e-Yantra Robotics Competition Plus
(eYRC+ Pilot)
eYRC+#246

Team leader name	**Shyam Gupta**
College	**Malviya National Institute of Technology,Jaipur**
e-mail	**rockysharmacool123@gmail.com**
Date	**24/12/2014**

Scope of the Task (7)

- Describe the algorithm used for solving path planning in this task.

<Teams should write in their own words a description of algorithms used in this task.
You can also draw some diagrams/figures, flowcharts to illustrate the algorithm used.
Answer format: Text
Word-limit: 100 words>

1. We use dijkstra algorithm for finding the shortest path between start point and end point.
   Dijkstra algorithm:
   - In this we take various destination as graph nodes and path between various nodes as edge of graph.
   - Initially we define two sets, set A and set B.
   - Set A contains nodes and edges of processed graph and set B contains unprocessed graph.
   - Initially set A is null and set B contains whole graph
   - We take start point from B and move it to set A and set start point equal to zero.
   - Now we pick up an edge going from set A to set B such that we select minimum value of its length plus node value in set A and assign this value to the node in set B and move this node to A and repeat this process until we reach the destination point.

   In image we consider centroid of a cell as a node of graph and define edge between two centroids of length 1.
   Black cell ignored and no edge is defined connecting black cell to any other cell.
   We start from red cell as a starting point and apply algorithm until we reach green cell.
   For selecting minimum value of node length + node value we use heap algorithm defined in class EyantraHeap.
   Class Track is used to store value of a node, minimum distance from starting node and another node coordinate connecting the edge between these nodes and the edge between these nodes is a part of the shortest path.

Camera and Image Processing (3)

Write down the answers to the following questions. For this part use first image (*test_image1.png*) in *"Task2_Practice/test_images"* folder.

- What is the resolution (size) of the test image?
- What is the position of the Start point and the End point in the grid in the test image? (Please refer to the *Task2_Description.pdf* for the definitions of Start point and End point and answer in (x,y) form, where the x-axis is oriented from left to right and the y-axis is oriented from top to bottom)
- Draw four shortest paths from the Start point to the End point (you may draw it manually if you desire). An example is shown below:

