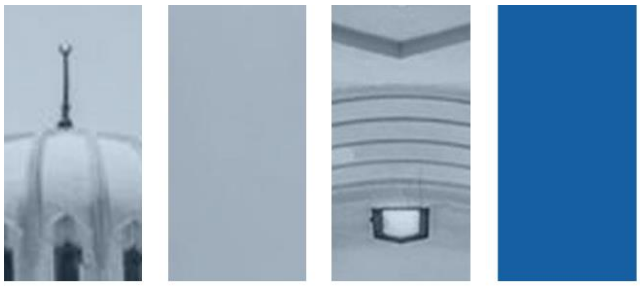


Program: Computer and Systems Engineering, Senior-1 Specialized Programs

Course Code: CSE374s

Course Name: Digital Image Processing

Digital Image Processing Project Phase 2

Ain Shams University
Faculty of Engineering
Fall Semester – 2022

Presented to
Prof. Mahmoud Khalil

Name	ID	Level	Program
Abdelrahman Sherif Fayez Ali	1900773	Senior-1	UGCSE-2018
Ahmed Mahmoud Mohamed Ibrahim	1901143	Senior-1	UGCSE-2018
Mazen Ehab Mohamed Maher	1901120	Senior-1	UGCSE-2018
Andrew Samir Kamel Gayed	1900242	Senior-1	UGCSE-2018
Mario Samy Aziz	16P3102	Senior-2	UGCSE-2013

2.1) Pipeline of Perception Step:

2.1.1) size of input image (from camera) is 160*320 pixels:

160 pixels vertically, 320 pixels horizontally

2.1.2) Define source and destination points for generating mask which will helps to get perspective transform. The destination points will be 10 pixels* 10 pixels, but we know that the size of grid 1pixel *1 pixel:

So, we need scale to be 10 --> but we did not make this scale as this will decrease fidelity!

we choose scale between 15 and 20 so that we have good fidelity (pixel will be represented in less than one pixel)

2.1.3) Apply perspective transform of camera image using perspect_transform and called "the result warped":

warped is image representing plane of camera image (top view)

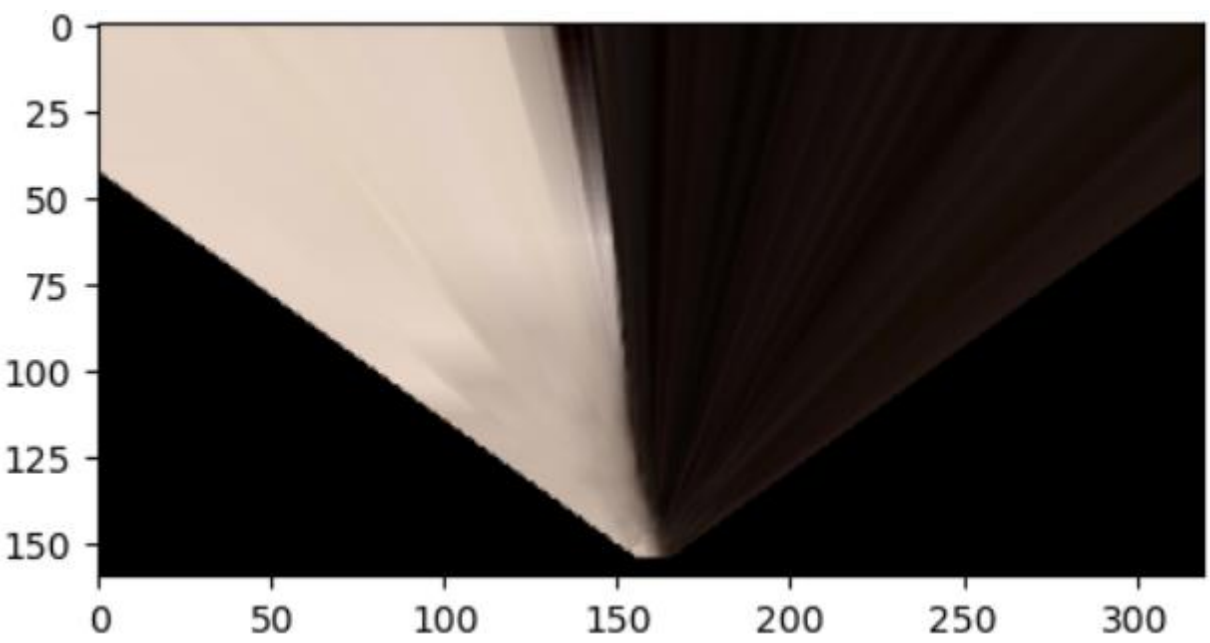
--> image like camera image but represent top view

