Localization in Known Environment

Pranaya Chowdary

Abstract— **This documentation addresses the problem of localizing a drone within a maze using visual snapshots and control actions. The drone's initial position relative to the maze is unknown, and the objective is to estimate its position accurately using camera images and movement controls. The implementation involves template matching techniques to match the drone's current snapshot with different positions within the maze. The strategy function in the provided code is responsible for localizing the drone by analyzing snapshots and comparing them with the maze image.**

The program aims to minimize error and print the estimated position of the drone. The paper discusses the problem statement, the approach to localization, and presents the implementation details of the strategy function. Results are visualized by drawing a rectangle around the estimated drone position on the maze image.

I. INTRODUCTION

Unmanned Aerial Vehicles (UAVs) often rely on localization to navigate from one location to another accurately. In this research, we address the problem of localizing a drone within a maze using visual snapshots and control actions. The goal is to estimate the drone's position accurately relative to the maze, leveraging camera images and movement controls.

The provided code consists of a Player object representing the drone within the maze. The Player object offers methods to retrieve the maze image, capture snapshots of the environment, and control the drone's movements. The task is to complete the strategy() function within the code to localize the drone by analyzing snapshots and comparing them with the maze image.

II. PROBLEM STATEMENT

The problem is to localize a drone within a maze using visual snapshots and control actions. The drone's initial position within the maze is unknown, and the task is to estimate its position accurately based on the provided resources.

The provided code includes a Player object representing the drone and offering methods to retrieve the maze image, capture snapshots, and control the drone's movements. The strategy() function within the code needs to be completed to implement the localization logic.

The Player object offers the following methods:

- getMap(): Returns an image of the entire maze.
- getSnapShot(): Simulates the action of the drone's camera and provides a 51x51 image of the environment surrounding the drone.
- move horizontal(): Controls the movement of the drone along the row.
- move vertical(): Controls the movement of the drone along the column.

The objective is to minimize error and print the estimated position of the drone at the end of the execution of the strategy() function.

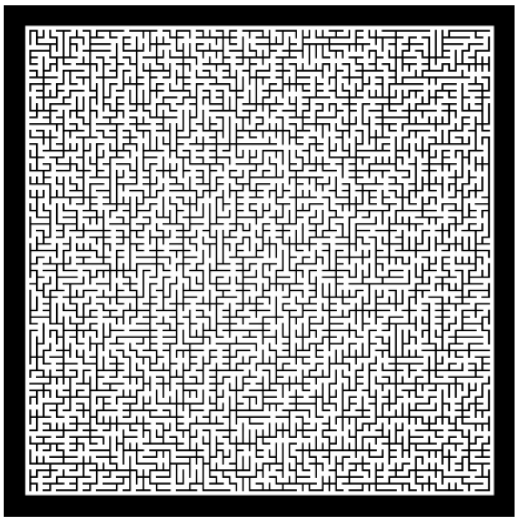

Fig. 1. Entire Maze

III. RELATED WORK

Several approaches have been proposed in the field of drone localization. Some of the related work includes:

Visual SLAM[1]: Simultaneous Localization and Mapping (SLAM) techniques utilize visual information from cameras mounted on drones to estimate their position and create maps of the environment. These methods often combine feature extraction, odometry estimation, and bundle adjustment to achieve accurate localization.

Sensor Fusion: Sensor fusion techniques integrate data from multiple sensors, such as cameras, inertial measurement units (IMUs)[2], and range sensors, to improve localization accuracy. The fusion of visual data with other sensor inputs can provide robust position estimation and mitigate the limitations of individual sensors.

Odometry-based Localization: Odometry techniques estimate the drone's position based on the changes in motion, such as speed, acceleration, and rotation. This information, combined with initial position estimates, allows for continuous localization even in the absence of external references.

IV. INITIAL ATTEMPTS

The initial approach employed in the implementation involved capturing the maze image and the initial snapshot of the drone's environment. I performed template matching

between the maze image and the snapshot to identify potential positions of the drone within the maze but didn't take the movement of the drone into account to give more surety of the position.

V. Final Approach

To localize the drone within the maze, the implemented approach involves template matching techniques. The strategy() function within the code carries out the localization process.

The strategy() function begins by capturing the maze image and the initial snapshot of the drone's environment. It then performs template matching between the maze image and the snapshot to identify potential positions of the drone within the maze.

The strategy() function proceeds to simulate the drone's movement in different directions (horizontally and vertically) and captures snapshots after each movement. Template matching is performed on these snapshots to compare them with the maze image and refine the drone's position estimation.

