Aerial Robotics Kharagpur Tasks
Operation Eagle Eye - The Drone Localization Conundrum

Upendra Singh

***Abstract*— Recent advancements in drone technology necessitate improved autonomous localization methods in unknown environments. This study explores the use of computer vision techniques to localize a drone based on a snapshot of its surrounding area within a simulated map. I implement a template matching strategy using Python and OpenCV to successfully pinpoint the drone's location on the map, demonstrating promising accuracy in controlled simulations.**

I. INTRODUCTION

In this study, I tackle the challenge of localizing a drone in an unknown environment using only a 51x51 snapshot of its surroundings. The snapshot is provided by a satellite, and the task is to identify the drone's exact location on a larger map. This problem is akin to the "lost robot" problem in robotics and computer vision, where the robot must determine its position within a pre-mapped environment using its onboard sensors.

Problem Statement The primary challenge lies in accurately identifying the position of the drone on a map, given only a small snapshot of its immediate surroundings. This situation is complicated by the limited field of view and potential for similar-looking areas within the map that could confuse localization efforts.

Approach My approach to solving this problem involved the development of a template matching algorithm using Python and OpenCV. Template matching is a technique in computer vision used for finding a sub-picture (a template) in an image. This method works by sliding the template image across the main image (the map in this case) and comparing the template with the patch of the image it covers. This comparison is performed using a statistical method; the highest similarity metrics point to the probable location of the template in the main image.

II. PROBLEM STATEMENT

In the Operation Eagle eye , our done was lost in complex maze environment and the GPS signal was lost. E had only snapshots of drone surrounding through the drone onboard camera .We have been access given to the maze map and drone interface and and supporting scripts (player.py, utils.py, MapGeneration.py) .

A. Requirement

1.Implement the strategy() function in player.py, utilizing visual data and control actions of drone to navigate and localize the drone in the maze.
2.Minimize localization error and confidently report Eagle Eye's approximate position within the maze. Following are the control actions of the drone.
• 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

III. INITIAL ATTEMPTS

As a Initial attempts ,I tried to understood the player.py , mapgeneration.py and utilis.py and learn about basic of Object Oriented programming to understand and while implement I misunderstood as implementing drone movement then at last I understood question correctly and implement Template matching algorithm.

IV. FINAL APPROACH

Our final approach to solving the problem of localizing a drone within a simulated environment involved implementing a template matching algorithm using Python and OpenCV. This approach allowed us to accurately determine the drone's position based on a snapshot captured by its onboard camera and compare it with a larger map of the environment.

A. Algorithm Overview

The template matching algorithm works by sliding a template (snapshot) over the larger image (map) and computing the similarity between the template and the overlapping region of the larger image. The position with the highest similarity score indicates the best match, corresponding to the drone's location within the map.

B. Implementation Steps

1) **Image Preprocessing**: Convert the map and snapshot images to grayscale and normalize their intensities to enhance the accuracy of matching.
2) **Template Matching**: Slide the template over the larger image and compute the cross-correlation scores at each position to find areas of high similarity.
3) **Find Best Match**: Identify the position with the highest cross-correlation score as the drone's location within the map. This score indicates the best potential match based on the similarity between the template and the image section.
4) **Visualize Results**: Draw a rectangle around the matched location on the map image for visualization.

This highlights the detected position visually, aiding in verification and analysis.

V. Code Overview

This section explains the functionality of each line of Python code used in the template matching algorithm.

- `import cv2`: Imports the OpenCV library, which provides functions for computer vision tasks.
- `import numpy as np`: Imports the NumPy library, which provides support for numerical operations and array manipulation.
- `from utils import Player, WINDOW_WIDTH`: Imports the Player class and WINDOW_WIDTH variable from the utils module. This module contains utility functions and variables related to the drone simulation.
- `def strategy()`: Defines the strategy function, which is responsible for executing the template matching algorithm.
- `player = Player()`: Creates an instance of the Player class, which generates the map and captures a snapshot of the drone's surroundings.
- `map_image = np.array(player.getMap()) * 255`: Converts the map generated by the Player class to a grayscale image and scales its intensities to the range [0, 255].
- `snapshot_image = np.array(player.getSnapShot()) * 255`: Converts the snapshot captured by the drone to a grayscale image and scales its intensities to the range [0, 255].
- `result = cv2.matchTemplate(map_image, snapshot_image, cv2.TM_CCOEFF_NORMED)`: Performs template matching by comparing the snapshot image with the map image using the normalized cross-correlation method.
- `min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)`: Finds the minimum and maximum values and their corresponding locations in the result of the template matching operation.
- `best_match_loc = max_loc`: Sets the best_match_loc variable to the location with the maximum correlation score, which represents the best match between the snapshot and the map.
- `map_with_match = cv2.cvtColor(map_image, cv2.COLOR_GRAY2BGR)`: Converts the grayscale map image to a BGR (color) image for visualization.
- `cv2.rectangle(map_with_match, best_match_loc, (best_match_loc[0] + WINDOW_WIDTH, best_match_loc[1] + WINDOW_WIDTH), (0, 0, 255), 2)`: Draws a rectangle around the matched location on the map image to highlight it.
- `cv2.imwrite("map_with_match.png", map_with_match)`: Saves the map image with the matched location highlighted as a PNG file.
- `return best_match_loc`: Returns the best match location as the output of the strategy function.
- `if __name__ == "__main__"`: Checks if the script is being run as the main program.
- `best_match_loc = strategy()`: Calls the strategy function to execute the template matching algorithm and stores the result in the best_match_loc variable.
- `print("Best match location:", best_match_loc)`: Prints the best match location to the console.

CONCLUSION

The main objective of the problem to test visual localization techniques and its implication. I used very general approach to this problem solving using Template matching algorithm .

References

[1] https://docs.opencv.org/3.4/d4/dc6/tutorial$_{p}y_{t}emplate_{m}atching.html$