Gebe dich nie auf

Jyotishko Banerjee

Abstract—

Ever fantasised yourself to be Sherlock Holmes, solving mystery cases? This project gives an oppurtunity to be a detective and make your dreams come true. Remember detectives observing some objects, finding minor clues and guessing about the flashbacks, or noticing some old painting and guessing its history and easily identifying if its fake? Well, luckily, to do all that, we are empowered with Computer Vision to guide us through the same journey.

First, we deal with a painting that has been tampered with, out of jealousy, to find out the distorted pixels and make a note of them to unleash a 2*2 filter. We take the filter and travel to anothe part of the world where we see Pablo Picasso distorting an image with the same filter by performing mathematical operations and claiming it as his own portrait. Can we find out the real portrait and expose Picasso? We better, because we have got more clues to find.

After we find out the real image that was distorted by Pablo Picasso and expose him, we shall have to use the image as a template to get hints to a password. The template is to be applied on a collage of various positions of Rick Ashtley and find out the top-left corner of the picture matched, get the co-ordinates, operate a function on them and get the password.

This password is the key that holds a maze within the zip file, that awaits to be solved by RRT algorithm applied by us and finally solving the mystery.

Stories apart, this project involves heavy utilization of numpy and OpenCV modules and teaches us their in depth concepts and uses. I personally enjoyed going through the overall process. Here comes the explainations oen by one.

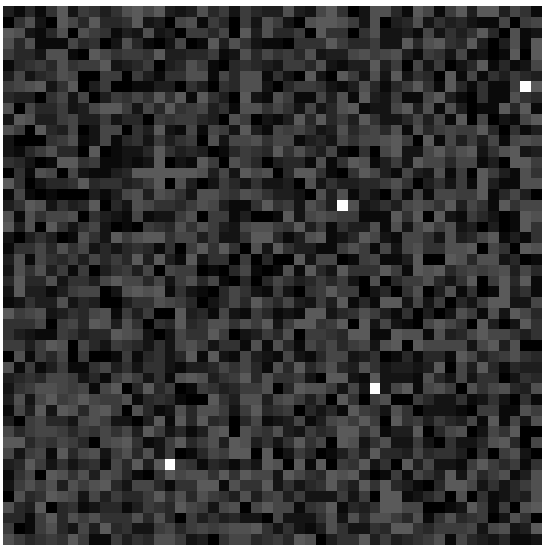
I. INTRODUCTION

Yelta Kcir, a π lover had a painting of π in his room, which was corrupted by another π lover Grant Sanderson. Then, we were explained how a 2×2 filter can be arranged by finding out distorted values of pixels in the corrupted π image. Initially, a lot of difficulty was faced to understand how to get the kernel because the statement about how to get them was very unclear. But with more reading and clarification from ARK members, it was finally clarified, which took about a week of reading and understanding. After the kernel was found, it had to be used on a distorted portrait that was claimed to be made by Picasso, to get the real portrait, which in turn had to be used as template on a collage, and then finding the co-ordinate of the top-left of the matched image and manipulating them to find password to unlock a maze on which RRT was to be applied.

Meanwhile I tried a few hit and trail methods to find out the template and the password to the zip file about which I have explained in the 'Initial Attempts' section. Overall, with enough trials by analogy and trackbar arrangement, it is possible to find the password and by reverse analogy, even the actual image and operations, which are highly unconventional methods. But finally, it was solvable with the conventional method that

II. PROBLEM STATEMENT

Skipping all the story part, the derived the problem goes as follows.



The above picture is a portrait that is related to π. Close inspection reveals that each pixel of the above image in a continuous manner contains the values of digits of π multiplied by 10. Like (0,0) has 30, (0,1) has 10, (0,2) has 40 and so on where the value of π is 3.14.... Now, four of the pixel values are deviated from the value of what it should have been. A 2×2 filter is to be constructed using the digits of π that should have been in the distorted pixels in descending order in the filter.

The filter obtained has been used to distort a portrait, which is to be reused to obtain the real portrait from the distorted portrait given below. The way in which it was distorted was as follows:

