Visionaries

Winter School Presentation

AKSHAY S MENON

ISHAAN BAJAJ

SANDRA K JOSEPH

Problem 1
Implementation of path finding algorithms

Task 1.1

- Maze with 0.2 to 0.3 probability of black pixels was generated.
- Two grey pixels were added to indicate the starting point and destination.
- The four algorithms (DFS, BFS, Dijkstra's and A*) were implemented to find the shortest path between the grey pixels.

Maze Generation

```python
import cv2
import numpy as np
import random
import time
from collections import deque

img = np.zeros((20, 20, 3))
m, n, p = img.shape

for i in range(m):
    for j in range(n):
        if random.random() >= 0.25:
            img[i][j] = [255, 255, 255]

grey = 0
while grey < 2:
    a = random.randrange(20)
    b = random.randrange(20)
    neighbouringWhite = 0
    for pos in [[0, 1], [1, 0], [0, -1], [-1, 0]]:

        if 0 <= a + pos[0] < 19 and 0 <= b + pos[1] < 20:
            if (img[a + pos[0]][b + pos[1]] == [255, 255, 255]).all():
                neighbouringWhite += 1

    if neighbouringWhite > 0:
        img[a][b] = [127, 127, 127]
        grey += 1

m, n, p = img.shape


cv2.namedWindow("maze", cv2.WINDOW_NORMAL)
cv2.imshow("maze", img.astype(np.uint8))
# img=pic.astype(np.uint8)
# cv2.imwrite('maze',pic)
cv2.waitKey(0)
```

Setting every pixel as white

The while loop runs until two grey blocks have been placed. A random block is picked and made grey. The nested for loop makes sure that there is at least one white pixel near the spawned grey pixel.

- Generated Maze

DFS

```
39  img1=img
40  yellow=[0,255,255]
41  red = [0, 0, 255]
42  blue = [255, 0, 0]
43  green = [0, 255, 0]
44  white = [255, 255, 255]
45  grey = [127, 127, 127]
46  black=[0,0,0]
47  fro=grey
48  to=grey
49  used=[200,200,20]
50
51
52
53  class node:
54      def __init__(self, ind, root):
55          self.x = ind[0]
56          self.y = ind[1]
57          self.root = root
58
59
60  def show_path(end, start):
61      print('start:(', start.x, start.y,')')
62      print('end:(', end.x, end.y,')')
63
64      present = end
65      while present != start:
66          img1[present.x][present.y] = yellow
67          present = present.root
```

Assigning values to colors

A class named node is created for storing the link to children

Function to show the final path

Code for dfs function

```
def dfs(start):
    q = deque()
    q.append(start)
    cv2.namedWindow('path', cv2.WINDOW_NORMAL)
    while len(q):
        current = q.pop()
        i, j = current.x, current.y
        cv2.imshow('path', img)
        cv2.waitKey(1)

        if j + 1 < img.shape[1]:
            if (img[i][j + 1] != white).any() and (img[i][j + 1] != black).all():
                if (img[i][j + 1] == to).all():
                    break
                img[i][j + 1] = used
                n = node((i, j + 1), current)
                q.append(n)

        if i + 1 < img.shape[0]:
            if (img[i + 1][j] != white).any() and (img[i + 1][j] != black).all():  # any value should be equal
                if (img[i + 1][j] == to).all():  # all the values should be equal
                    break
                img[i + 1][j] = used
                n = node((i + 1, j), current)
                q.append(n)

        if j - 1 > 0:
            if (img[i][j - 1] != white).any() and (img[i][j - 1] != black).all():
                if (img[i][j - 1] == to).all():
                    break
                img[i][j - 1] = used
                n = node((i, j - 1), current)
                q.append(n)

        if i - 1 > 0:
            if (img[i - 1][j] != white).any() and (img[i - 1][j] != black).all():
                if (img[i - 1][j] == to).all():
                    break
                img[i - 1][j] = used
                n = node((i - 1, j), current)
                q.append(n)
    show_path(current, start)
```

