Gebe dich nie auf
Aerial Robotics Kharagpur - Perception Task 2

Priyanshu Kumar

***Abstract*— The task has several parts, and we must implement computer vision and image processing concepts.**
In the first part, we are given a distorted image with information about the pi digits. We are supposed to extract the corrupted digits and use them as a filter in the second part to obtain the original artwork from the modified one.
In the third part, we have to use this restored artwork as a template in a given collage of images to find the exact match. This will lead us to the fourth part, in which we must plan the path in a given maze from start to end.
The concepts used in implementing the text referred to above have a wide range of real-world applications in the context of aerial robotics and in general. Some of its applications in aerial robotics include drones identifying predefined objects, such as vehicles, people, or infrastructure, by matching templates with real-time images and detecting missing persons in vast terrains using predefined visual templates. Path planning also has various applications like autonomous drone navigation and easy navigation of warehouse drones.

I. INTRODUCTION AND PROBLEM STATEMENT

"The digits of pi that were distorted can be each multiplied by 10*pi and converted respectively to the greatest integer less than or equal to them. The resulting numbers can be arranged in descending row-major order and used as a 2x2 filter.". The distorted pi image is attached below. We have to extract which digits of the pi_image are corrupted by comparing them with the actual digits of pi. Once we have found the actual values of the corrupted pixels, we have to multiply the value by 10*pi and arrange the numbers in row-major order and get the filter. 4 numbers [a,b,c,d] arranged in a 2x2 row-major matrix will be of the form $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$.

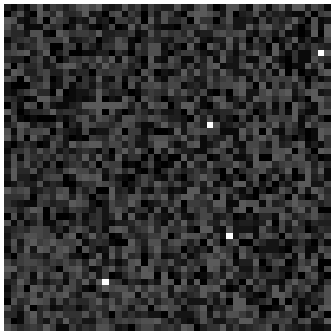


Fig. 1: pi_image.png

"..... perform 3 operations: bitwise OR, bitwise AND, bitwise XOR. He has a habit of distorting pictures by applying a filter on pre-existing pictures ..... apply the filter on the distorted image to recover the famous portrait.". Now, we are provided with the artwork modified by Picasso. We have to obtain the original artwork by applying the filter obtained on the modified one with only 3 possible operations, that is, OR, XOR and AND.

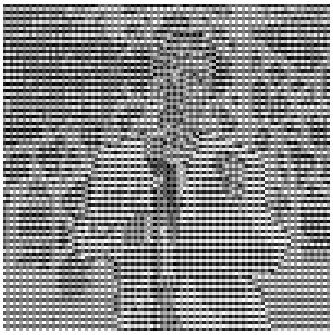


Fig. 2: artwork_picasso.png

"Given below is an image(collage.png) which contains the template found above ..... Add the abscissa and ordinate, multiply it with pi and round it to the greatest integer less than or equal to it to get the password to the zip file." The original artwork obtained will be a template for searching it in the collage.png attached below. We have to find the template in the collage, find the x and y coordinates of the upper left corner of the location where the template is found, multiply it with pi and take the floor of the number to obtain our password.

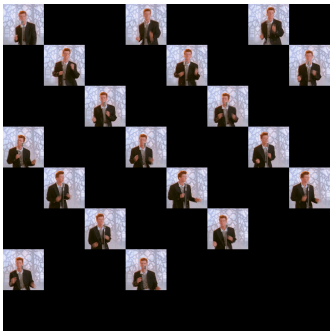


Fig. 3: collage.png

"In the images provided, perform the RRT-connect algorithm from start to end point and attach the image of the final path formed. ". The password obtained will unlock the zip folder, which contains a picture of a map onto which we have to apply the RRT path planning algorithm to find the path between the start and endpoints.

II. Related Work

Other approaches, apart from what I use, might be other algorithms in path planning like RRT*and A*. These algorithms provided better paths than RRT but at the cost of speed and computational power.

III. Initial Attempts

1. While finding the template in the collage, initially, I slid 1 pixel after each check, but as it was evident that no info would be returned from the black area, I started increasing it in steps of 300.