The function evaluates the template matching results and selects the best matching position based on the obtained results. It draws a rectangle around the estimated drone position on the maze image for visualization purposes.

```
def strategy():
# This function is to localize the
    position of the newly created
    player with respect to the map
Map = player.getMap()

# template matching the original
    position
Snap = player.getSnapShot()
orig = template_matching(Map, Snap)

# template matching after moving
    horizontally
to_right = player.move_horizontal
    (20)
Snap = player.getSnapShot()
right = template_matching(Map, Snap)
right = right - to_right

to_left = player.move_horizontal
    (-20)
Snap = player.getSnapShot()
left = template_matching(Map, Snap)
left = left - to_left

# template matching after moving
    vertically
to_up = player.move_vertical(20)
Snap = player.getSnapShot()
up = template_matching(Map, Snap)
up = up - to_up

to_down = player.move_vertical(-20)
Snap = player.getSnapShot()
down = template_matching(Map, Snap)
down = down - to_down

results = [0, 0, 0]
for i in range(3):
    if orig[i].all() == right[0].all
        ():
         results[i] += 1
    if orig[i].all() == left[0].all
        ():
         results[i] += 1
    if orig[i].all() == up[0].all():
         results[i] += 1
    if orig[i].all() == down[0].all
        ():
         results[i] += 1

# Draw a rectangle around the result
top_left = (
    orig[results.index(max(results))
        ][1],
    orig[results.index(max(results))
        ][0],
)
h, w = Snap.shape
bottom_right = (top_left[0] + w,
    top_left[1] + h)
cv2.rectangle(Map, top_left,
    bottom_right, (0, 0, 255), 2)

# Display the result
cv2.imshow("Template Matching Result
    ", Map)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The strategy() function utilizes the template_matching() function to compare the original snapshot with snapshots after different movements. It keeps track of the three best matching position based on the template matching results.

```
def template_matching(image, template):
    # Load the image and template
    img = image
    temp = template

    # Perform template matching
    result = cv2.matchTemplate(img,
        template, cv2.TM_SQDIFF)

    result2 = np.reshape(result, result.
        shape[0] * result.shape[1])
    sort = np.argsort(result2)

    # returning 3 best matches
```

```
    return np.array(
        [
            np.unravel_index(sort[0],
                result.shape),
            np.unravel_index(sort[1],
                result.shape),
            np.unravel_index(sort[2],
                result.shape),
        ]
    )
```

VI. RESULTS AND OBSERVATIONS

Below are the results:

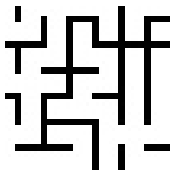

Fig. 2. Original Position

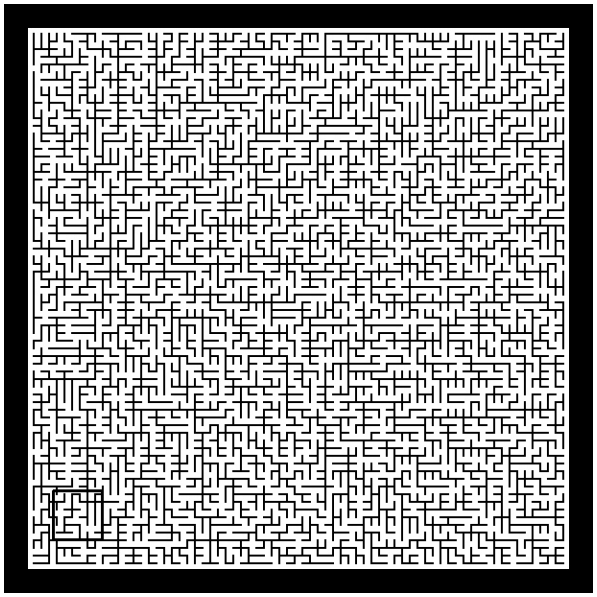

Fig. 3. Position Found

The Player object successfully identifies the position of the drone within the maze every time beyond reasonable doubt.

VII. FUTURE WORK

- Improved Template Matching Techniques: Explore advanced template matching algorithms, such as normalized cross-correlation or scale-invariant feature matching, to enhance the accuracy and robustness of the localization process.
- Integration of Sensor Fusion: Investigate the integration of sensor fusion techniques to combine visual information with data from other sensors, such as IMUs or range sensors. This can improve localization accuracy and robustness, particularly in challenging environments or under changing conditions.

CONCLUSION

This research paper addressed the problem of localizing a drone within a maze using visual snapshots and control actions. The implemented approach utilized template matching techniques to estimate the drone's position relative to the maze.

The strategy() function in the provided code was completed to carry out the localization process. The function analyzed snapshots, performed template matching with the maze image, and refined the drone's position estimation based on movement controls.

The results demonstrated the effectiveness of the implemented approach in estimating the drone's position within the maze. The visualization of the estimated position provided a clear representation of the drone's location.

In conclusion, the research paper presented a solution for drone localization using visual snapshots and control actions. The implemented approach can be further enhanced and extended to accommodate more complex environments and localization scenarios.

REFERENCES

[1] A. Khole, A. Thakar, S. Shende and V. Karajkhede, "A Comprehensive Study on Simultaneous Localization and Mapping (SLAM): Types, Challenges and Applications," 2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS), Coimbatore, India, 2023.

[2] M. Zhang, X. Xu, Y. Chen and M. Li, "A Lightweight and Accurate Localization Algorithm Using Multiple Inertial Measurement Units," in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 1508-1515, April 2020.