"perspect_transform" is a function that takes 3 paraments:

parameter 1: image that we want to get perspective for it (camera image)

parameter 2: source points

parameter 3: destination points



2.1.4) Take copy of wrapped image using np.copy and called it roi:

roi --> perspective transform of camera image

copy is function from np library take image or part of image

ex: copy(image) = copy(image[:,:]) --> copied image is identical to original image

ex: copy(image[130:140,150:170]) --> copy from image from 130 to 140 (vertically)

--> copy from image from 150 to 170 (horizontally)

--> copied image is 10 pixels in height and 20 pixels in width

2.1.5) Apply colour threshold on wrapped image:

we apply colour threshold (on wrapped image) three times with different threshold values to identify navigable terrain/obstacles/rock samples in top view image.

we apply colour threshold (on wrapped image) with values 90, 180, 160 to identify navigable terrain in top view image and called result threshedroi image.

we apply colour threshold (on wrapped image) with values 185, 140, 15 to identify rock samples in top view image and called result tgt_img image.

we apply colour threshold (on wrapped image) with values 100, 100, 100 to identify obstacles in top view image and called result obs_img image.

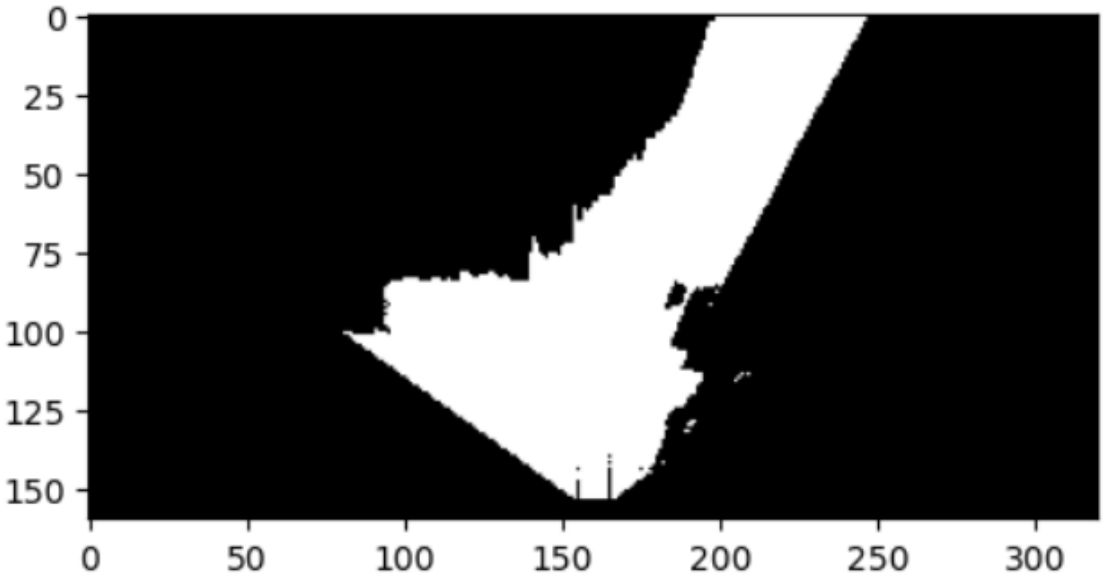
we get colour threshold using color_thresh function

