COMPUTER VISION PROJECT

Phase 2

Submitted to:

Dr. Mahmoud Ibrahim Khalil

Eng. Mahmoud Selim

Submission date:

02 January 2023

Submitted by:

Team 6

Zakaria Sobhy Abd-ElSalam Soliman Madkour 19P2676

Mahmoud Mohamed Omar Ibrahim 19P5803

Omar Ashraf Mabrouk Abdelwahab 19P8102

Abdel Rahman Mohamed Salah El Din 19P9131

Mahmoud Mohamed Seddik Hassan ElNashar 19P3374

Contents

1. Perception (Pipeline phases):

1. Calibration and perspective transform:

We modified the perception.py file where we added the calibration method in the percept transform function where the function cv2.getperceptiontransform() is applied that takes the input of four input vertices and four output vertices and creates a mask the maps the input vertices to the output the four input vertices are chosen by the analysis of a grid image obtained through calibration images provided with the project start-up code.

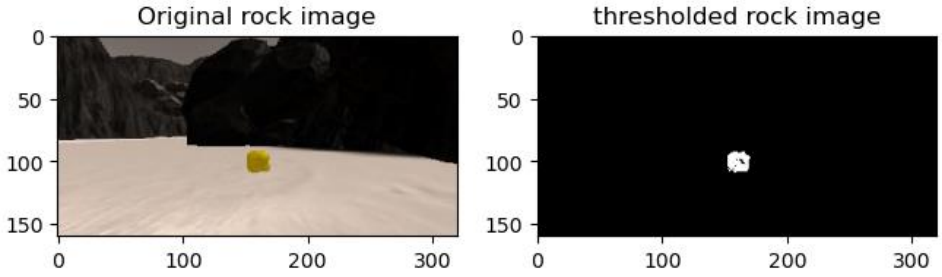


Figure 1 input image of a rock and applying threshold on it

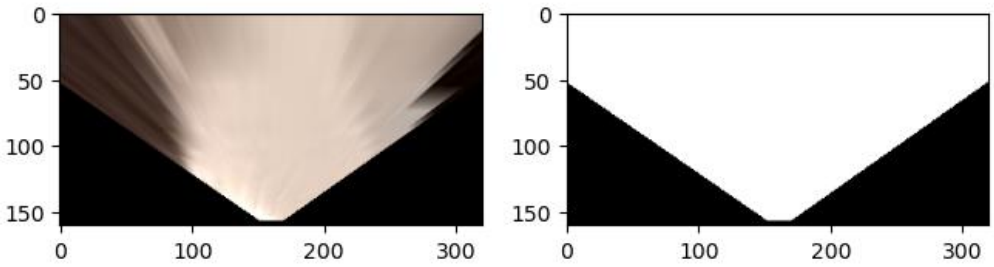


Figure 2 output bird eye view of input image with additional mask for obtaining the obstacle map from it

2.Thresholding:

The bird view of the image is then passed on a thresholding function, this function compares between RGB values of all pixels forming the image and the values of colour of the terrain thus producing a black and white image of the bird eye view where white region represents navigable terrain, a variant of this function is made to threshold for rocks and the output could be used to determine obstacles too through subtracting the threshed terrain image from the mask obtained in Figure 2

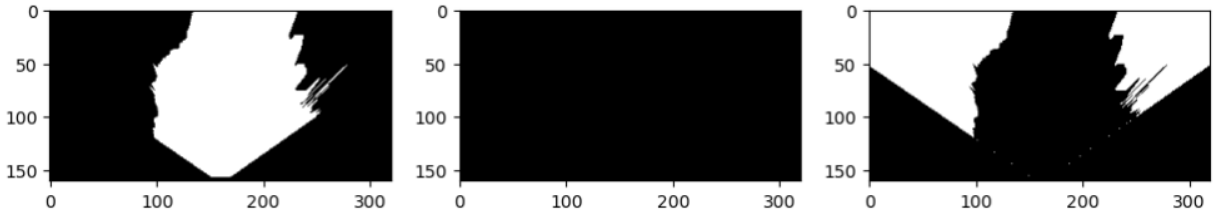


Figure 3 Threshed terrain *Threshed rock map* *Threshed obstacle map*

3. Limit the view for better mapping:

We then limit the view of the mapping of threshed terrain and obstacle map because the camera distorts far objects. This way we could achieve higher fidelity.

