e-Yantra Robotics Competition Plus
(eYRC+ Pilot)
eYRC+#1673

Team leader name	**Dheeraj Kamath**
College	**B.M.S. College of Engineering, Bangalore**
E-mail	**dheeraj.kamath@yahoo.com**
Date	**15-12-14**

SCOPE OF THE TASK

1. ALGORITHM USED

A set of images are given, each with a grid of 10x10. The aim of the task is to find the shortest path between the start and the end cell.

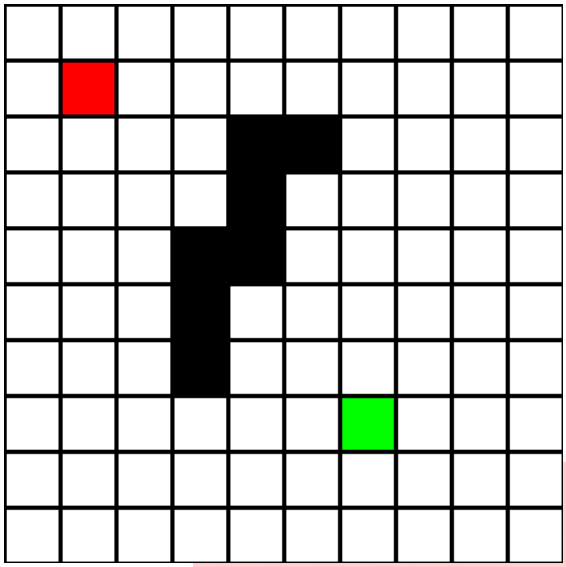
Firstly, the start cell (red), the end cell (green) and the obstacles (black) are determined and a 2D array representing the grid layout is created (which is 1 for a white space and 0 for an obstacle).

Then, **A* Algorithm** is implemented to find the shortest path. A class Node is created whose object is a single node from the grid with properties such as the **node position, parent node information and the values: f (total cost), g (cost from start node to the current node) and h (heuristic value – cost from current node to the end node).**

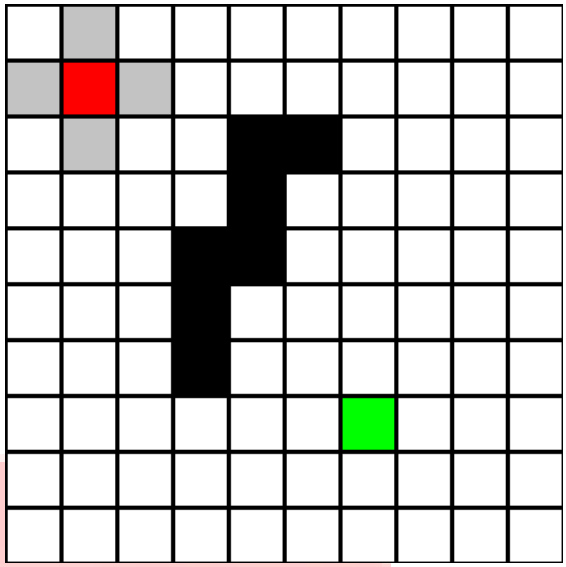
Total Cost, f = g + h

An **open list** is created which is made into a heap **to store the nodes that are processed** already and **the node with the lowest f value is at the top**. At first, the start node is added to this list. The **node with the lowest f value is popped** and its **adjacent nodes are found** out and the **current node is set as the parent node and is added to the closed list**. The adjacent nodes are then processed to get the lowest f value and then added to the open list heap.

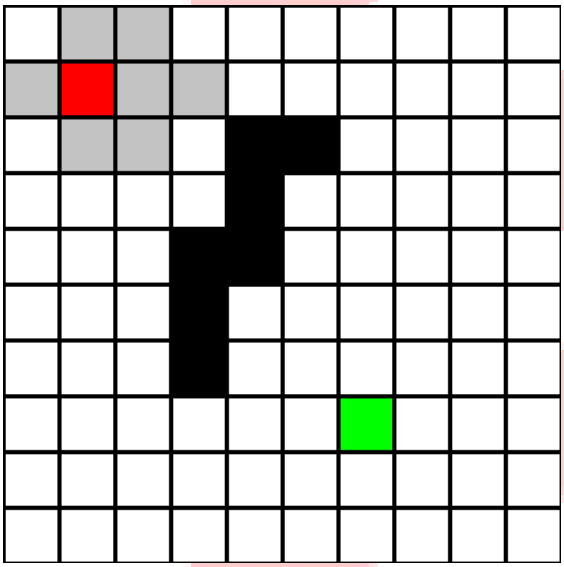
The **algorithm continues in the same manner until the end node is reached**. Then, starting with the end node, the **parent node of the all the nodes is back traced until the start node is encountered**. These set of nodes are then appended to the route_path list. This list is reversed and the shortest path from the start to the end is displayed. Then finally the **length of this list is calculated and the path is drawn** on the output image.