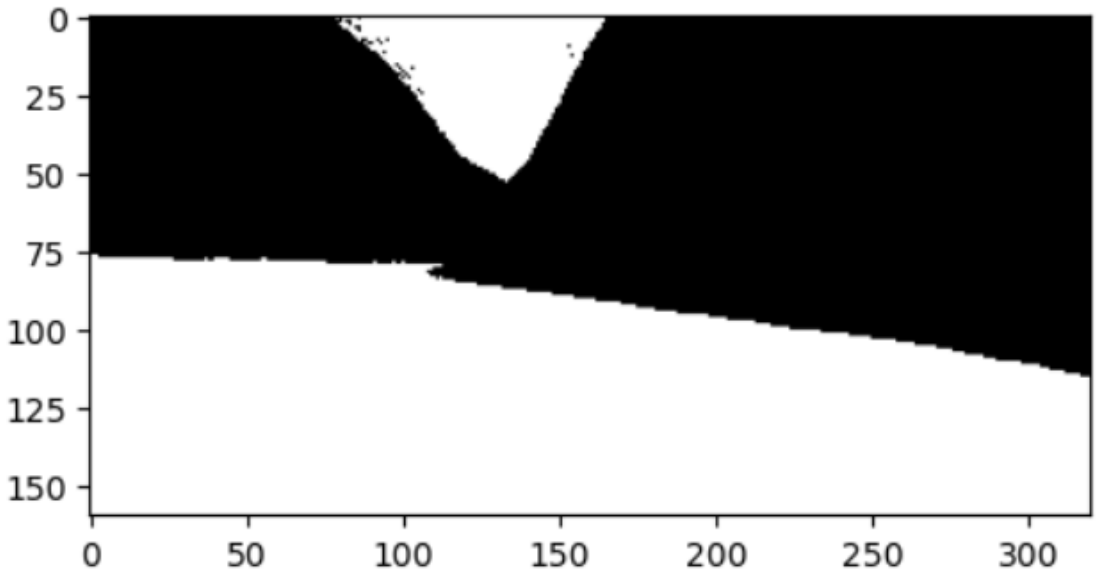
“color_thresh” is a function that takes 3 parameters:

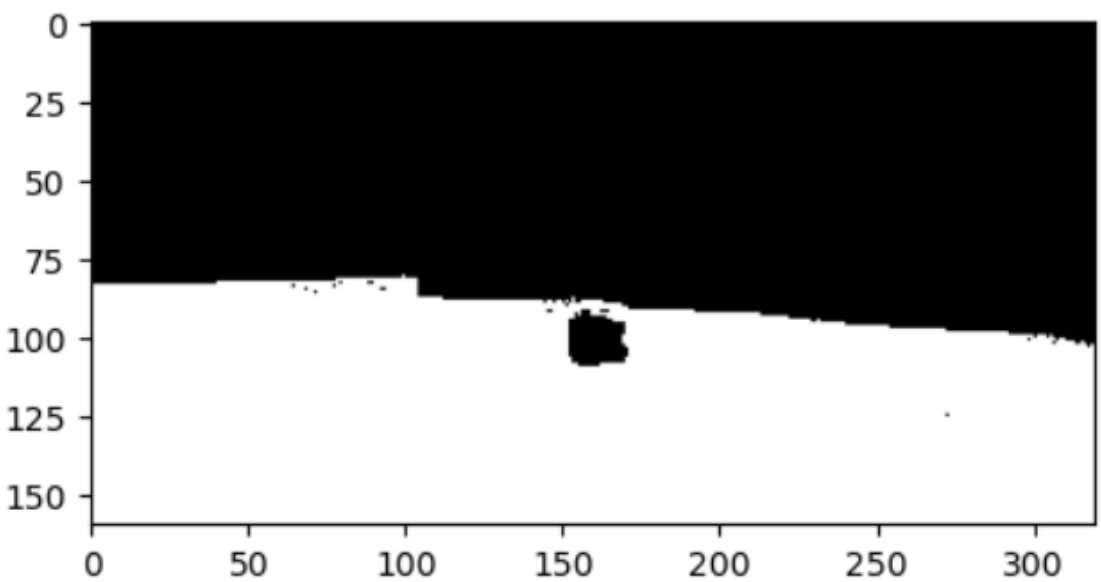
parameter 1: image that we want to apply threshold on it (wrapped image)