Neighbours of the current pixel is checked for existence and then checked for the destination pixel

```
114   flag = 0
115   for i in range(m):
116       for j in range(n):
117           if (img[i][j] == fro).all():
118               img[i][j]=[255,0,255] #start
119               start = node((i, j), None)
120               flag = 1
121               dfs(start)
122               break
123
124       if flag:
125           break
126       #else:
127           #print('no grey pixel found')
128
129   cv2.namedWindow('final', cv2.WINDOW_NORMAL)
130   cv2.imshow('final', img)
131   cv2.waitKey(0)
```

Loop is run on the image to find the starting point

BFS

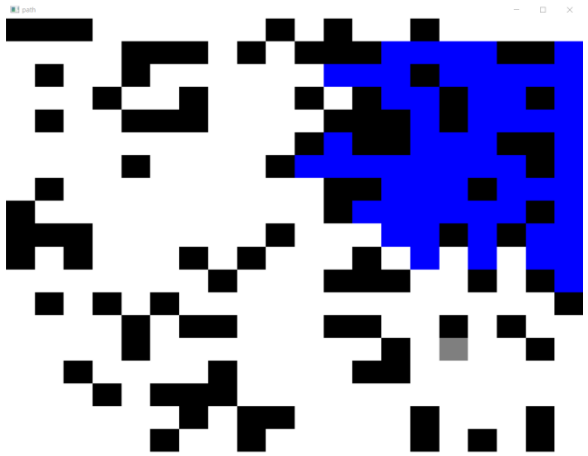
```
39  red = [0, 0, 255]
40  blue = [255, 0, 0]
41  green = [0, 255, 0]
42  white = [255, 255, 255]
43  grey = [127, 127, 127]
44  black=[0,0,0]
45
46  img1=img
47
48  class Node:
49      def __init__(self, index, parent):
50          self.x = index[0]
51          self.y = index[1]
52          self.parent = parent
53
54
55  def show_path(end, start):
56      print('e', end.x, end.y)
57      print('s', start.x, start.y)
58      current = end
59      print('here')
60      while current != start:
61          img1[current.x][current.y] = green
62          current = current.parent
63      cv2.namedWindow('final', cv2.WINDOW_NORMAL)
64      cv2.imshow('final', img1.astype(np.uint8))
65      cv2.waitKey(0)
```

```
68  def bfs(start):
69      q = deque()
70      q.append(start)
71      cv2.namedWindow('path', cv2.WINDOW_NORMAL)
72      while len(q):
73          current = q.popleft()
74          i, j = current.x, current.y
75          cv2.imshow('path', img.astype(np.uint8))
76          cv2.waitKey(1)
77
78          if j + 1 < img.shape[1]:
79              if (img[i][j + 1] != black).any():
80                  if (img[i][j + 1] == grey).all():
81                      break
82                  img[i][j + 1] = blue
83                  n = Node((i, j + 1), current)
84                  q.append(n)
85
86          if i + 1 < img.shape[0]:
87              if (img[i + 1][j] != black).any():  # any value should be equal
88                  if (img[i + 1][j] == grey).all():  # all the values should be equal
89                      break
90                  img[i + 1][j] = blue
91                  n = Node((i + 1, j), current)
92                  q.append(n)
93
94          if j - 1 > 0:
95              if (img[i][j - 1] != black).any():
96                  if (img[i][j - 1] == grey).all():
97                      break
98                  img[i][j - 1] = blue
99                  n = Node((i, j - 1), current)
100                 q.append(n)
101
102         if i - 1 > 0:
103             if (img[i - 1][j] != black).any():
104                 if (img[i - 1][j] == grey).all():
105                     break
106                 img[i - 1][j] = blue
107                 n = Node((i - 1, j), current)
108                 q.append(n)
109     show_path(current, start)
```

Here elements are popped from the front end of the queue

```
112  esc = False
113  for i in range(img.shape[0]):
114      for j in range(img.shape[1]):
115          if (img[i][j] == grey).all():
116              img[i,j]=red
117              start = Node((i, j), None)
118              esc = True
119              bfs(start)
120              break
121
122      if esc:
123          break
124
125  cv2.namedWindow('original',cv2.WINDOW_NORMAL)
126  cv2.imshow('original',img.astype(np.uint8))
127  cv2.waitKey(0)
128  print('Done!')
```