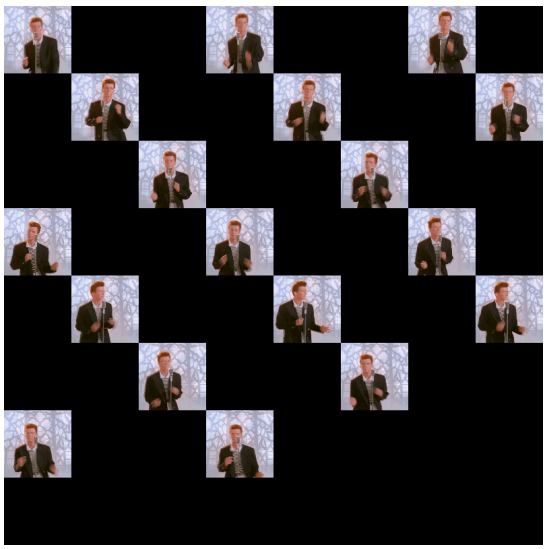
• The artist Picasso knows only 3 operations - bitwise_OR, bitWise_AND and bitwise_XOR which derives the fact and the operations performed between the kernel and picture is among the above 3

• The process is to be explained. First, the filter was kept in top-left corner and operated on the correspondingly matched pixels, then it was shifted rightward by 2 units and again the operation was made, in this way, by subsequent rightward and downward shift, teh whole portrait was covered and distorted, resulting in the following:

Our task is to take the filter (found before) and the distorted image shown above, to find the actual image that was distorted.

Please note the exact operations made on exact kernel parts are unknown to us and has been given for the problem solver to determine

After the real image is obtained, it is to be resized into 100*100. Then, it should be used as a template to apply template matching on the below collage from scratch.



The co-ordinate of the top-left corner of the matched part is to be noted, which is to be used to find the password to a locked zipped file. The password is integer less than or equal to $\pi \times$ (x co-ord + y co-ord).
After unlocking the zip file, a maze is obtained where the RRT algorithm is to be applied to make trees and find and finally highlight the path from start to end area.

III. Related Work

Attending the Winter School on Computer Vision, has helped me to a great extend to solve the problem easily once I figured out the logic and methods. During and after Winter School, I made a lot of small projects like co-ordinate tracker that would identify the colour and co-ordinate of a pixel on which the pointer was which I could operate with WASD, path finding project on applying Dijkstra and A* algorithm on a map, masking of a shirt, identification of road margins using Hough Transformation, building Template Matching from scratch, etc

IV. INITIAL ATTEMPTS

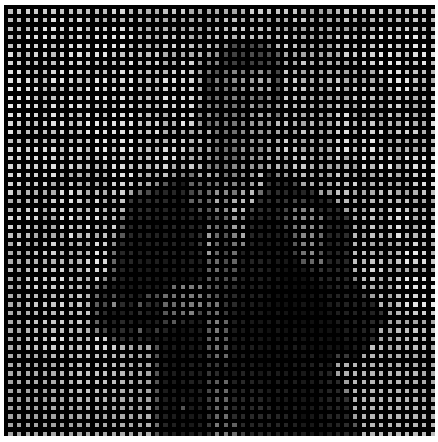
Initially, not being able to understand what to do with the π image, I applied an unconventional method of working backwards. Here it goes.

I analysed the distorted portrait and tried to guess which picture from collage could fit it. Careful analysis will limit my trials to 5-6 trials. With the 'Co-ordinate Tracker' function explained before in the RELATED WORK, I found out the pattern in the co-ordinates of the pics in the collage. Then, in a python file, with nested for loop, I generated the possible passwords. The co-ordinates were (0,0), (0,200), (0,400),.... I took them and applied the operation [(x+y)*π] in each iteration. After a few trials of putting passwords, I got the answer as 628. I got the maze, after which RRT was applied, explained more in FINAL APPROACH. Now, using numpy techniques, I extracted the image at (100,100). Now, it was about guessing the operations and values, the 2 big challenges.

I took the distorted image, and displayed only the (0,0), (0,2), (0,4),...,(2,0),(2,2)... i.e. every 1st pixel of the distorted image, others were blackened out.

Then tried with (0,1),(0,3)...

Same with other 2



Next, for 4 of them individually, I started applying AND, OR with various values and tried to equalize them with the real portrait, but none of them worked. And then something struck me.

Now my thought process went as follows, I realised that if AND or OR was applied by Picasso to distort the portrait, it was not possible for us to extract the real portrait because in AND or OR, if one initial and the result was known, we cannot predict the other initial with full accuracy. Like If A AND 0 is 0, A may be 1 or 0. So I was doubtful. Then, after a few internet surfs I realised 'With AND or OR we can the correct initial with 75% accuracy' like A OR B is C and C and B are known, then A can be predicted with 75% accuracy. And 'if XOR of A and B is C, then XOR of B and C is A or XOR of A and C is B'. I analysed the statements and understood them properly.

I then tried XOR with distorted portrait and from 0 to 255 for each of the 4 images and equalized them with the real portrait. In this way, I found that the values as 251, 219, 94, 0 which are $10*\pi*8$, $10*\pi*7$, $10*\pi*3$, $10*\pi*0$ respectively.