parameter 2: tuple of three elements.each element represents threshold value for one channel of 3 channels

parameter 3: Boolean represent we apply threshold for identify rock samples


0
25
50
75
100
125
150
0
50
100
150
200
250
300


0
25
50
75
100
125
150
0
50
100
150
200
250
300


0
25
50
75
100
125
150
0
50
100
150
200
250
300

2.1.6) we need to prevent collision by looking two meters square in front of rover

as each meter square is represented by 10*10:

note: we do not apply scaling now (we know that).

our idea is the following:

a- take copy of part of wrapped image (from 130 to 150 (vertically) and from 150 to 170 (horizontally)) which represent 2 m^2 in front of rover

using np.copy and called it **coll_roi**

coll_roi --> part of perspective transform of camera image which represent 2 m^2 in front of rover

b- applying not operation on coll_roi (part of perspective transform of camera image which represent 2 m^2 in front of rover).

we apply not operation using cv2.bitwise_not (bitwise_not is function from cv2 library take one parameter). parameter: image that we want to not operation on it (part of perspective transform of camera image which represent 2 m^2 in front of rover).

c- applying thresholding to identify obstacles from coll_roi (part of perspective transform of camera image which represent 2 m^2 in front of rover) and called the result **color_thresh.**

2.1.7) Finding edge pixels between the sand and the wall using get_contours function:

“get_contours” is a function that takes 2 parameters which is used to get array of arrays (each inner array contain pixels for a contour in image)

parameter 1: wrapped image

parameter 2: threshold values of navigable terrain for determining sand / not sand

“get_contours” is a function that returns 3 variables

variable 1: binary image that identify navigable terrain in top view image

variable 2: binary image of wrapped

variable 3: array of arrays represents pixels of each contours

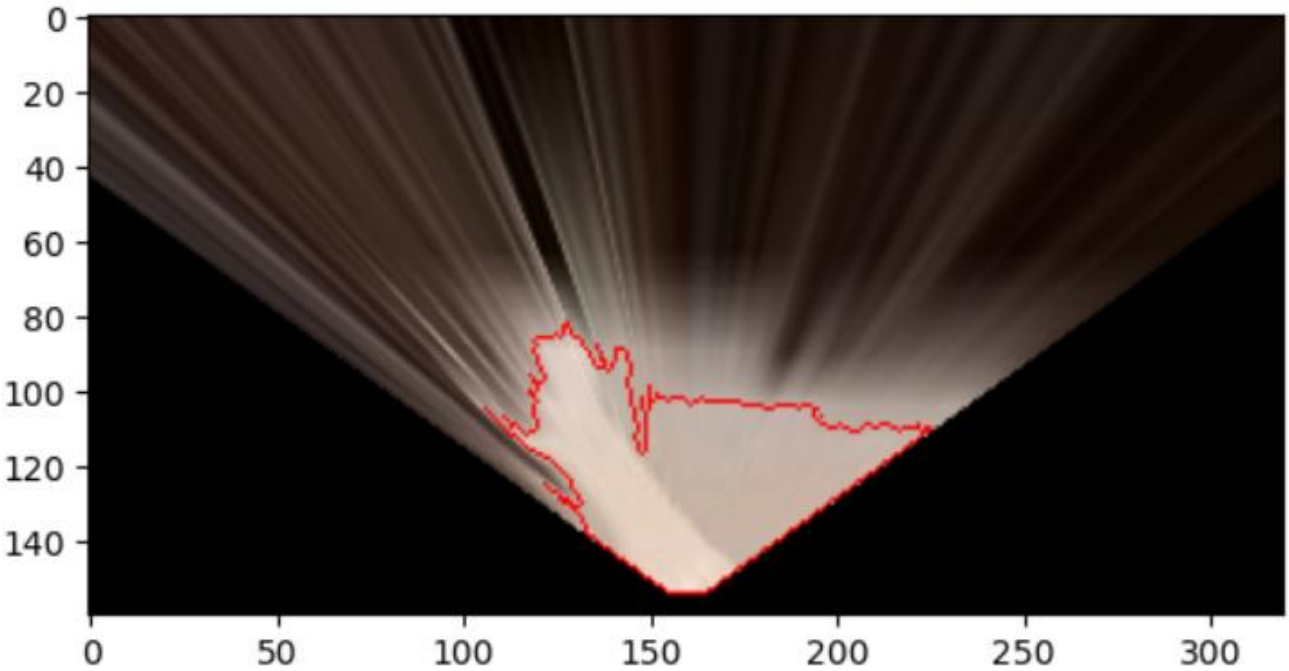
we find contours in wrapped image using cv2.findContours

