Path planning algorithm for cluttered environments using computational geometry approaches

Project Report

Submitted by

Shreyash Patidar
ME18B074

in partial fulfilment of requirements
for the award of the dual degree of

BACHELOR OF TECHNOLOGY in
MECHANICAL ENGINEERING

and

MASTER OF TECHNOLOGY in
ROBOTICS

DEPARTMENT OF MECHANICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI 600 036

January, 2023

CERTIFICATE

This is to certify that the project titled **Path Planning algorithms for cluttered environments using computational geometry approaches** submitted by **Shreyash Patidar (ME18B074)** to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology in Mechanical Engineering and Master of Technology in Robotics**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

	Shreyash Patidar
Prof. M. Ramanathan	(ME18B074)
Name of Guide:	**Name of student**
Signature:	Signature
	[illegible]

ABSTRACT

KEYWORDS: Path Planning, Computational Geometry, Voronoi Diagram

This project work involves developing a new algorithm for robotic Path Planning using Computational Geometry approaches. There are existing algorithms in field of Path Planning. Some of them decompose obstacle course into grid of evenly sized cells and others considers the geometry of each obstacle. Optimization methods and search algorithms are applied over it to find the best possible paths.

This algorithm considers the geometry of the obstacles. And the algorithm is inspired from the Voronoi Diagram based algorithms. The algorithm starts with approximating the obstacles with ellipses (based on the order of accuracy). Internal tangents between each pair of ellipses are used to identify nearest neighbours of each elliptical obstacle. For each pair of these nearest neighbours, a quadrilateral unit is defined in which one pair of opposite sides are the chords of ellipses and the other pair of opposite sides can be used by robotic agent to enter or exit these quadrilateral regions. The obstacle course is then represented in terms of these quadrilateral units. Graph search algorithms are then used to find paths in terms of these quadrilateral units.

This algorithm is expected to perform better than the grid search algorithms, as it condenses large amount of pixel/ grid information into a lesser number of nodes; keeping the geometry data intact. Thus lesser computational time in the graph traversal/search step. It is expected to perform even better in cluttered environments and in environments where obstacle geometry is of the order of size of robotic agent, and obstacle geometry plays a crucial role.

This algorithm can further be extended to 3D Path Planning, where obstacles will be approximated as ellipsoids instead of ellipses. And quadrilateral units will be replaced by frustums. And 3D paths will be planned in terms of these frustum regions.