V. Final Approach

Here goes the conventional method.

Analysing the π image by co-ordinate tracker, I find out that the pixels values are digits of π multiplied by 10 and four of them are 255 i.e. distorted from original value as given the trend, maximum value of a pixel can be 90 (as maximum digit can be 9) and definitely has to be a multiple of 10. So, I use a python program to find the value of the actual pixel values that should have been in their place. I put the digits of π in a string and iterated it through each pixel of the pi-image, and wherever it did not match the pixel value, the actual value was printed. I arranged these 4 numbers in descending order, as a filter – 9,8,3,0.

Now, here is the catchpoint, from the previous experiments and INITIAL ATTEMPTS, given that the operation is bitwise_XOR, I have reasons to believe that the filter is actually 8,7,3,0 instead of 9,8,3,0, as getting the template by 8,7,3,0 actually produces a match while template matching. I applied bitwise_XOR on the distorted image and the filter 8,7,3,0. The actual image of Rick Ashley was got (XOR's special property was explained before in the INITIAL ATTEMPTS section).

The portrait was to be used as a template on grayscaled collage by using template matching from scratch. Now, in usual manner, using for nested loop takes a lot of time to actually finally get the result. And in this case it had overall 4 nested for loops which I had used when I had made template matching from scratch in Winter School. But like a month ago I had learnt that numpy functions are great way to make things execute faster. So, it took some time but I was able to replace the inner 2 loops with numpy functions and the time of execution was reduced to 83 secs. Finally, after whole algorithm finishes, the matched coordinate was printed as (100,100) and the password 628 by doing [(100+100)*π].

```
Scanning: 88 % completed
Scanning: 89 % completed
Scanning: 90 % completed
Scanning: 91 % completed
Scanning: 92 % completed
Scanning: 93 % completed
Scanning: 94 % completed
Scanning: 95 % completed
Scanning: 96 % completed
co-ord matched: [100, 100]
password: 628
```

The Template Matching Algorithm was used as follows:

• We take the template, in this case 100×100

• Then we start from top left corner and apply the 'Operation' one by one

• Like first the filter works on (0,0) to (99,99) square, then it shifts one pixel and acts on (0,1) to (99,100) shifting rightward by one pixel.

• I this way it covers the whole collage shifting pixel by pixel performing the operation on all of them and storing the value of the "Operation" at the top-left corner at each iteration

• There can be various operations, but in our program we have used the inner concept of TM_SQDIFF which means square of difference.

• Suppose, the template is in a particular part of the collage. Then the 2 values of the collage and template of the same corresponding positions will be subtracted and then squared. This will be done to all the 100*100 pixels and then will be added and stored to the top left most corner of the collage at the current iteration where the template is currently.

• In each iteration, we compare each summation with the minimum summation got till now, if the summation is lesser than minimum, it is assigned the minimum and the co-ordinate is noted and further iterations are continued

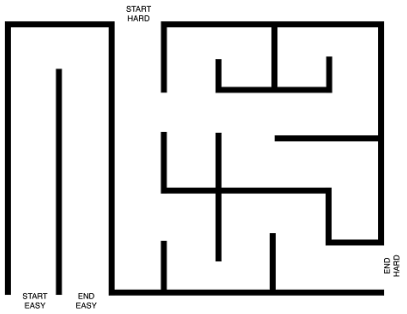
• In each iteration, we check if the summation is zero, then the co-ordinate of the the top-left position is assigned to min_coord

• Finally, after the whole collage is scanned, the co-ordinates having the minimum value till now is returned

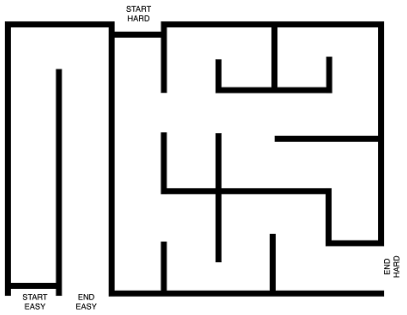
• We could have also compared each summation with the minimum summation got till now, if the summation is lesser than minimum, it would have been assigned the minimum and the co-ordinate would have been noted and further iterations were to be continued continued

• But, we chose the previous way, because, we need the exact template and not the minimum, which might not match fully. Also, we need to take care of the case where we might put in wrong template and match is not found. When this happens function returns zero, is identified in the main script and and "No match found" is printed

The following maze was got after unlocking the zipped file.



Before applying RRT, a few modifications were made. In the START positions, black boundaries were added by me which otherwise would have let the algorithm go not within the maze but from outwards.



