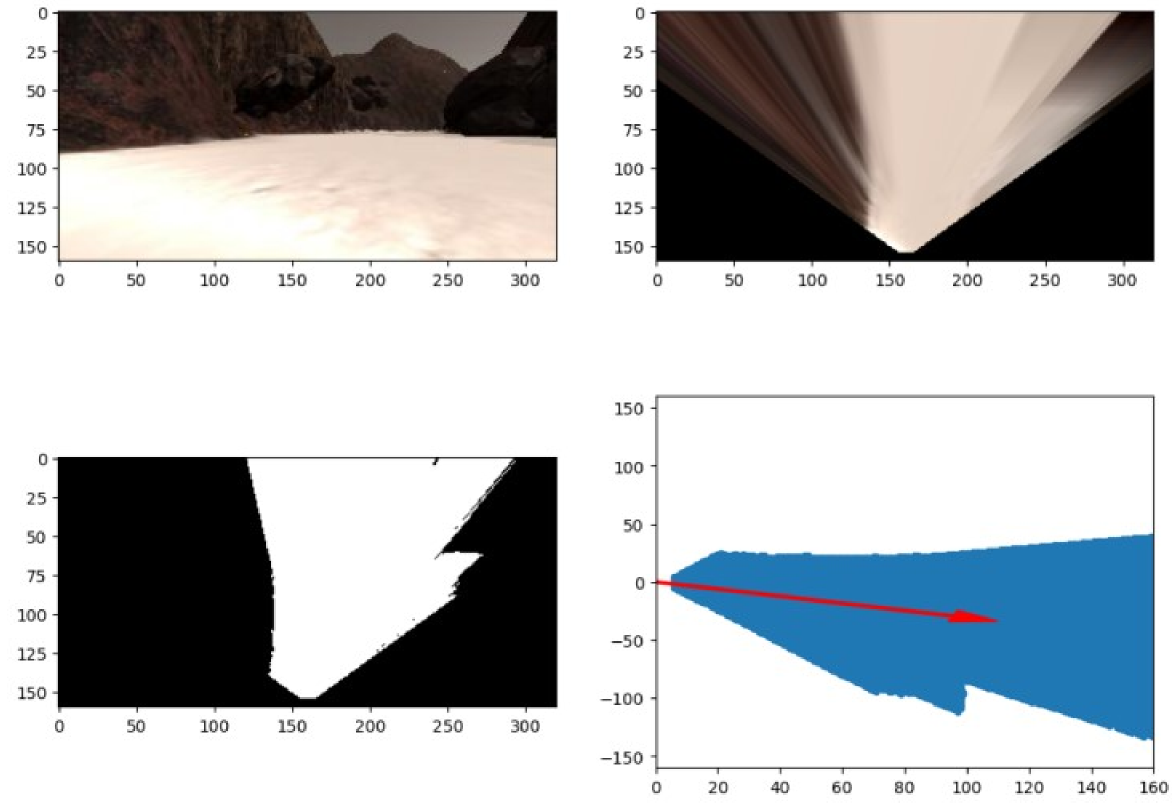
At the beginning we converted our BGR image to HSV then defining our yellow Range.

After that we used **inRange function** provided by open CV to get the needed musk then returning it

---

Coordinate transformation

We need to convert from image coordinate to Rover coordinates as shown in the next images



1) **rover_coords function** to convert the binary image To centric coordinate using translation

Now the Rover x axis represent the front of the rover.

Steps : create a nonzero pixels array of the binary image

Now using translation of pixel by reference to the Rover position (Bottom center of the image)

Note : we have to flipe the xpos and ypos

2) **to_polar_coords function** we here need to convert our coordinates to polar coordinate (distance , angle) in order to calculate the angle needed in the navigating the Rover.