2. While implementing RRT, first I thought of sampling random points only within a neighbourhood of each point of some fixed circle and continuing this, but this turned out to be inefficient as this will limit the rate of growth of trees at extremes and hence our tree will not be able to reach the goal in a fast manner.

IV. Final Approach

After importing the libraries, I defined a function which gives the details of the image which will be used throughout the task.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pygame
import time

def image_details(url):
    img = cv2.imread(url, cv2.IMREAD_GRAYSCALE)
    print(img)
    print(f"Image Size : {img.size}")
    print(f"Image Shape : {img.shape}")
    plt.imshow(img, cmap="gray")
    plt.show()
    return img
```

SUBTASK 1: FINDING THE FILTER

In pi_image.png, four pixels are distorted. To find these distorted pixels, I first found out the size of the image, which was 50x50 pixels. So I asked ChatGPT to provide me a text file containing the first 2500 digits of pi, and then I compared it with the pixel values. The pixel values were like 30, 10, 40, 10, ... So, to match them with the pi digits, I had to divide it by 10. Then, matching them pixel by pixel provided the distorted digits with which I obtained the filter by arranging them in row-major order. The filter obtained is $\begin{bmatrix} 0 & 251 \\ 94 & 282 \end{bmatrix}$.

```

img = image_details("pi_image.png")

pixel_values = (img//10)
print(pixel_values)

with open("pi_50x50.txt", "r") as file:
    pi_digits = [int(num) for line in file for num
        in line.strip().split(',')]
pi_digits = np.array(pi_digits)
pi_digits = pi_digits.reshape(50,50)

dist_digits = []
for a,A in zip(pixel_values, pi_digits):
    for b,B in zip(a,A):
        if b!=B:
            print(b,B)
            dist_digits.append(B)

filter = np.floor(np.multiply(dist_digits, 10*np.pi
    ))
filter = filter.astype('int')
filter = filter.reshape(2,2)
```

SUBTASK 2: RESTORING THE ORIGINAL ART

In artwork_png, we start at the top left corner, place the filter and apply our desired operation, then we shift the filter by two units, again apply the operation, and move on. We keep doing this across all the rows of the image. And we store our results in a new array. We have 3 options for the operations; the result of all three operations are attached below.

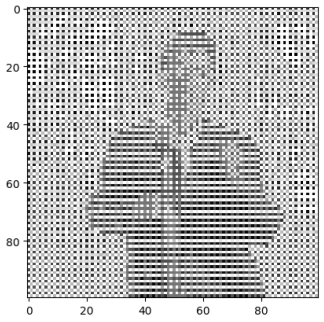


Fig. 4: OR operator

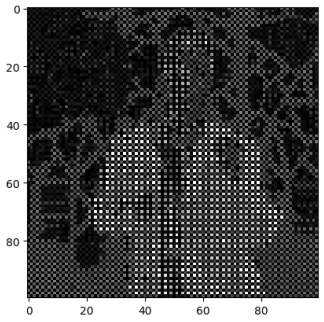


Fig. 5: AND operator

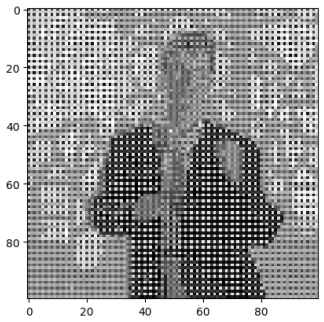


Fig. 6: XOR operator

It is very obvious to observe that the artwork is being restored by using the XOR operator, the code for which the same is attached below.

```
artwork = image_details("artwork_picasso.png")

pic = artwork.copy()
height,width = pic.shape
for i in range(0,height-1,2):
    for j in range(0,width-1,2):
        pic[i,j] ^= filter[0,0]
        pic[i,j+1] ^= filter[0,1]
        pic[i+1,j] ^= filter[1,0]
        pic[i+1,j+1] ^= filter[1,1]
plt.imshow(pic, cmap="gray")
plt.show()
```

SUBTASK 3: TEMPLATE MATCHING

```
template = pic
collage = image_details("collage.png")
```

We define a function which returns the error when we compare two images of same dimesnions.