Path finding in BFS



```python
import cv2
import numpy as np
import time

img = cv2.imread("generated maze.png")
#img = cv2.resize(img, (200, 200))
img_copy = img.copy()

cv2.namedWindow("image",cv2.WINDOW_NORMAL)
cv2.imshow("image",img)


red = (0, 0, 255)
grey=(127,127,127)
green = (0, 255, 0)
blue = (255, 0, 0)
orange = (0, 128, 255)
pink = (255, 0, 255)

w, h, c = img.shape


class Node():
    def __init__(self, parent, position):
        self.parent = parent
        self.position = position
        self.g = np.inf
        self.h = np.inf
        self.f = np.inf


def get_min_dist_node(open_list):
    min_dist = np.inf
    min_node = None
    for node in open_list:
        if open_list[node].f < min_dist:
            min_dist = open_list[node].f
            min_node = open_list[node]
    return min_node
```

Dijkstra + Astar

```python
def get_dist(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return (((x1 - x2) ** 2 + (y1 - y2) ** 2)) ** 0.5


def obstacle(position):
    x, y = position
    if img[y][x][0] == 0 and img[y][x][1] == 0 and img[y][x][2] == 0:
        return True
    return False


def goal_reached(position):
    x, y = position
    if img[y][x][0] == 127 and img[y][x][1] == 127 and img[y][x][2] == 127:
        return True
    return False


def show_path(node):
    print('show path')
    current_node = node
    path = []
    while current_node is not None:
        path.append(current_node.position)
        current_node = current_node.parent
    path.reverse()
    for i in range(len(path) - 1):
        cv2.line(img_copy, path[i], path[i + 1], blue, 1)
    cv2.namedWindow('final path', cv2.WINDOW_NORMAL)
    cv2.imshow("final path", img_copy)
    cv2.imwrite("final_path.png", img_copy)
    if cv2.waitKey(1) == 'q':
        cv2.destroyAllWindows()
        return
```

```
80  def astar_algorithm(start, end):
81      print('astar called')
82      open_list = {}
83      closed_list = []
84      start_node = Node(None, start)
85      start_node.g = start_node.h = start_node.f = 0
86      open_list[start] = start_node
87      while len(open_list) > 0:
88          # print("dict size = ", len(open_list))
89          current_node = get_min_dist_node(open_list)
90          img[current_node.position[1]][current_node.position[0]] = orange
91          open_list.pop(current_node.position)
92
93          if current_node.position == end:
94              print("Goal Reached")
95              show_path(current_node)
96              return
97
98          for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]:
99              node_position = (current_node.position[0] + new_position[0], current_node.position[1] + new_position[1])
100             if node_position[0] > (w - 1) or node_position[0] < 0 or node_position[1] > (h - 1) or node_position[1] < 0:
101                 continue
102             if node_position in closed_list:
103                 continue
104             if obstacle(node_position):
105                 continue
106
107             img[node_position[1]][node_position[0]] = pink
108             new_node = Node(current_node, node_position)
```

```
110             new_node.g = current_node.g + get_dist(current_node.position, new_node.position)
111             if n==1:
112                 new_node.h=0
113             elif n==2:
114                 new_node.h = get_dist(new_node.position, end)
115             new_node.f = new_node.g + new_node.h
116 
117             if new_node.position in open_list:
118                 if new_node.g < open_list[new_node.position].g:
119                     open_list[new_node.position] = new_node
120             else:
121                 open_list[new_node.position] = new_node
122 
123         if current_node.position not in closed_list:
124             closed_list.append(current_node.position)
125 
126         cv2.namedWindow('path_finding', cv2.WINDOW_NORMAL)
127         cv2.imshow("path_finding", img)
128         cv2.waitKey(1)
```