Steps :- we need to use pythagoream therem in order to calulate the distance

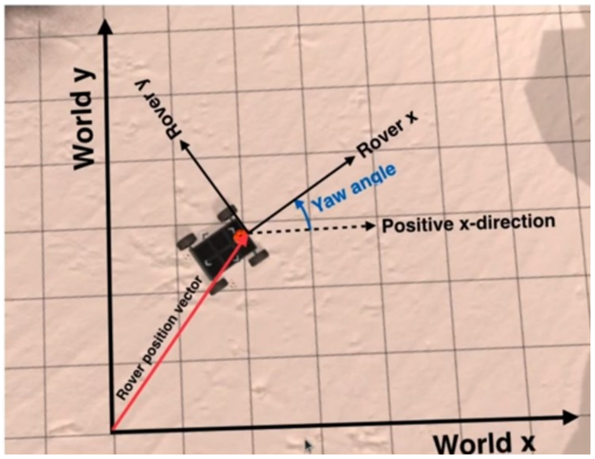
between each pixels

use arc tan to calculate the angle from the vertical for each pixel

return calculated angle and distance

---

3) **rotate_pix function** we need here to rotate image by yaw angle so rover x and rover y be parallel to word x and word y respectively.



As explained in the lecture we will use 2D rotational matrix

Steps :- convert the yaw degree to radians to be able to deal with it

Apply matrix rotation method

Return the xpix_rotated and ypix_rotated at the end

---

4) **translate_pix function**

Here we will translate the pre rotated pixel array in the previous function By the positions X and Y values given by the rover’s Location

steps :- Translate the X and Y pixels by the value of x and y rotation scaled

Return xpix_translated, ypix_translated

---

5) **Pix_to_word function**

We take all the parameters needed to map the rover centric coordinates to the world coordinates and returning the arrays of the provided x and y coordinates

mapped to the world coordinates

steps :- make X and Y parallel to the word axes using (Rotation)

using translation to Translate the rotated arrays by the rover's location in

the world

finally clip the array ranges to fit within the world map

---

6) perspect_transform(img, src, dst)

==> **Function**

Performs a perspective transformation on the raw image array to change

the rover's camera image to a synthetic map image (bird-eye view of environment).

==> **Parameters**

* 3D Numpy array (x, y, RGB) of the rover's camera

* 2D Numpy array of integers indicating a square in the raw image

* 2D Numpy array of floats indicating how the square from the raw image should be transformed to make a perfect square for the output

==> **Return**

- 3D Numpy array (x, y, RGB) of the bird-eye view image

==> **Operation**
Applies the mask M on the received camera image (img) and returns bird-eye view image

7- perception_step(Rover)
==> Function

- Calculates the Rover's current environment from the position values and

the front camera image. Update the Rover's state after perception.
==> Parameters

- Class of the Rover's state

==> Return

- Updated Rover class

==> Operation

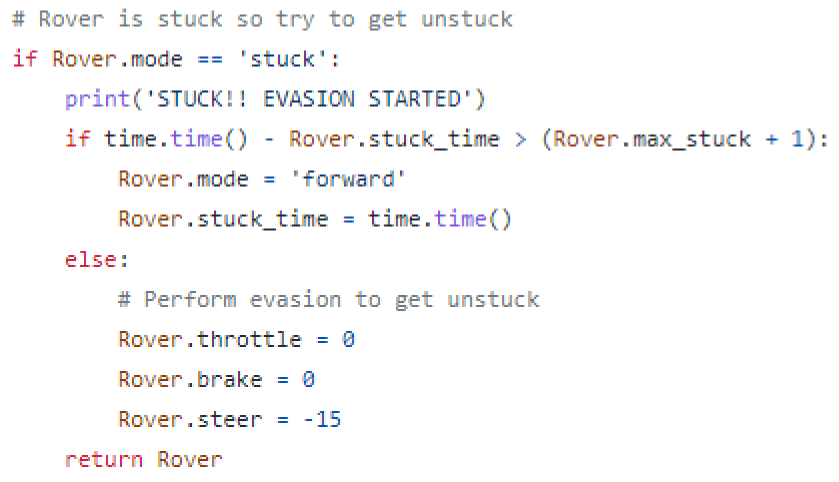
- Define source and destination points for perspective transform
- Apply perspective transform using function percpect_transform(img, src, dst)
- Apply color threshold to identify navigable navigable_thresh(img=warped, rgb_thresh=(160, 160, 160)), obstacle_thresh(img=warped, rgb_thresh=(160, 160, 160)), rock_thresh(img=warped)
- Update Rover.vision_image
- Convert map image pixel values to rover centric coordinates using function rover_coords(binary_img)
- Convert rover centric pixel values to world coordinates
- Update Rover worldmap
- Convert rover-centric pixel positions to polar coordinates using function to_polar_coords(x_pixel, y_pixel)
- Update Rover pixel distances and angles
- Show the pipeline of your rover journey:

cv2.imshow("warped",warped)