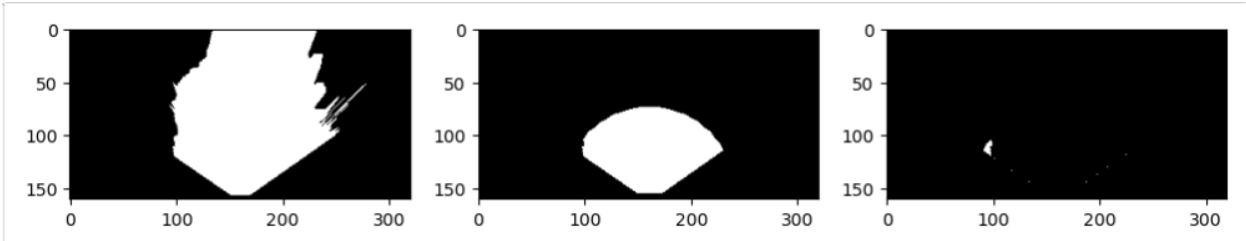


Figure 4 Trimmed view for mapping

4. Coordinate transformations:

At this stage in the pipeline, it is required to position the image received from the rover in terms of the position of the rover on the map so we need to modify the orientation of the shape so that it collides with the x coordinate of the rover then we will rotate the output image to be in a direction parallel to the x-axis but not necessary on it so we need to translate it back to origin.

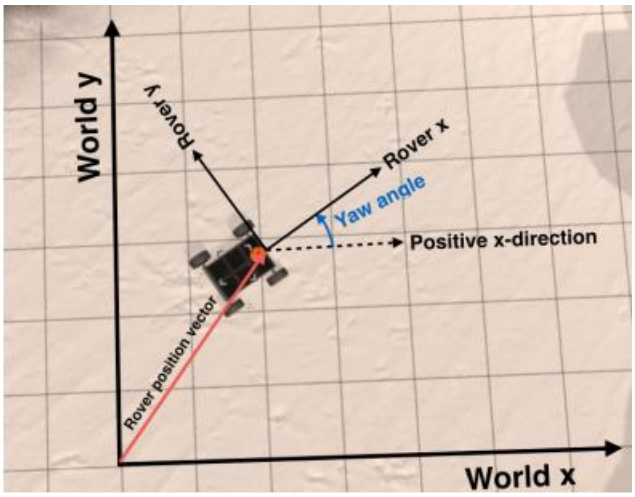
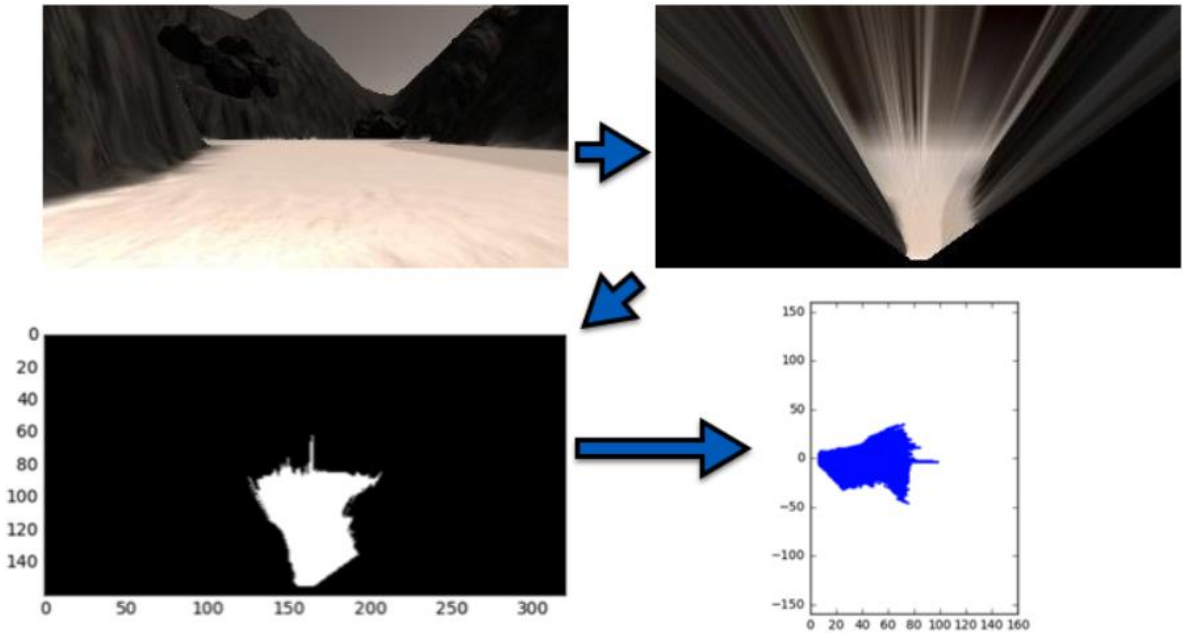