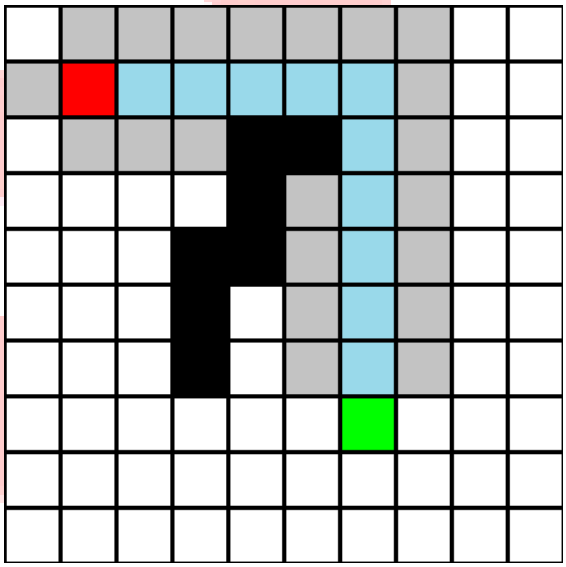
Start node is added to open list



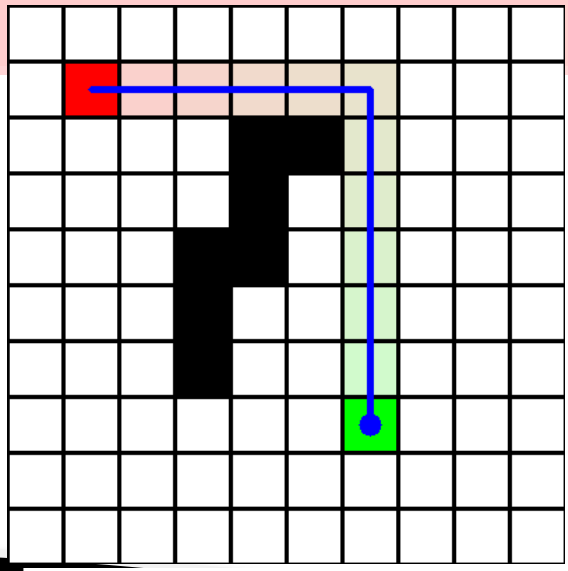
Adjacent nodes are found out



Node with least f value is added to the closed list



Obstacles are avoided and path is determined by back tracing the parent nodes



Final path is drawn on the output image

DATA STRUCTURES USED

HEAP

Heaps are binary trees for which every parent node has a value less than or equal to any of its children. A heap can be thought of as a priority queue where the most important node will always be at the top, and when removed, its replacement will be the most important. In the program, the open list is heapified and stores the node with the least f value at the top.

LIST

A list is a container of any type of similar data types that are organized in order from first to last (starting value of list index from 0). For example, in the program, information about the given image is stored as a list of lists in which each nested list contains the information about a single row of cells from the given image. Here, a wall or start/end cell is assigned the value 0 (False) and a blank space as 1 (True).

CLASS

A class is way to bind data and its associated functions together. In the program, a class Node is created with data/properties such as the position, f, g and h values and parent information about a node. Another class path_algorithm is created which contains data like the open/closed lists and functions to find the shortest path.

