Localisation in known environment

Jyotishko Banerjee

***Abstract*—**

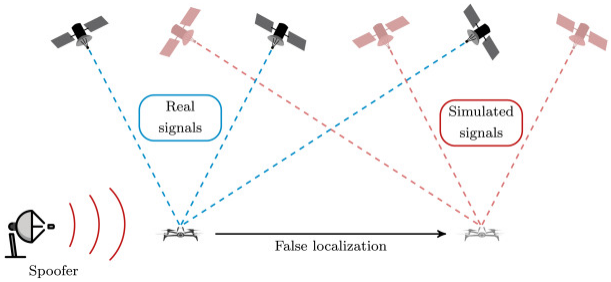
Suppose you have to travel somewhere new on your own and don't even have any idea how to do that. What do you do? You pull out your phone and open Google Maps, put the location, switch on the GPS and boom! The path to travel is in front of you. But suppose you are travelling towards an off town area, or maybe a village, then the Google Maps you trust becomes one the most untrusted sources to navigate, many-a-times because the area has not been researched and updated in the Maps, but majorly because the internet connection is poor and the Google Maps start glitching, shows your current location wrong or even show wrong routes.

What if there was a way in your hone itself could determine the location or suppose sensors on your phone sent some data to the main server and rather than tracking from radio signals to determine your location, they use the sensor data to predict our location. Suppose the area you are travelling in is previously known by the server, now you click photos of the area and send it to main server, the main server matches the data with the pre-existing data and predicts your location. This doen't involve radio waves getting disrupted or getting bend to misjudge your position.

The drone positions are predicted by satellites by sensing from which postions radio waves are coming, which are prone to inaccuracies. This happens a lot of times with drones where location of drone is misjudged causing a lot of trouble. Hence, in this paper, we look at a way to overcome them by alternate methods of localisation.

I. INTRODUCTION

Conventional UAVs (Unmanned Aerial Vehicle) use GNSS (Global Navigation Satellite Systems) such as GPS for localization. Localization means getting to know the exact location of an UAV in a known environment. However, GNSS systems are usually prone to inaccuracies due to use of radio signals to determine distance between satellites and UAVs. Radio signals, due to various obstacles and layers in the atmosphere can go through a lot of changes and intake a lot of noise. Also, if the drone is moving with enough speed, lags are created while transmitting the radio signals. Hence, it becomes difficult to localise the drone and leads to wrong navigation that might lead to destruction of UAV parts by coming across obstacles, getting lost in the environment or losing wireless connection with the controller.



This problem has lead to development of unconventional localization methods that avoids using radio signals. More about this have been given in section RELATED WORK.

II. PROBLEM STATEMENT

To solve a problem properly, it is first necessary to understand the problem properly. In this question, understanding the the 2 initial .py files - MapGeneration.py and utils.py is necessary.

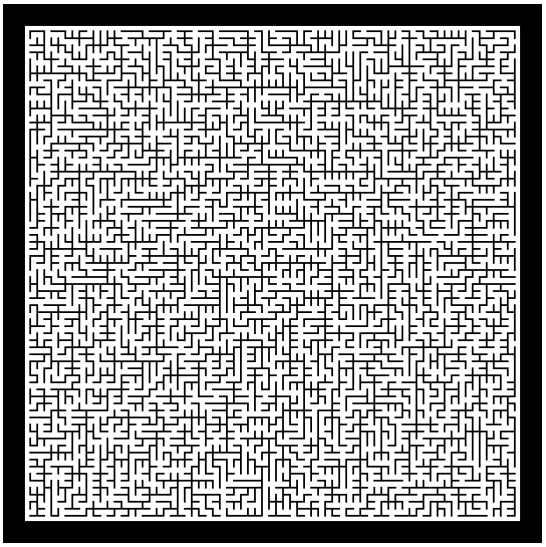
The MapGeneration.py generates random mazes using library mazelib (which had to be installed from GitHub), also it generates a random position for the drone in the maze.

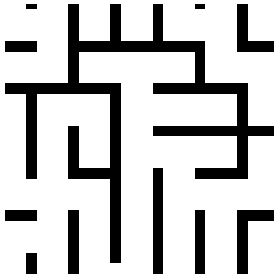
The utils.py defines the Player class and the 4 functions under it that can be done by the Player instances:

- getMap(): Returns image of the entire maze
- getSnapShot(): Returns image of drone's surrounding area of dimensions 51×51
- move_horizontal(): Movement horizontally less than or equal to number of steps given
- move_vertical(): Movement vertically less than or equal to number of steps given

The random position and the map are assigned as attributes of the Player instance while initiation, being imported from MapGeneration.py

In the player.py under strategy function, we are expected to localize the drone i.e. find the location of the drone using maze image, local snapshots and control actions. The final determined location should be as accurate as possible and should finally be printed.





