Localization in known environment Documentation

U Skanda Aithal *

Abstract— **This documentation introduces a problem of localizing a drone within a maze using maze images, local snapshots, and control actions. The goal is to accurately determine the drone's position, starting from an unknown initial position.**

Localization is crucial in autonomous robotics as it enables robots to determine their position in a known or unknown environment. By utilizing sensor inputs, such as cameras or GPS, robots can accurately navigate and perform tasks, making localization essential for autonomous operation and efficient interaction with the surrounding world. The approach used in this code snippet is simple template matching between the maze and snapshots. The drone is moved and additional snapshots are taken to confirm the location if multiple matches are found. The code leverages OpenCV functions for template matching.

I. INTRODUCTION

The problem involves localizing a drone within a maze using a combination of maze images, local snapshots, and control actions. The drone's initial position within the maze is unknown, and the goal is to accurately determine its position. The provided files include player.py, utils.py, and MapGeneration.py.

The Player object represents the drone and offers methods such as getMap(), getSnapShot(), and movehorizontal() and movevertical().The task is to complete the strategy() function in player.py to implement a localization strategy for the drone. The goal is to get the drone's position with minimum error. The program should print the final position of the drone at the end of the strategy() function's execution.

I have used simple template matching between the maze and the snapshots to achieve this task. I have also moved the drone and have taken screenshots after moving to confirm the location if in case multiple matches were found.

II. PROBLEM STATEMENT

The problem at hand involves the localization of a drone within a maze using a combination of available information.

The drone has no knowledge of its initial position within the maze, and the objective is to accurately determine its position. To achieve this, the problem provides three files: player.py, utils.py, and MapGeneration.py.

The Player object represents the drone and offers several methods to interact with the drone's environment. The getMap() method retrieves an image of the entire maze, allowing the drone to understand the overall structure of its surroundings. The getSnapShot() method simulates the drone's camera, providing a 51x51 image of the immediate environment surrounding the drone. The movehorizontal() and movevertical() methods enable the drone to move horizontally and vertically within the maze.

The main task is to complete the strategy() function within player.py. This function implements the localization strategy for the drone. The objective is to devise a program that minimizes the error and uncertainty in determining the drone's position. The strategy should utilize the available information, including the maze image, local snapshots, and control actions, to accurately estimate the drone's position within the maze.

Upon executing the strategy() function, the program should have a high degree of certainty regarding the drone's position. The localization algorithm should strive to minimize error and provide an accurate final position for the drone. The expected output of the program is the printed final position of the drone within the maze.

In summary, the problem requires implementing a localization strategy for a drone within a maze. By leveraging the provided maze image, local snapshots, and control actions, the aim is to accurately determine the drone's position. The solution entails completing the strategy() function in player.py, resulting in a minimized error and the successful localization of the drone within the maze.

III. RELATED WORK

The method used here is template matching since I had the image of the maze and local snapshot. But in real world applications, localization is tackled using methods which involve probabilistic approaches. SLAM (Simultaneous Localization and Mapping) is also used solve the localization problem. In dynamic environment, distance filter can be employed to detect the changes in environment and the process of localization can be seperated from mapping..

IV. INITIAL ATTEMPTS

Initial approach was to just use template matching once using the snapshot of the given location. Even though it worked there might be a case where multiple matches can be found. So then I used multiple snapshots by changing drone position to confirm the drone location.

V. FINAL APPROACH

The provided code snippet involves the localization of a drone within a maze using computer vision techniques.

Map = player.getMap(): The getMap() method retrieves an image of the entire maze and assigns it to the Map variable.

step = 20: The step variable represents the step size or distance that the drone will move within the maze.