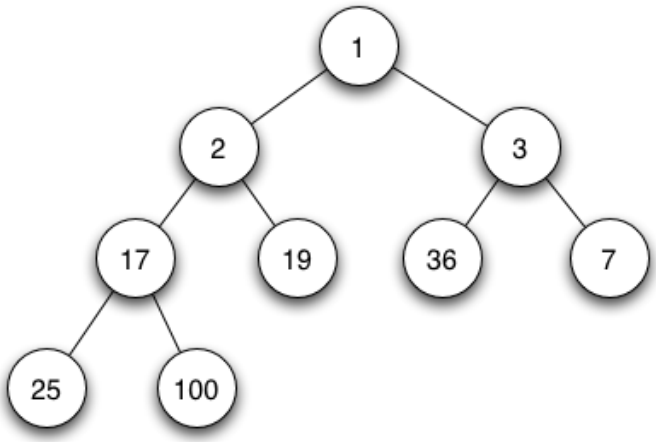


Diagram representing a Binary Heap

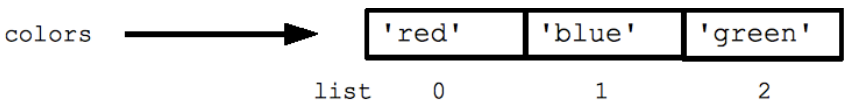


Diagram representing a List in Python

CAMERA AND IMAGE PROCESSING

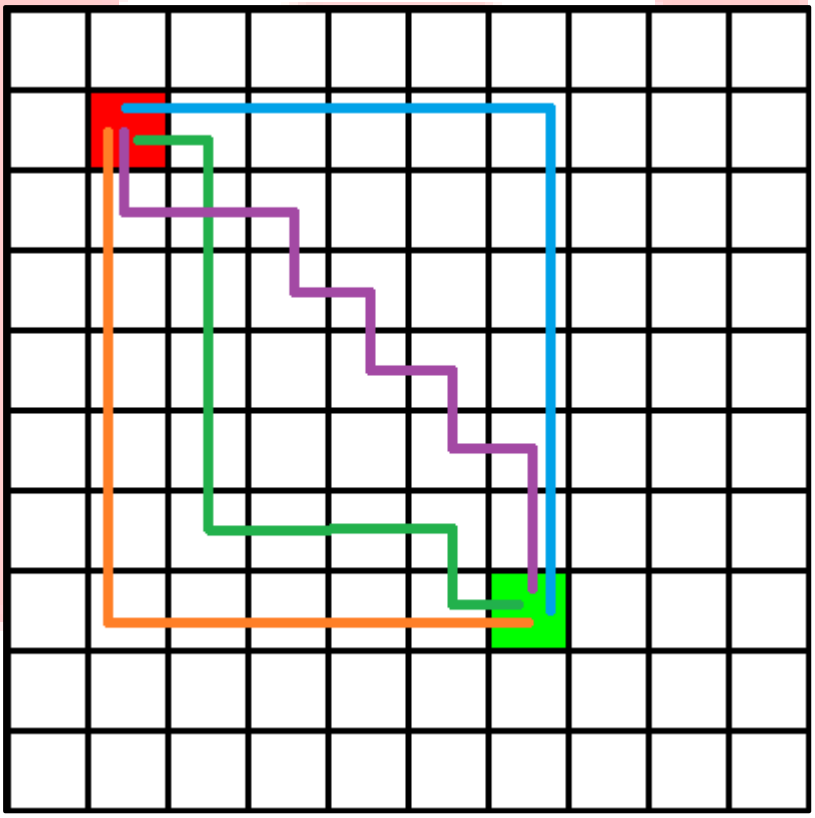
2. What is the resolution (size) of the test image?

400 x 400 pixels (Height x Width) (x3 Channels - RGB)

3. What is the position of the Start point and the End point in the grid in the test image?

Start Point - (2,2)
End Point - (7,8)