- cv2.imshow("navigable",navigable*255)
- cv2.imshow("rock",rock_samples)
- cv2.imshow("obstacle",obstacles*255)
- cv2.imshow("image",img)
- cv2.waitKey(5)

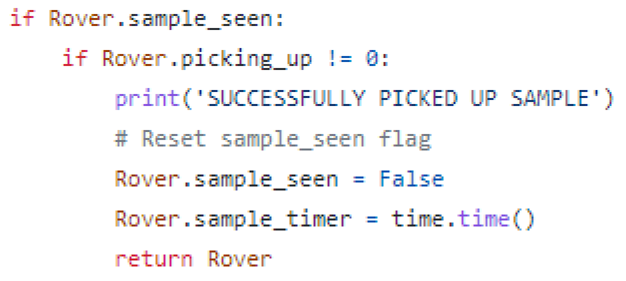
to the decision file we added conditions to enhance the motion of the rover:

1. if it’s stuck, the rover would change it’s directions to get unstuck, would be checked in many other moods for the rover whenever it’s stuck.

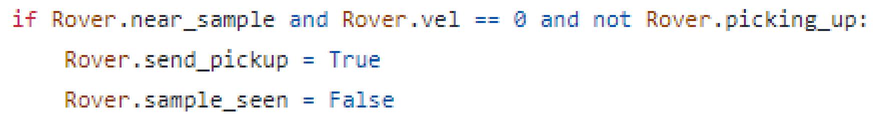


```
# Rover is stuck so try to get unstuck
if Rover.mode == 'stuck':
    print('STUCK!! EVASION STARTED')
    if time.time() - Rover.stuck_time > (Rover.max_stuck + 1):
        Rover.mode = 'forward'
        Rover.stuck_time = time.time()
    else:
        # Perform evasion to get unstuck
        Rover.throttle = 0
        Rover.brake = 0
        Rover.steer = -15
    return Rover
```

2. If rover have seen a sample it would locate it, then if the rover is near the sample and it’s velocity is equal to zero it would pick it

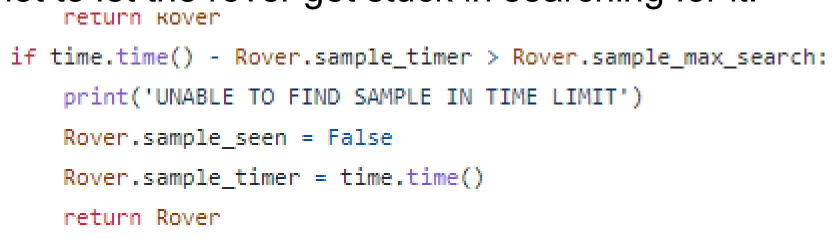


```
if Rover.sample_seen:
    if Rover.picking_up != 0:
        print('SUCCESSFULLY PICKED UP SAMPLE')
        # Reset sample_seen flag
        Rover.sample_seen = False
        Rover.sample_timer = time.time()
        return Rover
```



```
if Rover.near_sample and Rover.vel == 0 and not Rover.picking_up:
    Rover.send_pickup = True
    Rover.sample_seen = False
```

3. If a time passed without picking the rock. Marked as unseen, in order not to let the rover get stuck in searching for it.