*

place1, place2, place3: These empty lists will store the coordinates of potential locations where the drone might be present.

area = player.getSnapShot(): The getSnapShot() method captures a snapshot of the environment surrounding the drone.

res = cv2.matchTemplate(Map, area, cv2.TM CCOEFF NORMED): The matchTemplate() function compares the Map image with the area snapshot using the correlation coefficient as the matching method. The result is stored in the res variable.

loc = np.where(res == 1.0): The where() function of NumPy finds the coordinates where the matching result is equal to 1.0 (indicating a perfect match). These coordinates are stored in the loc variable.

hor = player.move horizontal(step): The move horizontal() method is called with the step value as an argument to move the drone horizontally within the maze. The resulting horizontal movement is stored in the hor variable.

area1 = player.getSnapShot(): Another snapshot of the environment is captured and assigned to the area1 variable.

res1 = cv2.matchTemplate(Map, area1, cv2.TM CCOEFF NORMED): Similar to the step where res is defined, the correlation coefficient is calculated between Map and area1, and the result is stored in res1.

loc1 = np.where(res1 == 1.0): The coordinates where the matching result is 1.0 are stored in the loc1 variable.

ver = player.move vertical(step): The move vertical() method is called with the step value as an argument to move the drone vertically within the maze. The resulting vertical movement is stored in the ver variable.

area2 = player.getSnapShot(): Another snapshot of the environment is captured and assigned to the area2 variable.

res2 = cv2.matchTemplate(Map, area2, cv2.TM CCOEFF NORMED): The correlation coefficient is calculated between Map and area2, and the result is stored in res2.

loc2 = np.where(res2 == 1.0): The coordinates where the matching result is 1.0 are stored in the loc2 variable.

The following for loops iterate over the coordinates obtained from loc, loc1, and loc2, and adjust the coordinates to the center of the detected areas by adding 25 pixels to both the x and y coordinates. These adjusted coordinates are then appended to the respective place lists.

The final loop checks for potential drone locations by comparing the adjusted coordinates from place1, place2, and place3. If a combination of coordinates satisfies the condition, the location is printed as the drone's location.

Finally, the OpenCV windows are closed using cv2.destroyAllWindows().

Overall, the code performs drone localization within a maze by comparing the captured snapshots with the maze image. It calculates the correlation coefficient and finds the potential drone locations based on matched regions. The drone's movement is simulated by adjusting the coordinates and checking for consistency between the different snapshots. The code aims to determine the drone's location within the maze accurately.

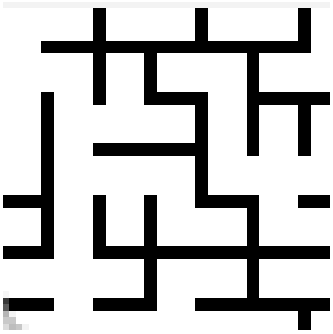

Fig. 1. Snapshot of the local environment

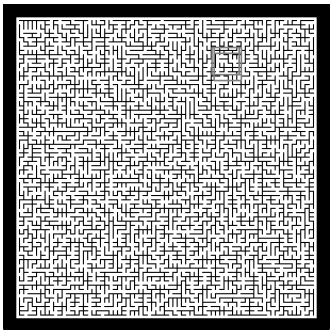

Fig. 2. The image of the maze after template matching in three locations

VI. Results and Observation

The results of this program is were accurate even when I took just one snapshot. In the final approach I used three snapshots to confirm the drone location in case of multiple matches, so they are accurate. I have assumed that the drone is at the centre of the snapshot of the local snapshot provided. The result for the template matching and image of the snapshot is given.

VII. Future Work

The program is pretty simple with just one method being used. We can use more snapshots if the maze is very repetitive in patterns.

CONCLUSION

This task was to just localize the drone in the maze given by using the help of the snapshots and controlling the drone movements. This was accomplished by just using template matching. Multiple snapshots were used and the consistency between different snapshots is checked to determine the position accurately. This was a really simple introduction to the problem of localization. Localization is used in autonomous robotics to determine the position of the robot in a known or unknown environment using sensor inputs. The environment can also be dynamic in real life.

References

[1] https://www.sciencedirect.com/science/article/pii/S1319157821000550

[2] https://info.vercator.com/blog/what-is-slam

[3] Dali Sun, Florian Geißer, Bernhard Nebel, "Towards Effective Localization in Dynamic Environments", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) , 2016