Gebe Dich Nie Auf

U Skanda Aithal *

Abstract— **In this task I mainly used the process of template matching and implemented RRT(Rapidly exploring Randim Tree) Algorithm on the maze image provided.**

In order to complete this task I have first recovered the digits of pi that were distorted. Then using the transformations given in the task on the recovered digits I have obtained the kernel that can be used to extract the template from the given distorted image. I have performed all possible combinations of the given operations between the image and the kernel and have chosen one of the best possible images that I could extract as the template and performed template matching using a program that I have written which finds the square of the difference between the image and the template pixel wise and adds them. Using this I obtained the password for the zip file and got access to the maze image. I implemented the RRT Algorithm on the maze obtained as instructed. In the implementation of RRT algorithm I have created two classes, Maze and RRT. The Maze class is just to read the maze and initialize the starting and ending points. The RRT class has the methods required for the implementation. The methods in the class are sample - to sample new points, inObstacle - to check if that point is blocked by any obstacle, dist - to find the distance between two points and inRange - to find if the sample point is closeby the endpoint. The process of template matching can be used in very small applications of object detection. The RRT Algorithm can be used in automated vehicles in path planning.

I. INTRODUCTION

The task was to obtain the template for performing the template matching from a distorted image. The kernel needed to do this was also indirectly provided. After performing the template matching the coordinates of the top left corner of the region where the template is matched will give us the password to a zip file when transformed according to the given instructions. The zip file contained a maze on which we were supposed to implement RRT Algorithm. Initially I did not understand the problem statement so I just considered one of the three mentioned operations (bitwise and, bitwise or and bitwise exclusive or) were used between the kernel and the image to distort the template. But the results were not satisfactory. Then I understood my mistake and performed all combinations of the operations mentioned above and chose one of the most satisfactory images that I could extract as the template. The program I used for matching template finds the square of the difference between the kernel and the image pixel wise and adds them and assigns it to the index of the top left corner. For the implementation of the RRT Algorithm I used two classes, Maze and RRT. Maze is just used for reading the maze and initializing the starting and ending points. The RRT class has the methods required for the implementation.

*

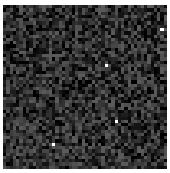

Fig. 1. Distorted pi image

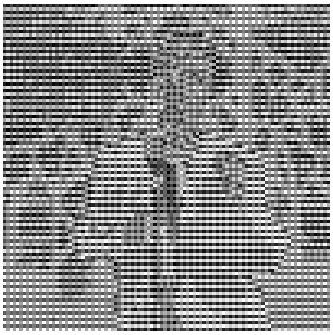

Fig. 2. Distorted template

II. PROBLEM STATEMENT

Explanation

The first task was to recover the kernel used to distort an image. The kernel was a (2x2) matrix. An image which had the digits of pi in the tenth place of pixel values was provided. But it was distorted, some of the pixel values were missing. The digits that were distorted had to be found and then multiplied by 10*pi and converted to the greatest integer less than or equal to them. These numbers when arranged in decreasing row major order will give us the kernel. The operations used to distort the image were given (bitwise and, bitwise or and bitwise exclusive or). After the kernel is obtained all we had to do was find the combination of these operations which when applied between the kernel and the image, the template will be obtained. The template thus found should be used to perform template matching on another image given and find the coordinate (x,y) of the top left corner of the matched template in the image. The template matching should be done without using inbuilt functions of openCV library. The ordinate and abscissa of the coordinate when added, multiplied with pi and converted to the greatest integer less than or equal to it will give the password to a zip file containing the image of a maze. Rapidly exploring Random Tree(RRT) Algorithm was to be implemented on the maze between given starting and ending points.

III. Related Work

For the process of template matching one can use 1. Cross correlation 2. Normalised cross correlation 3. Sum of absolute difference

IV. INITIAL ATTEMPTS

RRT Algorithm - To check if the newly sampled point is blocked by obstacle : initially thought of checking if the obstacle is present in the whole rectangular region formed by the two points as the ends of a diagonal. But it was a really bad approximation.

V. Final Approach

Implementation

The first task was to recover the distorted digits in the image with the pixel values as digits of pi. The distorted pixels were white, so I iterated over the pixels of the distorted pi image and found the coordinates of the pixels which were white. If the coordinate of a pixel is (x,y) then (x*50+y) will give its position considering the numbering of position starts after the decimal point (1 is the first digit in 3.14). After finding the positions of all the missing digits I used a website to obtain the digits in those positions. I multiplied those digits with 10*pi and converted them to the greatest integer less than or equal to them as instructed and arranged them in decreasing row major order to obtain the 2x2 kernel. The three operations that were told to have been used to distort the template were bitwise and, bitwise or and bitwise exclusive or. I performed all 81 combinations of these 3 operations between the kernel and the distorted template and chose one of the best templates that I could extract as the template. The template is then used to perform template matching. For template matching I calculated the square of the difference between the template and the image pixel wise and added them and assigned them to the coordinates of the top left corner where the template matching is performed on the image. Since in the image the pictures were in certain region, the top left coordinates of the pictures were always an integral multiple of 100, I did not iterate over the whole image. I performed the operation only in those places where pictures could be found. To store the result, instead of having a matrix of the size of the image I used a matrix of dimension (64x3) where the first two elements of each row stored the coordinates and the third row stored the value. The image on which the template matching was to be performed was of the shape (800x800). There were 64 places with the top left corners with the coordinates that are integral multiples of 100. So the result matrix had 64 rows. Then I found the coordinates which had the least value by using a simple loop. I added the ordinate and abscissa of the coordinate, multiplied them with pi and converted it to the greatest integer less than or equal to it as instructed. I got the password (628) for the zip file which had the image of the maze on which we were supposed to implement RRT.