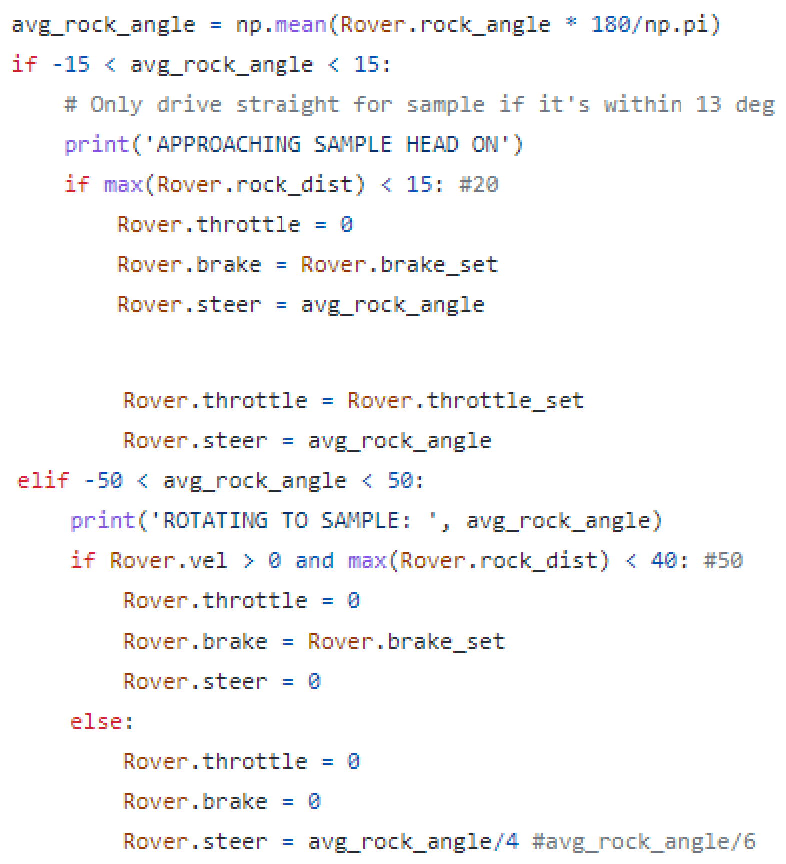


```
    return Rover
if time.time() - Rover.sample_timer > Rover.sample_max_search:
    print('UNABLE TO FIND SAMPLE IN TIME LIMIT')
    Rover.sample_seen = False
    Rover.sample_timer = time.time()
    return Rover
```

4. We put conditions for best way to pickup the samples without being stuck or hit.
    1. The object is being picked up if it’s between 15 deg (exclusive). When object is on distance less than 15 start the breaks and steer

    to the rock angle

    2. If the object is +- 50 degrees we would rotate to it
       When getting nearer(less than 50) the brakes would start



```
avg_rock_angle = np.mean(Rover.rock_angle * 180/np.pi)
if -15 < avg_rock_angle < 15:
    # Only drive straight for sample if it's within 13 deg
    print('APPROACHING SAMPLE HEAD ON')
    if max(Rover.rock_dist) < 15: #20
        Rover.throttle = 0
        Rover.brake = Rover.brake_set
        Rover.steer = avg_rock_angle


        Rover.throttle = Rover.throttle_set
        Rover.steer = avg_rock_angle
elif -50 < avg_rock_angle < 50:
    print('ROTATING TO SAMPLE: ', avg_rock_angle)
    if Rover.vel > 0 and max(Rover.rock_dist) < 40: #50
        Rover.throttle = 0
        Rover.brake = Rover.brake_set
        Rover.steer = 0
    else:
        Rover.throttle = 0
        Rover.brake = 0
        Rover.steer = avg_rock_angle/4 #avg_rock_angle/6
```

For the Jupiter notebook test functions:

First we read the path of image from a csv excel file. Using panda library.

A class is then created to save the read images called “data bucket”.

The images would be then processes using **process_image()** function, where:

1. Defining the source and destination of perspective transform.
2. Apply the perspective transform on image using **perspective_tarnsform()** function
3. Apply threshold to each of rocks, obstacles and navigable terrain.
4. In order to be seen by the rover, convert it to rover centric coordinates using rover_navigable()
5. Scale it to world map to be seen on the screen and update it.
6. The output image would be saved in out_put[] list after that

The output image would the be converted to a video using

ImageSequenceClip(data.images, fps=60) function