```
def template_matching(image, template):
    error = (image - template)**2
    error = np.sum(error)
    return error
```

Now we perform template matching on collage.png using our template.

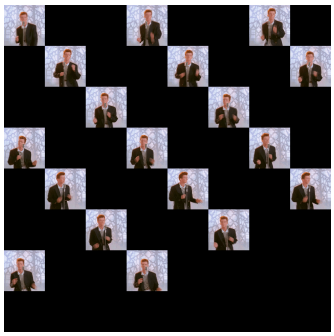


Fig. 7: collage.png

```
h,w = collage.shape
min_error = np.inf
for i in range(0,h-199,100):
    for j in range(i%300,w-99,300):
        sample = collage[i:i+100,j:j+100]
        error = template_matching(sample,template)
        print(error)
        if(error<min_error):
            min_error = error
            x,y = i,j
print(x,y)
```

Now we have the coordinates which is (100,100). (100+100)*pi = 628.3185307179587 So, the password is 628.

SUBTASK 4: PATH PLANNING

```
maze_img = image_details("maze.png")
```

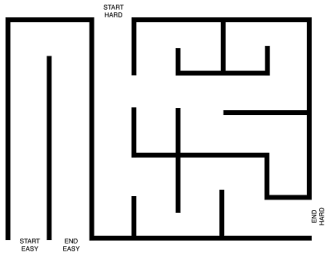


Fig. 8: maze.png

The implementation of RRT from scratch is done in the python notebook.

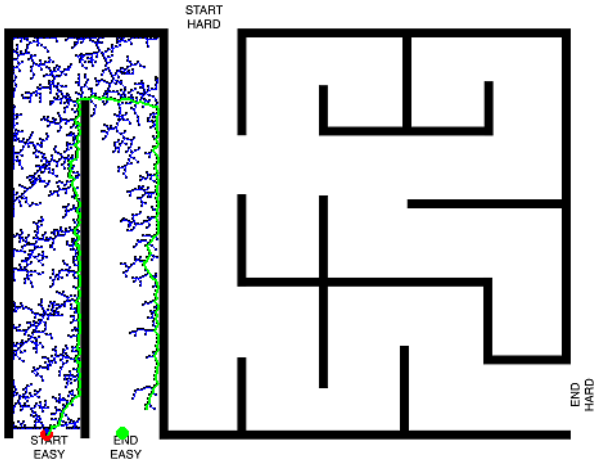
RRT stands for Rapidly Exploring Random Tree. In this algorithm, a tree-like structure is formed. At any iteration, we randomly select a point in the screen, and then we find the node of the tree which is closest to that random point. Then we move towards that random point from the nearest point in steps of whatever our stepsize is (of course, only if there is no wall ahead) and establish a new node there. This nearest node will be the parent of the new node. We keep on performing this until we reach our goal.

In my implementation, we first have to select the start and goal positions on the pygame window by clicking at the desired locations; then, only the algorithm will start.

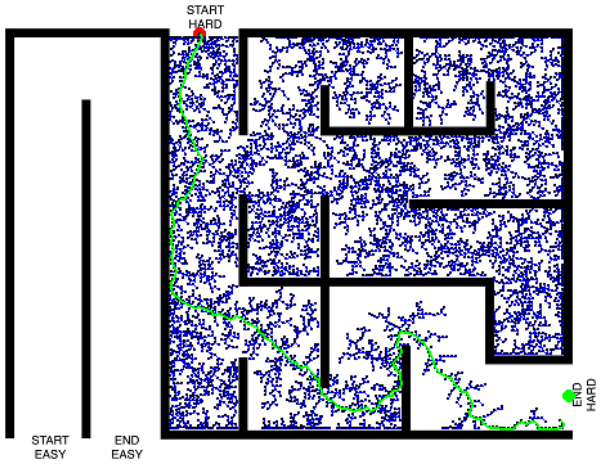