Figure 5 Bird eye view of the rover



4.1 Rover coordinates transformation:

We need to transform the image to be at the origin of our coordinates

From the axes the image is at the centre of the y-axis (horizontal axis) and at max of x-axis (vertical axis) so we will use a translation matrix

$$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -n_x \\ 0 & 1 & -\frac{n_y}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

to bring the rover centre to the origin.

After that we will apply a rotation matrix of angle -90 degrees to make x-axis the horizontal axis and y-axis the vertical axis so that it conforms with the rover coordinates system.

$$\text{Rotation matrix} \begin{bmatrix} \cos(\emptyset) & \sin(\emptyset) & 0 \\ -\sin(\emptyset) & \cos(\emptyset) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

By multiplying both matrices we get a transformation matrix

$$\begin{bmatrix} 0 & -1 & \frac{n_y}{2} \\ 1 & 0 & -nx \\ 0 & 0 & 1 \end{bmatrix}$$

that rotates given image around x-axis which is the vertical axis thus another so to be on the horizontal axis (x-axis of the rover) then we will apply another rotation matrix of angle -90 degrees

$$\text{Rotation matrix} \begin{bmatrix} \cos(\emptyset) & \sin(\emptyset) & 0 \\ -\sin(\emptyset) & \cos(\emptyset) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Thus, total translation matrix will be left multiplying the rotation matrix with the previous transformation matrix resulting in

$$\begin{bmatrix} -1 & 0 & n_x \\ 0 & -1 & \frac{n_y}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

So by multiplying the total transformation matrix with a vector of x (horizontal distance from vertical axis) and y (vertical distance from horizontal axis) we get

$$x_2 = -x_1 + n_x$$

$$y_2 = -y_1 + \frac{n_y}{2}$$

4.2 Polar coordinates:

Magnitude of a vector $=\sqrt{x^2 + y^2}$

To get angle use $\arctan(\emptyset) = \frac{y}{x}$

This will be used later to get the angle at which the rover will rotate with

4.3 Pixel to world function:

This function incapsulate a number of transformations as rotation then translation with a scale then clipping this is used when transforming output image after converting it into the rover coordinates

2. Main pipeline:

1. Run perspective transform on the input image(Rover.img) so to get the bird eye view of the image
2. Then the output image is threshold using the colour of the navigable space to obtain navigable region