4. Draw four shortest paths from the Start point to the End point.

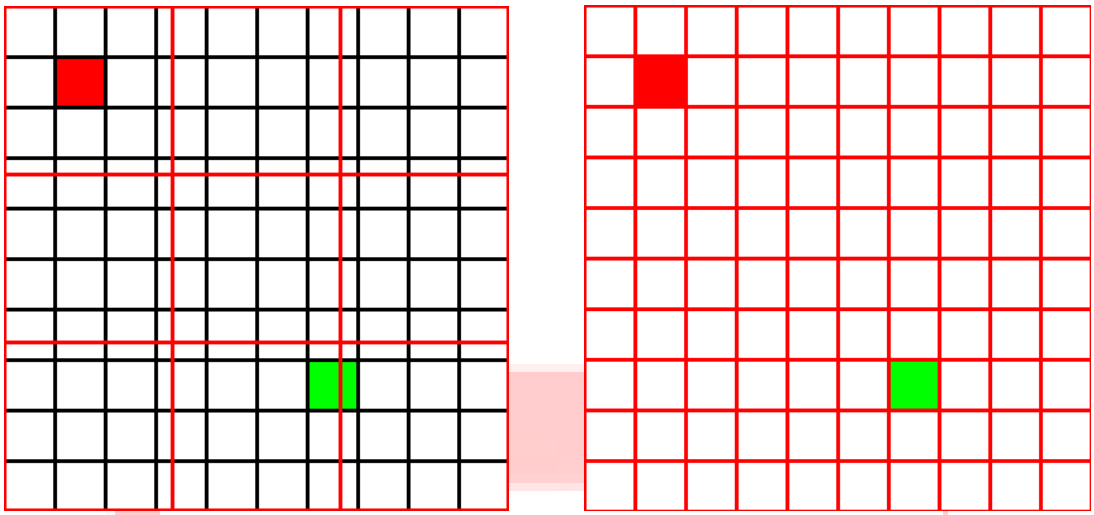


SOFTWARE USED

5. 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.

```
# Import OpenCV and Numpy
import numpy as np
import cv2

# Required Function:
# Function which takes filename -> file name and n -> number of equally
# spaced lines and draws a red grid on the image, img and returns it

def draw_grid(filename,n):

    # Read the image
    img = cv2.imread(filename)

    # To get the size of the image
    h,w,c = img.shape

    # Drawing the grid for n=1
    if(n==1):
        # Drawing the lines
        cv2.line(img,((int)(w/2),0),((int)(w/2),h),(0,0,255),2)
        cv2.line(img,(0,(int)(h/2)),(w,(int)(h/2)),(0,0,255),2)

    # Drawing the grid for n>1
    elif(n>1):
        if(n==2):
            x = w
            y = h
        else:
            x = int(round(float(w)/(n-1))) # Rounding the values
            y = int(round(float(h)/(n-1)))
        # Drawing the lines
        for i in range(0,n):
            cv2.line(img,(x*i,0),(x*i,h),(0,0,255),2)
            cv2.line(img,(0,y*i),(w,y*i),(0,0,255),2)

    return(img) # Return the image

# Set the file name here
filename = 'test_images/test_image1.png'

# To input the value of n, by the user
n = input('Enter n: ')

# Calling the function to draw the grid
img = draw_grid(filename,n)

# Show the image
cv2.imshow('Image',img)

# Close and exit
cv2.waitKey(0)
cv2.destroyAllWindows()
```

SAMPLE OUTPUT



Grid with n = 4 | Grid with n = 11

6. Write a function space_map in python to detect the layout of the grid.

```
# Import OpenCV and Numpy
import numpy as np
import cv2

# Required Function:
# Function which takes an image as its argument and returns the a list
# containing the grid layout (Start, End, Obstacle -> 1, White Space -> 0)

def space_map(img):

    grid_map = []      # Initialise grid_map list
    h,w,c = img.shape # Get dimensions of the image
    y = 20

    # Process all the cells in the image
    for i in range(0,10):
        temp = []
        x = 20
        if(y<h):
            for j in range(0,10):
                if(x<w):

                    # Creating a small ROI to detect the color
                    roi = img[y-2:y+2, x-2:x+2, :]

                    # Convert to HSV image
                    hsv = cv2.cvtColor(roi,cv2.COLOR_BGR2HSV)
                    hue,sat,val,ret = cv2.mean(hsv)
```

```
                    # Check for Obstacles
                    if(sat==255 or val==0):
                        flag = 1

                    # Check for White Spaces
                    elif (sat==0 and val==255):
                        flag = 0
                    temp.append(flag)
                    x = x+40

        # Append the values to the list
        grid_map.append(temp)
        y = y+40

    return grid_map # Return the grid_map list

# Read the image
img = cv2.imread('test_images/test_image1.png')

# Calling the function to get the grid layout
grid_map = space_map(img)

# Displaying the grid layout
print 'grid_map ='
for i in grid_map:
    print i

# Show the image
cv2.imshow('Image',img)

# Close and exit
cv2.waitKey(0)
cv2.destroyAllWindows()
```

SAMPLE OUTPUT

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