Gebe dich nie auf

M B Thejesshwar

***Abstract*— In this project, I have implemented the RRT-Connect algorithm for path planning in a given maze. The problem involved restoring a distorted image using bitwise operations, performing template matching to extract relevant portions, and finally executing RRT-Connect to find a path between two specified points in the maze. The implementation was done using Python and OpenCV, and all steps were performed without inbuilt template matching or path-planning functions. The approach ensures efficient path discovery while handling real-world constraints.**

I. INTRODUCTION

Path planning is an essential part of robotics, where the goal is to find a feasible route between a start and goal position in an environment while avoiding obstacles. The problem in this case was to:

1) **Restore a distorted image** by applying a **2x2 filter derived from pi.**
2) **Extract a template** using **template matching**.
3) **Apply RRT-Connect** to navigate a maze.

The task required an efficient method to explore paths, and **RRT-Connect** was chosen due to its rapid convergence in complex environments. The approach was implemented and validated using OpenCV and Matplotlib.

II. PROBLEM STATEMENT

1) **Restore an Image:** A given image was distorted using bitwise operations. The task was to reverse this distortion using a predefined 2x2 filter based on the digits of pi.
2) **Perform Template Matching:** Identify the template in a larger collage and determine its position using pixel-wise difference calculations.
3) **Path Planning in a Maze:** Given a maze image with a start and goal position, implement **RRT-Connect** to find a collision-free path.

III. RELATED WORK

Path planning techniques like A* and RRT are commonly used for navigation in robotics. However, RRT-Connect is particularly useful for large search spaces because it expands trees from both the start and goal points, making it computationally efficient. The algorithm has been widely used in autonomous navigation and robotic arms.

IV. INITIAL ATTEMPTS

1) **Restoring the Image:**
   - The filter was extracted using the first few digits of .
   - Each pixel in the distorted image was restored using **bitwise XOR** with the 2x2 filter.
   - A **row-major traversal** was used for filtering.
   - The restored image was verified visually.
2) **Template Matching:**
   - The extracted template was resized to **100x100** pixels for uniformity.
   - A brute-force **pixel-wise absolute difference** method was used to compare image regions.
   - The region with the **minimum difference** was chosen as the best match.
3) **Path Planning:**
   - The maze was loaded as a binary matrix (white = free, black = obstacle).
   - A simple **random sampling** approach was initially tested but was inefficient.
   - RRT-Connect was chosen to ensure faster path convergence.

V. FINAL APPROACH

*a) **1. Image Restoration Algorithm:**:*

- Extract a **2x2 filter** based on :
  - Take the first four digits after the decimal in .
  - Multiply each by **10** and take the **floor**.
  - Arrange them in **descending row-major order** to form a 2x2 matrix.
- Traverse the image in a **2x2 sliding window** and apply **bitwise XOR**.

*b) **2. Template Matching Algorithm:**:*

- Resize the extracted template to **100x100 pixels**.
- Slide the template over the image and compute **absolute pixel-wise difference**.
- Store the coordinates of the position with the **minimum difference**.
- Compute password:
  - Add the template's top-left **x and y coordinates**.
  - Multiply by and take the **floor**.

*c) **3. RRT-Connect Path Planning Algorithm:**:*

- **Step 1: Initialize Trees** One tree grows from the **start**, another from the **goal**.
- **Step 2: Extend Trees**
  - Randomly sample a point in the maze.
  - Connect the closest tree node to the sampled point in a **step size of 10 pixels**.
  - Ensure no collision occurs by checking the **line segment** between points.
- **Step 3: Connect Trees**
  - If a node from one tree gets close to the other tree, attempt to **connect them**.

  - If successful, construct the **final path**.
- **Step 4: Path Extraction** Traverse back through the parent pointers to get the full path.

VI. Results and Observation

- The **restored image** closely resembled the original.
- Template matching successfully located the template at **(x, y) = (100, 100)**.
- The computed password was **(100 + 100) * pi = 628**.
- **RRT-Connect** efficiently found a path, requiring around **2700 iterations**.
- The final path was visualized using Matplotlib, with the start and goal marked in **green** and **red** respectively.

VII. Future Work

- Improve template matching using **cross-correlation** for better accuracy.
- Implement **smoother path planning** using **Bézier curves** to refine RRT-Connect output.
- Extend the method to **3D environments** for drone navigation.

CONCLUSION

The project successfully demonstrated the restoration of a corrupted image, accurate template matching, and effective **RRT-Connect path planning**. The approach was computationally efficient and yielded reliable results. Future improvements can enhance accuracy and extend the implementation to complex robotic applications.

References

[1] OpenCV Documentation - Bitwise Operations: https://docs.opencv.org/master/d0/d86/tutorial_py_image_arithmetics.html

[2] NumPy Documentation - Array Manipulation: https://numpy.org/doc/stable/reference/generated/numpy.ndarray.htmlg/master/d0/d86/tutorial_py_image_arithmetics.html

[3] RRT-Connect Algorithm Reference: https://cs.stanford.edu/people/nielsen/rlr-rrt.pdf

[4] Template Matching Implementation: https://docs.opencv.org/master/df/dfb/groupimgprocobject.html

[5] Path Planning in Robotics - RRT Overview: https://www.kuffner.org/james/papers/rrt-connect.pdf