3. The obstacles region is then obtained by subtracting ones from output binary image while applying absolute function to disallow negative values and then multiplying it by the additional mask created from the function to ensure that the obstacles generated are from the bird eye view from image and not noise
4. Apply image trimming to remove distorted parts from image mapping
5. Then the navigable region is converted the rover coordinates
6. From the posx,posy,yaw angle obtained from the rover class the pixels of navigable terrain is there respective position in relation to the origin rather than relative to rover
7. And obstacles are then mapped the same way
8. Both of navigable terrain and obstacles is put on the world map where each is put in a respective colour channel
9. Conflicts between terrain and obstacles are resolved prioritizing the navigable terrain over the obstacles
10. Then the rocks are found through thresholding by a RGB value then converted their pixels converted to the rover coordinates then to the world coordinates by using posx,posy,yaw angle
11. It appears as a bright colour where all colour channels are set to max value

---

3. Decision file

This file represents the brain of the rover where it takes the information from the processed image and makes decisions based on it. This file initially contained two modes: forward and stop, we introduced two other modes stuck and rock.

3.1 Forward mode

In case of the forward mode, we applied a small change to the navigation direction calculation. It was stated in the description that the rover exhibits better behaviour when near the walls so we added a term to the navigation direction to make it a wall crawler. We added a factor of the standard deviation of the navigation angles from the perception step. One might wonder why the standard deviation. This is simply because in case of narrow hallways we need the deviation to

be small while in case of open wide regions we need the deviation to be large so as to stick to a wall without continuously hitting it.

```
Rover.steer = np.clip(np.mean((Rover.nav_angles-(wall_side*offset)) * 180 /
np.pi), -15, 15)
```

3.2 Rock mode

We enter to the rock mode from the main forward mode if the sample_angles exist (ie: the rover has seen a rock) in this part we check if the rock is on the same side of the wall we are adhering to and the navigation pixels to it are within allowed limit the we enter the rock mode. We also start a timer that if elapsed pops up the rock mode from the mode stack so that we don't get stuck while trying to pick a rock.

```
if Rover.samples_angles is not None and
wall_side*np.mean(Rover.samples_angles * 180/np.pi) < (Rover.max_rock_angle)
\
        and np.min(Rover.samples_dists) < Rover.max_rock_distance:
    # Rover.steer = np.clip(np.mean(Rover.samples_angles * 180 / np.pi), -15,
15)
    Rover.rock_time = Rover.total_time
    Rover.mode.append('rock')
```

In the rock mode we calculate the navigation angle of the rock through averaging its angles from the perception step.

If this angle is found then the rover still sees the rock else then it became out of the rover sight and we shift back to the previous mode through popping the rock mode from the mode stack.

```
elif Rover.mode[-1] == 'rock':
    # Steer torwards the sample
    mean = np.mean(Rover.samples_angles * 180 / np.pi)
    if not np.isnan(mean):
        Rover.steer = np.clip(mean, -15, 15)
    else:
        Rover.mode.pop() # no rock in sight anymore. Go back to previous
state
```

Then we check on the rock timer if it has elapsed then we resume to the old mode and ignore this rock to avoid getting stuck in this mode. (it is worth noting that all the timers in the decision file are applied using variables that are set to the current time of the rover when initialized and to determine the timer value we subtract the current time from this variable)

```
# if 20 sec passed gives up and goes back to previous mode
if Rover.total_time - Rover.rock_time > Rover.rock_time_max:
    Rover.mode.pop()  # returns to previous mode
```

Then we check if we are near a sample then we break and set throttle to zero to allow for pick up

```
# if close to the sample stop
if Rover.near_sample:
    # Set mode to "stop" and hit the brakes!
    Rover.throttle = 0
    # Set brake to stored brake value
    Rover.brake = Rover.brake_set

# if got stuck go to stuck mode
elif Rover.vel <= 0 and Rover.total_time - Rover.stuck_time >
Rover.rock_stuck_time_max:
    Rover.throttle = 0
    # Set brake to stored brake value
    Rover.brake = Rover.brake_set
    Rover.steer = 0
    Rover.mode.append('stuck')
    Rover.stuck_time = Rover.total_time
else:
    # Approach slowly
    slow_speed = Rover.max_vel / 2
    if Rover.vel < slow_speed:
        Rover.throttle = Rover.approch_rock_throttle
        Rover.brake = 0
    else:  # Else break
        Rover.throttle = 0
        Rover.brake = Rover.brake_set
```