Then, the usual RRT was applied. RRT Algorithm –

• Set start and end node and the iteration limit

• Suppose, we reach a step where there are n nodes in the tree

• A random point in C space will be generated (literally any) called Xr

• The closest node from Xr called Xc will be linked to Xr.

• From beforehand, we have set a max_dist such that if the distance between Xr and Xc is below max_dist, Xr will be set as new node Xn.

• Else, if the distance exceeds the max_dist, Xn will be set on the line joining Xc and Xr at max_dist from Xc

• Once, Xn is set, it becomes part of the tree and its parent is Xc

• In this way, each generated node is linked to its parent, and the parent in turn, is linked to their parent. In this manner, all nodes can backtrack to Start

• Then, in the next iteration choose another completely random point and continue the same as step 3 to 6.

• For first step, the only available node is Start, so it will treat Start as Xc and allocate Xn accordingly

• If the Xn is sufficiently close to the End, then the algorithm returns the sequence of nodes it went through to reach there and the path henceforth is remembered

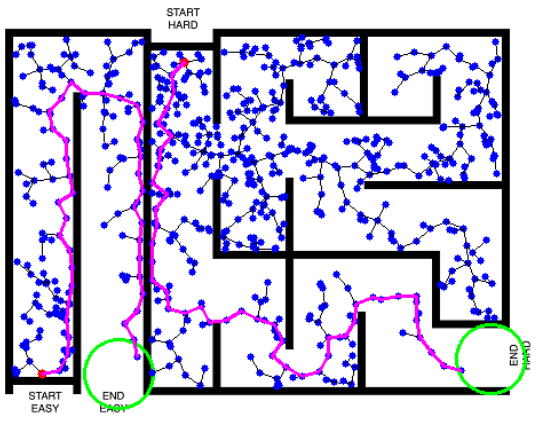
Then consider cases with obstacles

• Suppose after the Xn is decided, if the path between Xc and Xn have an obstacle, then the Xn will not be allocated and the algorithm will move to another Xr in the step iteration

• If there is an obstacle itself in the place of Xr, it will not be considered and another random point will be chosen on the next iteration

RRT Algorithm's utilisation in the program -

In a loop, the iteration limit was set. In each iteration, Xr spawned randomly inside the image and looks for the closest available node and sets it as Xc i.e. its parent node. Now using point_on_line_at_dist function, it finds Xn, the new node at maximum or lower distance from Xc. The above function also checks for obstacles, if obstacle is found, it returns 0 instead of co-ordinate of Xn and in the program, if Xn is 0 then the loop continues without setting Xn. The function display_path shows real time path finding. The algorithm stops when iteration limit is reached or a node gets in the END area.
Now, each node is stored in a list, whereas each of its parent is stored in another list. In the function display_final_path at last, the final path is highlighted.



Finally, the pictures presented here are taken using cv2.imwrite function.

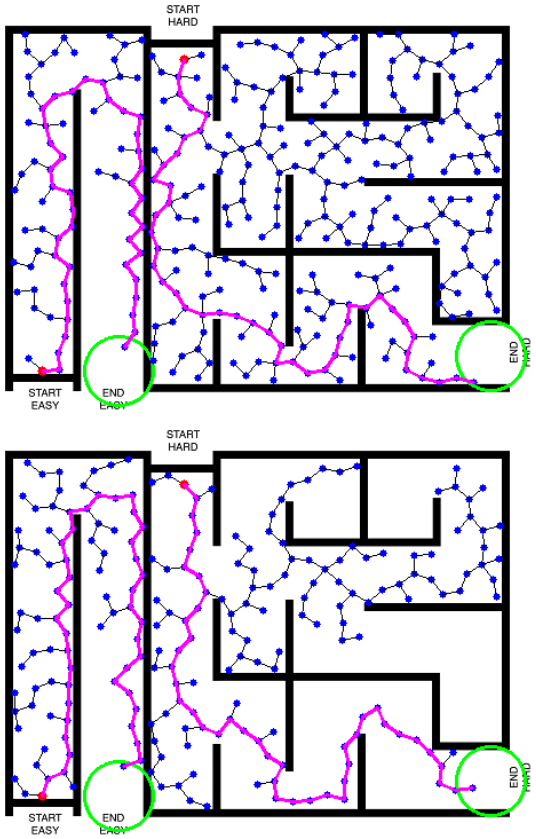
VI. Results and Observation

As the RRT is based on Sampling Method, hence each time the program runs, a lot of things will vary, the number of nodes established, the final path obtained, the final time taken to find the path.