“findContours” is function of cv2 library that takes 3 parameters:

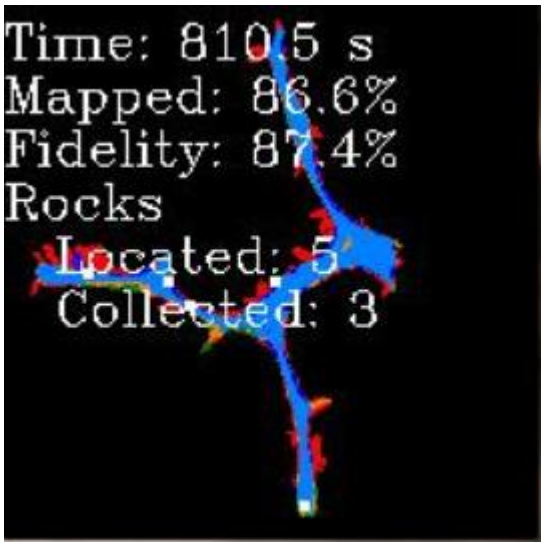
parameter 1: copy of binary representation of image that get want to find its contouts

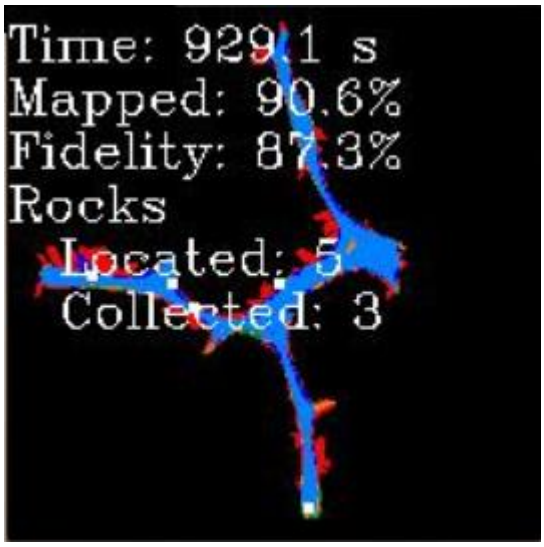
parameter 2: cv2.RETR_TREE

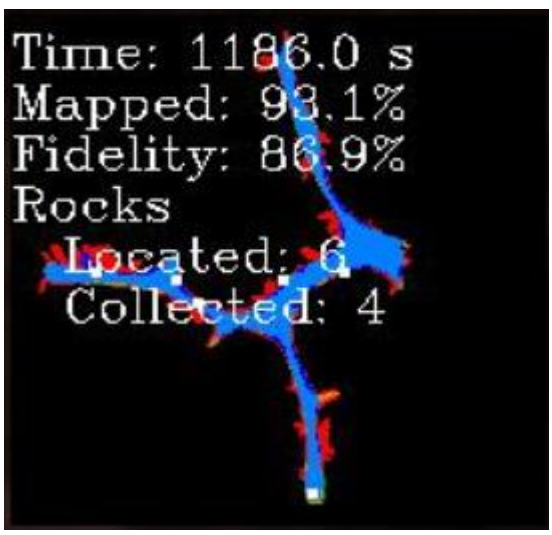
parameter 3: CHAIN_APPROX_NONE --> get all points without compression/ extrapolation