3.3 Stuck mode

This mode is the easiest as in case you get stuck you simply stop the rover make it turn around and then move to another mode

```
# If we're already in "stuck". Stay here for 1 sec
elif Rover.mode[-1] == 'stuck':
    # if 1 sec passed go back to previous mode
    if Rover.total_time - Rover.stuck_time > 1:
        # Set throttle back to stored value
        Rover.throttle = Rover.throttle_set
        # Release the brake
        Rover.brake = 0
        # Set steer to mean angle
        # Hug left wall by setting the steer angle slightly to the left
        Rover.steer = np.clip(np.mean((Rover.nav_angles-(wall_side*offset)) *
180 / np.pi), -15, 15)
        Rover.mode.pop() # returns to previous mode
    # Now we're stopped and we have vision data to see if there's a path
forward
    else:
        Rover.throttle = 0
```

```
        # Release the brake to allow turning
        Rover.brake = 0
        # Turn range is +/- 15 degrees, when stopped the next line will
induce 4-wheel turning
        # Since hugging left wall steering should be to the right:
        Rover.steer = wall_side*15
```

You can get into stuck mode through forward mode if you velocity is low and you are stuck in the same place for more than the max time set for stuck.

```
# Alternates between stuck and forward modes
if Rover.vel <= 0.1 and Rover.total_time - Rover.stuck_time >
Rover.stuck_time_max:
    # Set mode to "stuck" and hit the brakes!
    Rover.throttle = 0
    # Set brake to stored brake value
    Rover.brake = Rover.brake_set
    Rover.steer = 0
    Rover.mode.append('stuck')
    Rover.stuck_time = Rover.total_time
```

You can also enter stuck mode from the rock mode because as we said earlier if the rock timer has elapsed it will go to stuck mode and ignore this rock to avoid wasting time.

```
elif Rover.vel <= 0 and Rover.total_time - Rover.stuck_time >
Rover.rock_stuck_time_max:
    Rover.throttle = 0
    # Set brake to stored brake value
    Rover.brake = Rover.brake_set
    Rover.steer = 0
    Rover.mode.append('stuck')
    Rover.stuck_time = Rover.total_time
```

---

4. Hyperparameters

To allow for more freedom in tuning the performance of the rover all the tunable hyperparameters were put in the rover file inside the rover class. They are listed below.

```
self.stuck_time_max = 5
self.rock_time_max = 20
self.rock_stuck_time_max = 20
self.throttle_set = 0.5  # Throttle setting when accelerating
self.brake_set = 10  # Brake setting when braking
self.stop_forward = 50  # Threshold to initiate stopping
self.go_forward = 250  # Threshold to go forward again
self.max_vel = 2  # Maximum velocity (meters/second)
```

```
self.wall_side = 1  # 1 for right wall -1 for left wall
self.offset_weight = 0.8  # weight of the std to be added to the nav
direction -> hyperparameter to be manipulated later
self.approch_rock_throttle = 0.5
```

```
# Rock Max distance and angle
self.max_rock_distance = 50
self.max_rock_angle = 20  # --> in degree
```

```
# Perception file
# Threshold for the navigable terrain
self.red_threshold = 180  #
self.green_threshold = 180  #
self.blue_threshold = 160
# limiting the view of the rover
self.decision_mask_size = 8  #
self.mapping_mask_size = 6
# set limits for pitch roll and steering for update
self.steer_update_limit = 5
self.pitch_update_limit = 1
self.roll_update_limit = 1
```