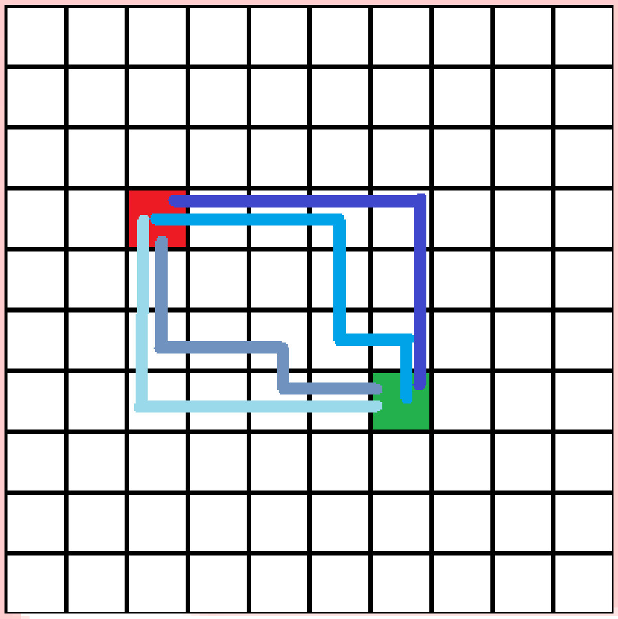


Figure: Example solution with four shortest paths drawn

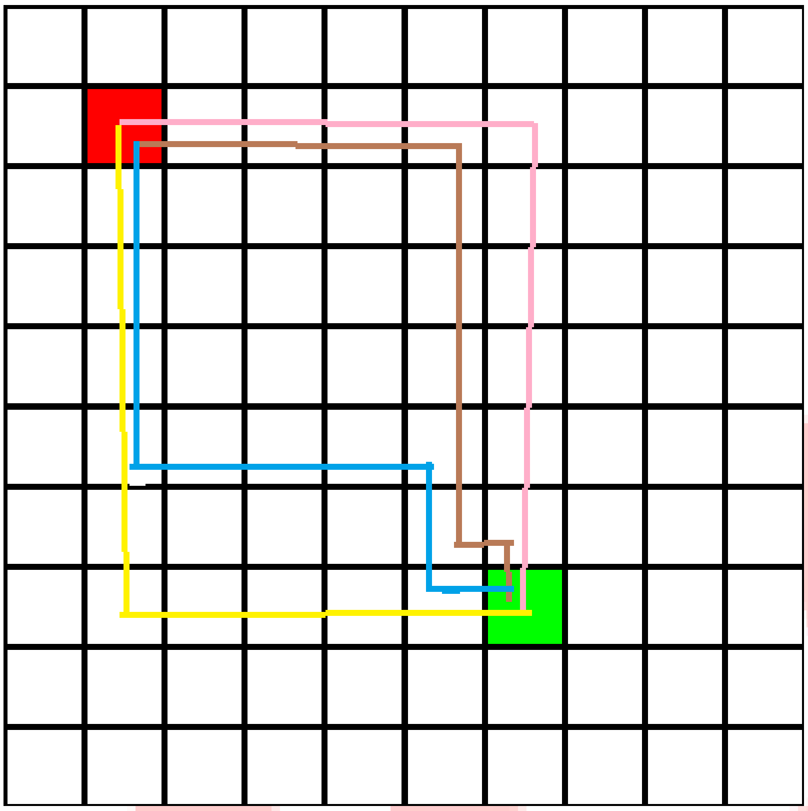
<

Answer format:

- Answer to question 2 in bulleted form
- Answer to question 3 in bulleted form
- Image pasted for question 4

>

- 400 X 400 pixels
- Start Point=(2,2) and End Point=(7,8)

- 

Software used (10)

Write down the answers to the following questions. For this part use first image in *"Task2_Practice/test_images"* folder.

- Write a function in python to open the image and return an image with a grid of **n** equally spaced horizontal and vertical red lines(RGB values (255, 0, 0)). You are required to write a function *draw_grid(filename,n)* which takes two arguments:
  a. filename: color image
  b. n: number(integer datatype) of equally spaced horizontal and vertical lines

Output of program should be the image with the specified red grid drawn on it.
<Answer format:
Use the snippet given below by adding your code after the comment: Inline comments are mandatory to explain the code>

```
def draw_grid(filename,n):

    filename-- input color image stored as file
    n-- integer from 1 to 10
    returns img-- the image with the red grid (having specified number of
    lines) drawn on it
    '''
    #add your code here
```

```
## reading the image and storing in img variable
        img = cv2.imread(filename)

        for i in range(0,n+1):
                ##draw n equally spaced horizontal lines on image of red colour
                cv2.line(img,(0,(40*i)),(40*(n),(40*i)),(0,0,255),3)

                ##draw n equally spaced vertical lines on image of red colour
                cv2.line(img,((40*i) ,0),((40*i) ,40*(n)),(0,0,255),3)

## return image with grids formed by these vertical and
horizontal lines
        return (img)
```

- Write a function `space_map(img)` in python to detect the layout of the grid as shown in the test image (Figure 1) below. Function `space_map(img)` takes a test image as input and returns a 10x10 matrix called “`grid_map`” of integers with values either 0 or 1. Each square must be identified as either navigable space(`0`), or obstacle(`1`). The Start and End points are considered as obstacles for this question. An example is shown in Figure 2 below.