As seen above, each case, a lot of things are different. A major problem with this algorithm is, the more nodes get added and as the algorithm progresses, the program gets loaded with more data and the execution speed slows down.

Now, I have found a weird modification, that if applied in this maze, decreases the time complexity by a lot. It basically involves skipping the iteration where Xr is Xn meaning, Xr spawns in less than maximum distance allocated by us to a node, this prevents unnecessary creation of nodes, making program less loaded and also increasing

reach of the tree per iteration.



However, the above approach of skipping mostly works with limited amount of obstacle and a lot of free space. If the path has lots of obstacles then the conventional approach is favourable.

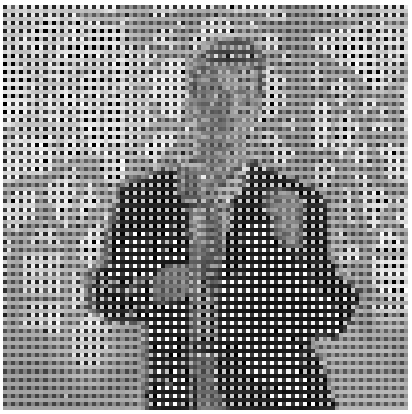
If we talk about Dijkstra's or A* algorithm, they would have taken up a lot of time to reach the goal. They work better in very small C spaces, but in real life, a robot usually gets a huge C-space, now judging each and every pixel on the way will take up a huge amount of time which is very unfavourable. Hence, Sampling Method based algorithms like RRT work better in real life path-finding.

VII. Future Work

Let me explain why I think that the filter obtained i.e. 9,8,3,0 is not the correct one. I will be using a bit of **Game Theory**.

Let us consider the ARK members have framed the question by keeping in mind that we will actually be able to derive the real image by applying the filer on the distorted image. If they would have applied bitwise_and or bitwise_or, then we will never be able to get the correct image due to reasons explained in INITIAL ATTEMPTS. So that leaves us with only bitwise_XOR that actually makes possible to get the correct image using the filter and distorted image due the special property of XOR explained in INITIAL ATTEMPTS.

Now, if we apply the filter 9,8,3,0 on the distorted image, the resultant image first of all doesn't make a sense as given below.



Secondly it doesn't find a proper match during Template Matching, which reconfirms the fact that 9,8,3,0 is not the correct filter.

```
Scanning: 88 % completed
Scanning: 89 % completed
Scanning: 90 % completed
Scanning: 91 % completed
Scanning: 92 % completed
Scanning: 93 % completed
Scanning: 94 % completed
Scanning: 95 % completed
Scanning: 96 % completed
No match found
```

According to me, to get the real filter, 9 is to be replaced by 7, which means the distorted pixel shouldn't have been [42,15] which had value 90 but anything else like [42,10], [42,39] or [43,2] where pixel value was 70. Hence, the correct filter is 8,7,3,0

Now, coming to path finding algorithms, RRT doesn't give us the shortest path to the goal. To overcome this, we need to apply RRT* which does 2 work - rewiring the new node if necessary and rewires the previously established tree to ensure nodes have the shortest path from the start generating extremely straight paths. But this also increases the execution time.

The main problem faced in general is the execution speed of Python. Although, Python is preferred in AL/ML for its huge availability of libraries and easy syntaxes and codes, but speed still remains an issue.

To overcome this, especially in OpenCV, the library functions need to be used properly to get the most efficient code. Like instead of using nested for loops, numpy operations should be used.

Talking of overall scenario, a programming language with easy syntax like Python but with speed like C++ would have been a much better choice if is made in the future.

CONCLUSION

The problem involved churning our brain to think like a detective and wide applicability of OpenCV functions and Numpy arrays. The problem taught us a few Image Processing methods in depth and their application.

Image Processing is an important part of AL/ML through which we teach the machine to analyse images and videos and get data through them. Further level of Image Processing includes application of AI/ML models like CNN.

In ARK, Image Processing is very much necessary for drones to give image feed for analysis, for object identification, obstacle identification and safe navigation through Drone Localization combined with other sensors. It is lot like 'eyes of the drone'.

In general, RRT is one of the most widely used path-planning algorithms in real life which is used by robots worldwide for path-planning.

REFERENCES

[1] S.M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning", TR 98-11, Computer Science Dept., Iowa State University, October 1998

[2] Technology Robotics Society, IIT KGP, "Winter School" on "Computer Vision" Slides and Lectures, 2023

[3] Tim Chinenov, "Robotic Path Planning: RRT and RRT* ", Medium, 2019

[4] Google, "OpenCV Documentation"

[5] Rick Ashtley, "Never Gonna Give You Up", Rick Ashtley, YouTube, 2010