In my implementation, I have devised a tree-like structure with each node having some parent and some child, except for the first and last node. Whenever a point is randomly selected, we then find the nearest node, which is stored in our list of nodes, by evaluating the Euclidian distance between every possible pair. Then from that point, we move towards the new point in stepsize. Once reached the goal, for backtracking, we just have to follow the parent of each node, which is a fairly easy task as every node has only one parent.

The algorithm results for various stepsize and time taken is attached on the next page.

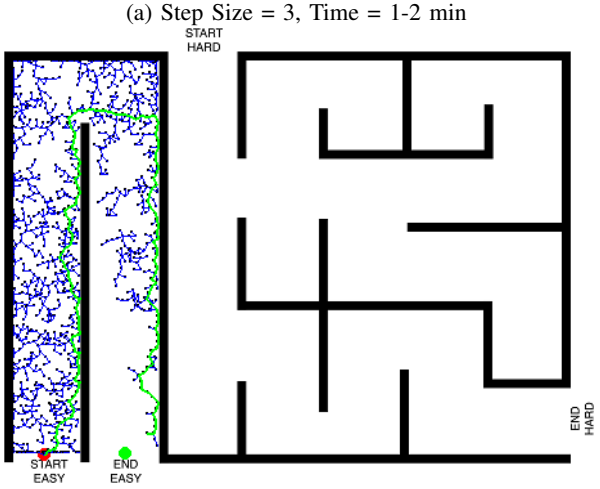
For choosing optimal stepsize, we have 2 factors to consider, that is, time taken and smoothness of the path. As we increase stepsize, the time taken to reach the goal reduces. But at the same time, the path becomes more zigzaggy, So step size of 5 seems to be an optimal tradeoff between the two.



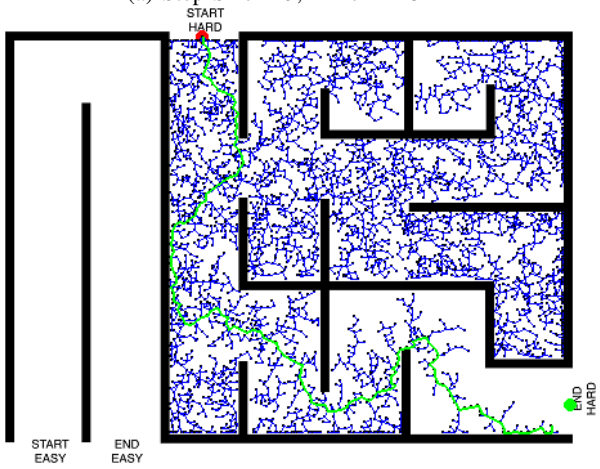
(a) Step Size = 3, Time = 1-2 min



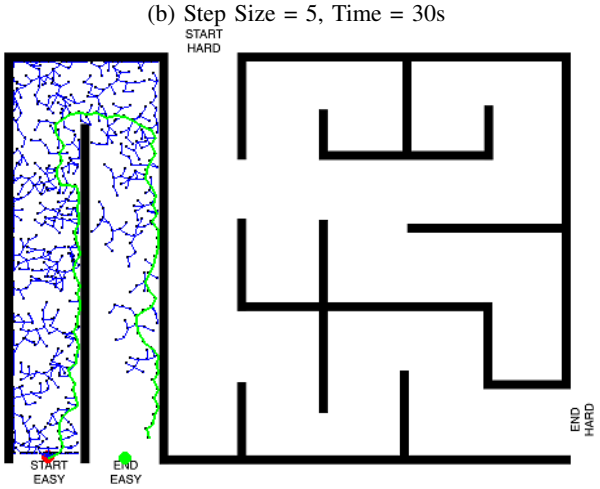
(a) Step Size = 3, Time = 4-5 min



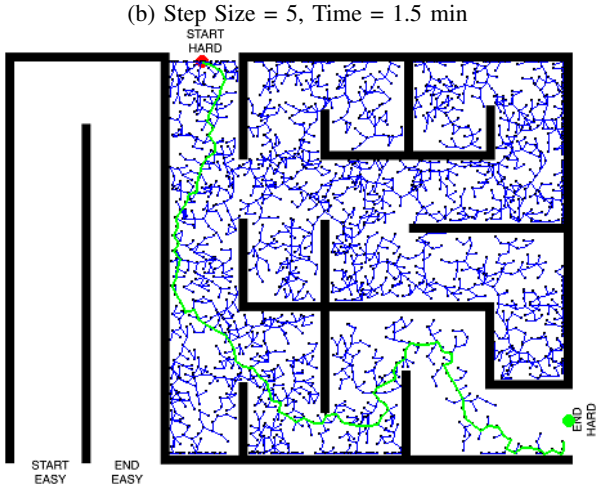
(b) Step Size = 5, Time = 30s



(b) Step Size = 5, Time = 1.5 min



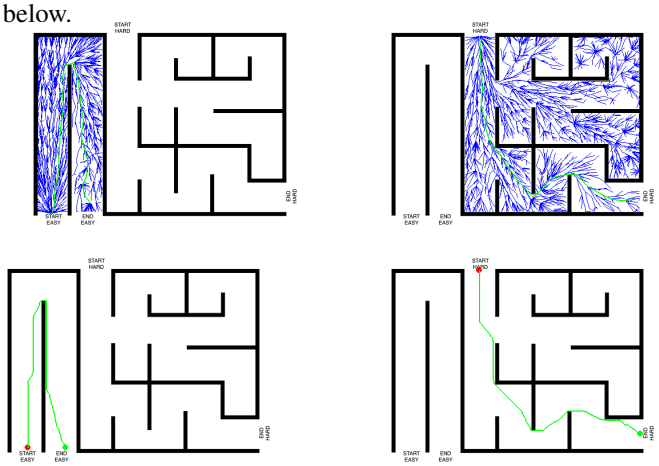
(c) Step Size = 7, Time = 10s



(c) Step Size = 7, Time = 30-60 s

V. Results and Observation

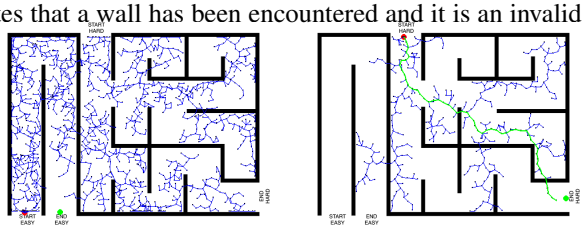
I have compared my results with various other algorithms like RRT*, and A*.
The results and comparision of the algorithms are provided below.



RRT* algorithm accounts for rewiring of nodes even after the goal is reached, thus finding an optimal path but at the cost of computational power, and also RRT* is slower than RRT.
Similarly, the A* algorithm also searches for an optimal path using heuristics but is slower than RRT. Large memory is required for A* It tends to perform poorly in higher dimensions.