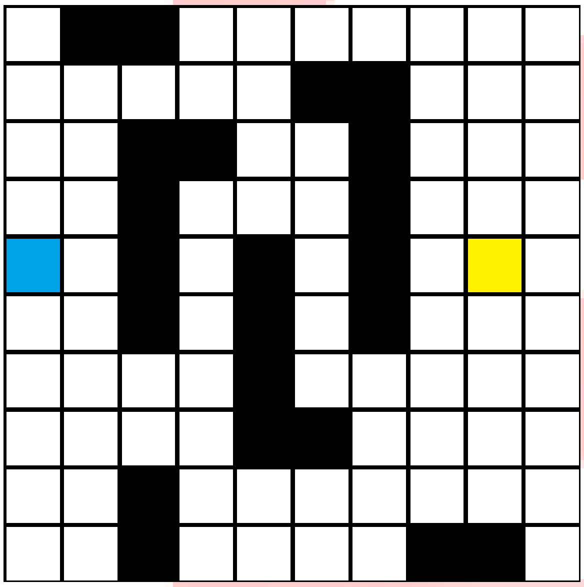
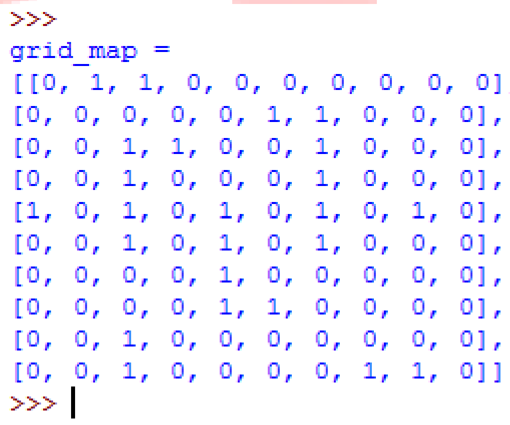


Figure 1: Example Test Image



```
>>>
grid_map =
[[0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
[0, 0, 1, 1, 0, 0, 1, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
[1, 0, 1, 0, 1, 0, 1, 0, 1, 0],
[0, 0, 1, 0, 1, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 1, 1, 0]]
>>>
```

Figure 2: Example output

<Answer format:
Use the snippet given below by adding your code after the comment: `#add your code here`.
Inline comments are mandatory to explain the code>

```
def space_map(img):
    '''
    img-- input color image stored as file
    result— output binary image
    '''
    #add your code here
```

```
##define lower and upper parameters
param1 = [0,0,0]
param2 = [255,255,255]

##converting the image to HSV form and storing it in variable named hsv
hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

##converting the parameters into the form opencv can understand
lower = np.array(param1)
upper = np.array(param2)

##masking the hsv image using the two parameters
 mask  = cv2.inRange(hsv, lower, upper)

##bitwise anding the image with mask and storing result in res variable
 res   = cv2.bitwise_and(img, img, mask= mask)

##converting resulted image to gray
 gray = cv2.cvtColor(res,cv2.COLOR_BGR2GRAY)

##defining threshold value as 240 and storing resultant image in thresh1
variable
 ret,thresh1 = cv2.threshold(gray,240,255,cv2.THRESH_BINARY)

##drawing vertical and horizontal lines on thresh1 image
 for i in range(0,11):
     cv2.line(thresh1,(0,(40*i)),(400,(40*i)),(255,255,255),3)
     cv2.line(thresh1,((40*i) ,0),((40*i) ,400),(255,255,255),3)

## finding the contours of thresh1
contours, hierarchy=
cv2.findContours(thresh1,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)

## defining an array cb to store coordinates of obstacles
 cb=[]

## find the centroids of contours
 for i in range(1,len(contoursb)):
          M = cv2.moments(contoursb[i])
          cxb=int(M['m10']/M['m00'])
          cyb=int(M['m01']/M['m00'])
          cv2.circle(img,(cxb,cyb), 5, (255,0,0), -1)
          cb.append((cxb/40,cyb/40))

## define a list grid_map to store the values of grid
 grid_map=[]
 for i in range(10):
     abc=[]
     for j in range(10):
          abc.append(0)
     grid_map.append(abc)

 for i in range(10):
     for j in range(10):
          if (i,j) in cb:
               grid_map[j][i]=1

## return the 10 X 10 matrix of 0 or 1
 return grid_map
```