III. Related Work

Localisation refers to process of estimating the position of an object relative to a given reference based on sensor inputs. For drone localization, visual localisation involves camera images to estimate the drone's position in a given environment. In real life, inputs are taken from various sensors in the drone and the taken data is analysed to predict the position of the drone in the known environment. This alternate method escapes using radio signals to send the position to satellite to determine its position, but enables the drone itself to predict its position. In this problem, we see a virtual way to use Visual Localisation to determine drone location in a random maze.

utils.py has used OOPs(Object-Oriented Programming) concepts involves classes, objects, attributes, etc. to define the Player class whose instance was later declared in player.py being imported from utils.py. To understand the functions of Player and how it was designed, understanding concepts of OOPs was necessary which I was learning while I was trying to solve Question number 2 i.e. making a Vending Machine using Finite State Machine.

IV. INITIAL ATTEMPTS

When I gave a reading to the problem statement a first few times, I was confused regarding what to apply. I thought, we got the whole map, screenshot of surrounding area, isn't it indicating usage of a simple Template Matching. But was it going to be this simple? Were we allowed to use OpenCV to determine the location, or just complex mathematical equations and matrices?

Nevertheless, I tried thinking of some complex ways that would actually not use OpenCV modules, but something more practical and with clever and proper uses of all 4 functions. I gave enough thoughts, but couldn't come up with more practical methods. So, I asked a doubt to ARK members if we are allowed to use OpenCV modules, they confirmed. So, I went with the usual template matching method added with some complications to ensure accuracy which I have presented in FINAL APPROACH.

V. Final Approach

Initailly, required imports are made from utils, then cv2, numpy and keyboard. The main script in player.py calls the function strategy. I have completed the strategy function as follows:

I got the map and the snapshot and applied template matching with high accuracy of 1 and stored the results in loc. This high accuracy ensures that I only pic the right spot accurately.

But what if there are more than 1 place where the template matches, or if due to some glitch, or slight inaccuracies, the matching doesn't occur at accuracy of 1? I have defined 3 if-else cases to handle the situation.

```
res = cv2.matchTemplate(map, snapshot, cv2.TM SQDIFF)
loc = np.where(res ==1)
```

First condition is with accuracy of 1, if the only one location is matched, then the location is printed.

If no location is matched, then in a while loop is run until at least a match is found or accuracy becomes less than 0. In each iteration, the accuracy is decreased by 0.01. Next case if 2 matches are found in previous case then we apply a strategy of moving the drone randomly and then again doing some template matching. Basically we import random and for each vertical and horizontal movements randomly select number of steps within -10 to 10. And feed it to move horizontal and move vertical functions and store the returned actual number of steps moved to 2 variables. Then, we again take a screenshot on current position and template match with the map with 0.95 accuracy.

Now, mathematically speaking, given the random nature of the mazes generated using mazelib, 2 places of the same maze having exactly same snapshot, even exactly same snapshot of the neighbourhood is tends to 0.

Now, after applying template matching of the random neighbourhood of more than one matched area, we can find the exact location of the drone among multiple possible areas which were determined previously.

Finally, the drone is brought back to original position using the control movements feeding in the opposite of the actual steps the drone had moved. At the end, the final position of the drone is printed with the number of locations matched. Template Matching Function works as follows: • We take a filter, in this case 51×51

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

• Like first the filter works on (0,0) to (50,50) square, then it shifts one pixel and acts on (0,1) to (50,51) shifting rightward by one pixel.

• I this way it covers the whole map 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 TM_SQDIFF means square of difference.

• Suppose, the template is in a particular part of the map. Then the 2 values of the map and template the same corresponding positions will be subtracted and then squared. This will be done to all the 51*51 pixels and then will be added and stored to the top left most corner of the map at the current iteration where the template is currently.

• Accuracy values will be assigned to each of them, like the pixel where the sum is 0 will have 1 accuracy, and as the value will keep increasing, the accuracy will keep decreasing.

• We can access the points later by specifying accuracy with np.where() function

There are other methods of Template Matching as well where accuracy criteria may be different.

Now, it is to be remembered when the position through template matching is determined, it gives the position of top left corner of the area. But, our drone is in the middle, So before printing, we need to add half of the dimension of the snapshot to determined location to get the exact drone location.

I have also defined a function display() which takes in the map, snapshot and the location determined, using cv2.circle and cv2.rectangle functions, I have displayed the map, the snapshot, highlighted the snapshot area with green and put a red dot on the map on the location determined along with added feature of them blinking using 'for' loop for easy identification.
Also, I have coloured the central position of the snapshot red.