VI. Future Work

There are a number of problems in my algorithm.
1. Step Size: As of now, I am limited to taking a step size of up to 7 only; if I increase the step size, a tendency to jump walls starts to arise, which we do not want. To tackle this, we can apply an algorithm which checks for all the pixels between the initial and final node, maybe in a staircase manner if it is diagonal, and if a black pixel is found, it indicates that a wall has been encountered and it is an invalid



move.

2. Random Space Too Large: If we consider the easy maze and sample random points from the complete canvas, it leads to inefficiency. If we somehow devise a method to sample points in the required region, it will be more efficient and the path will also not creep along the walls as we saw in paths of easy maze.

CONCLUSION

This project involved solving a series of computational challenges related to image processing, filtering, and path planning. This project demonstrated the use case of mathematical transformations, bitwise operations, custom image processing, and path planning algorithms. The combination of filter-based reconstruction, template matching, and RRT-Connect pathfinding showcased the effectiveness of these techniques in handling distorted images and navigating complex environments.

References

[1] **First Principles of Computer Vision** , https://www.youtube.com/@firstprinciplesofcomputerv3258
[2] **Wikepedia**, https://en.wikipedia.org/wiki/Kernel_(image_processing)
[3] **Numpy Documentation**, https://numpy.org/doc/
[4] **PyGame Documentation**, https://www.pygame.org/docs/
[5] **YouTube, Aaron Becker**, https://www.youtube.com/watch?v=Ob3BIJkQJEwpp=ygUEcnJ0IA
[6] **YouTube, Tong Zhao**, https://shorturl.at/q6UC7
[7] **ChatGPT**, https://chatgpt.com/