- Choice is given to the user whether to use Dijkstra or Astar

```
if __name__ == '__main__':
    n=int(input("Enter 1 for Dijkstra\nEnter 2 for A*: "))
    for i in range(w):
        for j in range(h):
            if (img[i][j] == grey).all():
                start = (j, i)
                break

    for i in range(w-1,-1,-1):
        for j in range(h-1,-1,-1):
            if (img[i][j] == grey).all():
                end = (j, i)
                break

    print(start)
    print(end)

    #start = (126, 285)
    #end = (536, 368)
    begin_ = time.time()
    astar_algorithm(start, end)
    end = time.time()

    print("algorithm time = ", (end - begin_))

    cv2.namedWindow("path_finding", cv2.WINDOW_NORMAL)
    cv2.imshow("path_finding", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Dijkstra

Astar

Google maps to binary image conversion

```python
import cv2
import numpy as np

img=cv2.imread("LOCALITY.jpeg",0)
img1=cv2.imread("LOCALITY.jpeg")

p,q=img.shape

for i in range(p):
    for j in range(q):
        if img[i][j]<251:
            img1[i][j]=[0,0,0]
        else:
            #print(img1[i][j])
            img1[i][j]=[255, 255, 255]

img1[285][126]=[0,0,255]
img1[368][536]=[0,255,0]

cv2.namedWindow("img1",cv2.WINDOW_NORMAL)
#cv2.imshow("img1",img1.astype(np.uint8))
```

```
23  ksize = (5, 5)  # kernel size
24  # for smoothening
25  img1 = cv2.blur(img1, ksize)
26  #cv2.imshow('before thresholding', img1.astype(np.uint8))
27  for i in range(p):
28      for j in range(q):
29          if (img1[i, j] <= [127,127,127]).all():
30              img1[i, j] = [0,0,0]
31          else:
32              img1[i, j] = [255,255,255]
33
34  cv2.imshow('after thresholding', img1.astype(np.uint8))
35
36  cv2.imwrite("newloc.PNG",img1.astype(np.uint8))
37  cv2.waitKey(0)
```

Problem 2
Identification of Road Signs

Task 2.1

A video is made of images of traffic signals.

Templates are cut out of the images used in the video.

Templates are matched with the images in the video to find the signal.

The action that is required to be taken is stored in a text file.

Video Generation

```python
import cv2
import glob

frameSize = (500, 500)

out = cv2.VideoWriter('output_video.avi',cv2.VideoWriter_fourcc(*'DIVX'), 0.2, frameSize)

for filename in glob.glob('images1/*.png'):
    img = cv2.imread(filename)
    #cv2.imshow('image',img)
    img = cv2.resize(img, (500, 500))
    cv2.waitKey(0)

    out.write(img)
out.release()
```

Video is generated from images stored in a file.

Template Matching

```python
import cv2
import numpy as np
import glob

vid = cv2.VideoCapture('output_video.avi')
file = open('commands.txt', 'w')

while True:
    ret, img = vid.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    for filename in glob.glob('temp/*.png'):
        temp = cv2.imread(filename, 0)
        result = cv2.matchTemplate(gray, temp, cv2.TM_CCOEFF_NORMED)
        w, h = temp.shape[::-1]
        threshold = 0.9
        # Store the coordinates of matched area in a numpy array
        location = np.where(result >= threshold)
        for pt in zip(*location[::-1]):
            cv2.imshow('image', img)
            name = filename.replace('temp\\', '', 1)
            name = name.replace('.png', '', 1)  # name of the sign is same as the name of the template
            print('the sign board shows ', name)
            file.write(name)
            file.write('\n')
            break

    if cv2.waitKey(5000) & 0xFF == ord('d'):
        vid.release()

file.close()
cv2.destroyAllWindows()
```

Image is converted to grey scale.

When the template is found to match, the action corresponding to the template is stored in the text file.

- The Generate Video

```
the sign board shows  40
the sign board shows  270 right
the sign board shows  red light
the sign board shows  green light
the sign board shows  hump
the sign board shows  270 left
the sign board shows  50
```

Output