For the implementation of RRT Algorithm, I have implemented two classes. The Maze class is defined, which initializes the maze image, start position, and end position. The maze image is loaded using cv2.imread() function.The RRT class is defined, representing the RRT algorithm. It takes the maze, start position, and end position as input parameters. It initializes other variables such as ext (the number of division in which a line is divided to check if its points lie on an obstacle) and points (list of tree nodes). The sample() method generates a random point within the specified range of the start and end positions. The inObstacle() method checks if a point lies within an obstacle in the maze. It divides the line segment between the current point and a tree node into smaller segments (based on ext) and checks if any of the pixels in those segments correspond to the color of obstacles (black in the maze image). The dist() method calculates the Euclidean distance between a point and a tree node. The inRange() method checks if the current point is within a certain distance of the end position, indicating that the goal has been reached. The findParent() function is defined outside the class to find the parent node of a given node in the RRT. Then an instance of the Maze class and the RRT class is created, passing the maze image, start position, and end position as arguments. The main loop begins, which continues until the goal is reached or the program is terminated. The sample() method is called to generate a random point. The loop iterates over the existing tree nodes to check if the new point is in an obstacle or not. If not, it updates the minimum distance and index of the nearest node. If a feasible point is found, it is added to the tree as a new node, with the nearest node as its parent. A line is drawn between the new point and its parent in the maze image using cv2.line(). If the new point is within range of the end position, the goal is reached. The end point and the path from the last tree node to the goal are visualized in the maze image. The maze image is displayed using cv2.imshow() and the program waits for a key press using cv2.waitKey().

After the main loop, the backtracking process begins to visualize the path from the goal back to the start position. The maze image is updated accordingly, and the process is displayed using cv2.imshow() and cv2.waitKey().

In summary, the code uses the RRT algorithm to find a path from the start to the end position in a maze. It generates random points, checks for obstacles, and constructs a tree-like structure to explore the maze. The path is visualized using OpenCV functions for drawing lines and circles on the maze image.

VI. Results and Observation

I have performed all possible combinations of the three operations between the kernel and the distorted image. The best possible template I could extract is given. Using this template I have recovered the hidden password to the zip file containing the maze image. I have performed RRT Algorithm on this as instructed. An image of the maze as the algorithm is being implemented as an image after it is implemented is given.

Fig. 3. Template Extracted

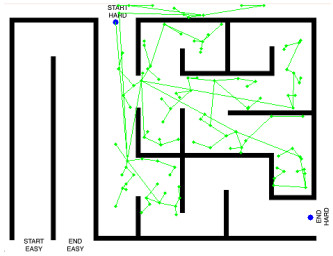

Fig. 4. During implementation

VII. Future Work

The RRT algorithm is not guaranteed to find the shortest path. The algorithm also explores some region unnecessarily, even though some of the points generated in the tree are much closer to the end point. To tackle this we can use a heuristic function like we use in A* algorithm to bias the generated points to be in the regions closer to the end point. This helps in generating optimal paths. This is implemented in one of the variations of RRT, RRT*-informed. There are many other variations of this algorithm like RRT*, RRT-Connect, RRT*-Smart and others that provide more optimal paths.

CONCLUSION

The problem statement was to implement RRT Algorithm on a maze which was to be accessed from a zip file which was protected with a password that was hidden. The major task was the implementation of the RRT Algorithm. In the implementation, one of the major problem was to tackle obstacle detection. To handle this I came up with the method of dividing the line into several junctions and checking if these junctions lie on the obstacle. This worked well enough to implement the path in the given maze.

Path planning algorithms in general is widely used in autonomous robotics to navigate in environments safely and efficiently. These algorithms help robots to carry the tasks of surveillance, mapping and exploration.

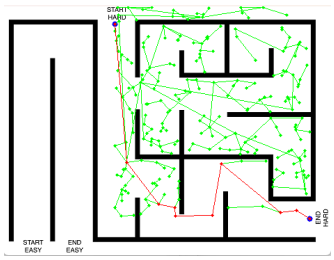

Fig. 5. After implementation

References

[1] Paridhi Swaroop, O., "An Overview of Various Template Matching Methodologies in Image Processing", International Journal of Computer Applications, Vol. 153, No. 10, (0975-8887), 2016.
[2] https://youtu.be/OXikozpLFGo