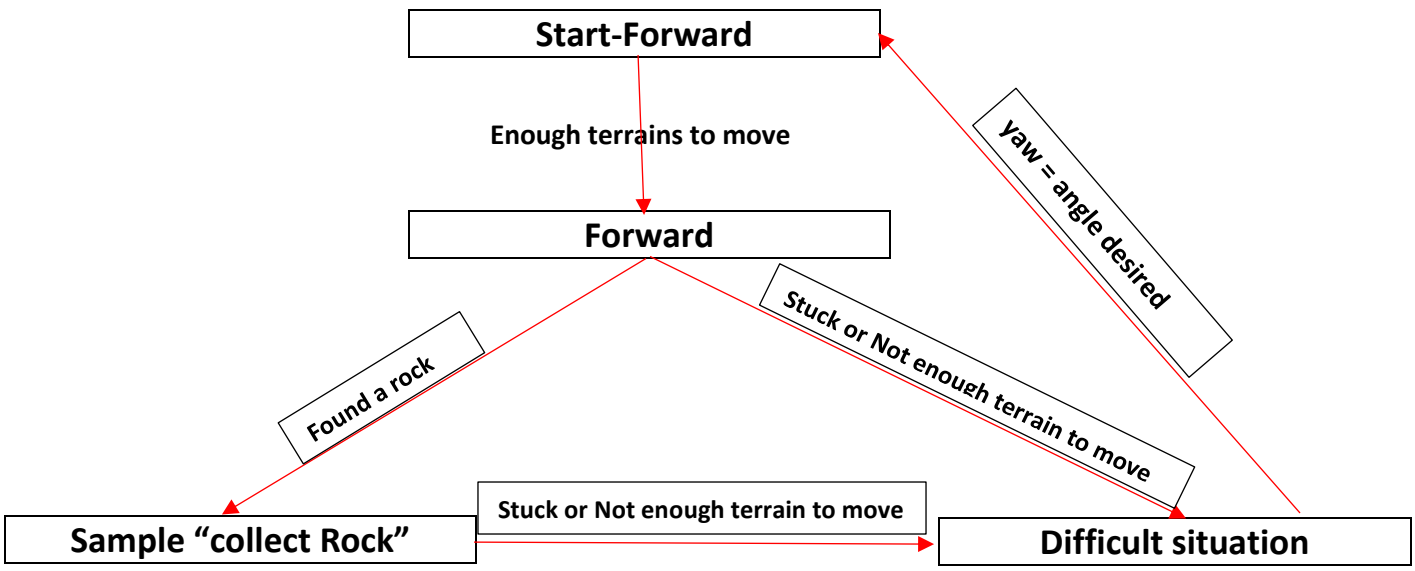
2.1.8) Update Rover worldmap (to be displayed on right side of screen and # update an image to include our navigation data on HUD and draw the entire contour on imgwcontour







2.2) Pipeline of Decision Step:



GitHub Link

https://github.com/Abdelrahman-Sherif-Fayez/Nasa_Mars_Rover_Project.git