Function movement() was defined that allows the user to move the drone using the Player functions using WASD. movemment():

• loc is taken as paraameter to get the drone position. The loc is converted to list from tuple because as the position of drone changes, changes must be made in loc to update the current position

• I print a statement to let the user know that the drone now can be controlled using WASD

• in a while loop, I use keyboard.is_pressed() functions and then player.move_vertical() and player.move_horizontal() under them to move the drone as per command given by the keyboard

• Then we we player.getMap() and player.getSnapShot() and assign them to map and snapshot variables respectively.

• These are binary images, so I convert them to np.uint8 format and then to BGR, all the values other than 0 (i.e. 1 as it was a binary image) are converted to 255

• Then we use cv2.circle and cv2.rectangle to locate the snapshot area and drone on the map, also we color the location of the drone red in the SnapShot.

• To break the loop, user just needs to type "q"

All of the above happens in each iteration, which means the real-time movements will be visible in both the map and the snapshot

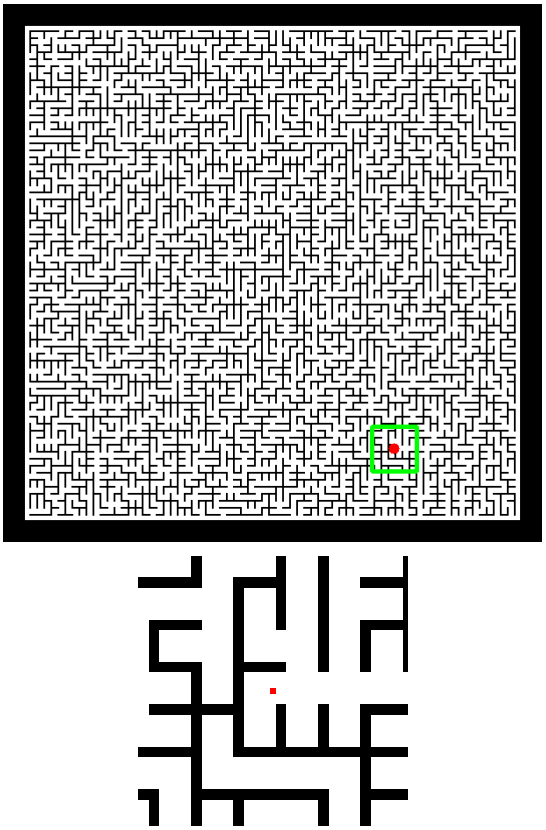
VI. Results and Observation

After checking a lot of results, it has been found that:
• The generation of mazes is so random that none of the cases have been observed where 2 portions have been found the same, i.e. the snapshot has matched with 2 areas at the same time

• However, in a few cases, no matching areas were found for cases with accuracy 1. There, slightly decreasing the accuracy (to 0.99) has successfully matched a part

Finally, each time the program is run, the exact initial position of drone is printed.

Along with that, the following images are displayed to give visual idea of drone location



Notice how the the snapshot area and the position has been marked in the map and the central position in snapshot, i.e. drone's position has been marked red.

Next, after some time of blinking, the marking becomes stable and we can control the drone's positions using WASD from keyboard and the real-time movements will be visible in both snapshot and map_identified.

Another important fact to be noticed is the speciality of cv2.namedWindow('snapshot', cv2.WINDOW_NORMAL) Declaring a window of size NORMAL actually lets the user freely change size of the window without causing unnecessary INTERPOLATION and damaging the pixels. Changing

the dimensions by clicking and holding will actually enlarge the pixels.
So if you find difficulty in seeing the snapshot because of its small size, do feel free to enlarge it on your own.

VII. Future Work

The unconventional method of drone localisation with drone sensors and not using GNSS is relatively new concept and has arrived due to complicacies and problems that it had possessed while working with drones in inaccessible areas.

Although, this project was more about virtual way of visual localisation by just using template matching, real life localisations are more complex and need a lot of work to be done to make them more improved and actually use them efficiently in the practical scenario. A lot more research and experimentations are required in this area to make it more perfect and commercially usable.

CONCLUSION

This project was about Drone Localisation to use drone sensors to estimate positions instead of radio signals being sent to satellites i.e. avoiding GNSS systems as they are prone to inaccuracies. Higher versions of the project hold future prospects to be used by drones.
For us, it was about exploring a few OpenCV concepts in depth with utilisation of probabilities to determine strategies to predict the drone position with high accuracy.

References

[1] Google, "OpenCV Documentation"
[2] Technology Robotics Society, IIT KGP, "Winter School" on "Computer Vision" Slides and Lectures, 2023
[3] Seung Yeob Nam and Gyanendra Prasad Joshi, "Unmanned aerial localisation using distributed sensors", Sage Journals 2017