Table of Contents

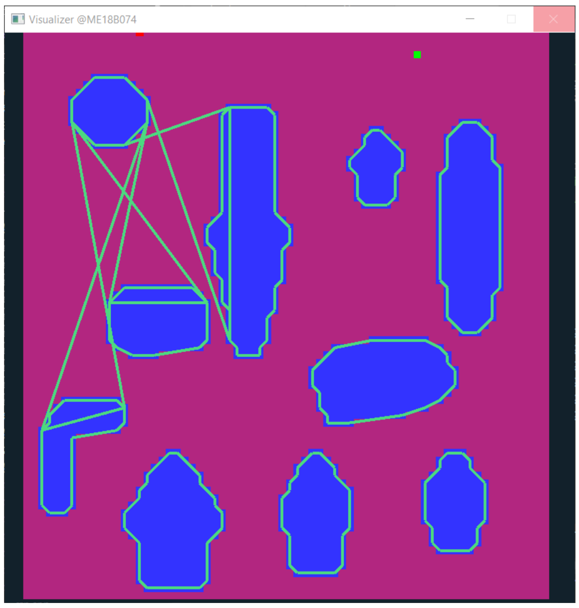
List of Figures

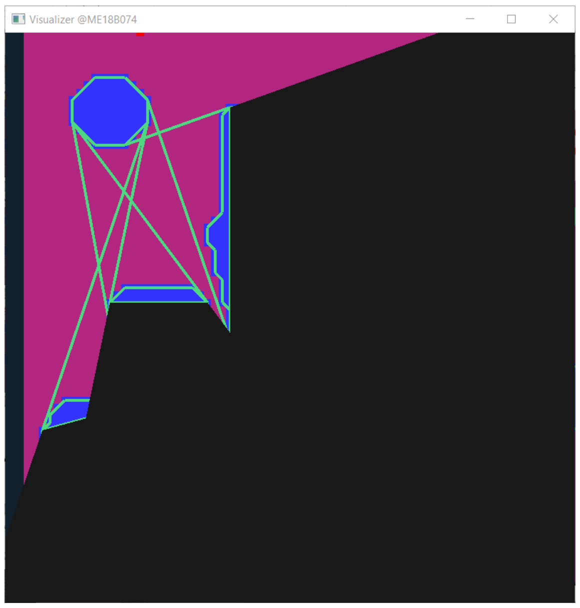
PATH PLANNING

Path planning is one of the most crucial technologies that make a robotic agent to take decisions autonomously. It is a way of finding the best suitable paths for a robotic agent. The simplest problem involves finding the path for a point object in the obstacle course. The more involved ones deals with finding the orientations and actions that the agent should perform to reach its goal pose.
This project work involves designing a new data-structure to represent nodes and obstacles, and building graphs to handle obstacles in cluttered environment more efficiently. The aim is to exploit the concepts of Computational Geometry to handle the geometry of obstacles more effectively, and at the same time using elliptical approximations to address the problem of higher order of Complexity, of Geometry based path planning algorithms.

Background

The work started with understanding existing geometry based algorithms and especially Voronoi diagram based algorithm. It was found that if local Voronoi diagram can be constructed, it will be easier and faster to traverse graphs and find paths. So I started with representing obstacles as 2D polygons and for each such obstacle; eliminating those obstacles whose edges won't contribute in the Voronoi diagram with the selected obstacle. Thus, identifying the pairs of "nearest-neighbours" obstacles.





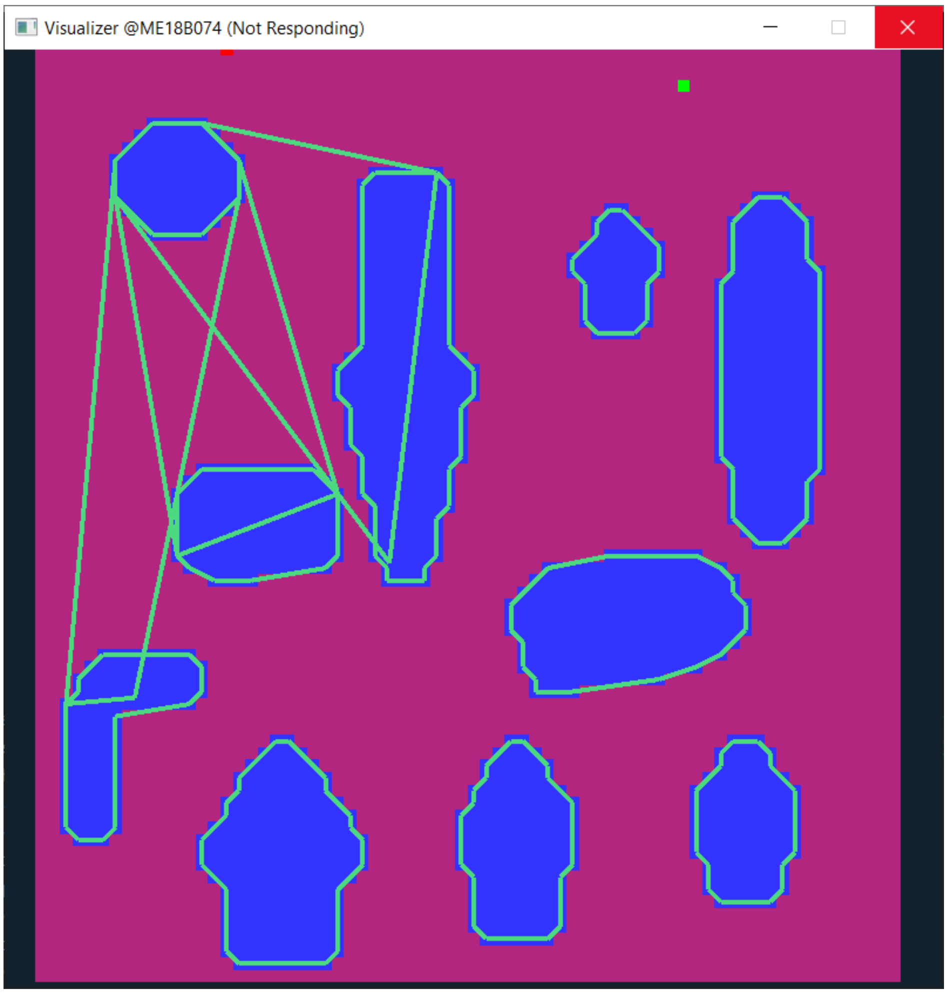


Figure 1: Nearest neighbours of an obstacle and their shielding

For each pair of these “nearest-neighbours”, a quadrilateral region is defined. When an agent is crossing this quadrilateral area, only the pair of corresponding obstacles will interact with the agent, i.e., only the geometry of these two obstacles will define the shape of planned path inside this quadrilateral.
Later it was found that representing the complete map using these quadrilaterals is enough to find paths around the obstacles.

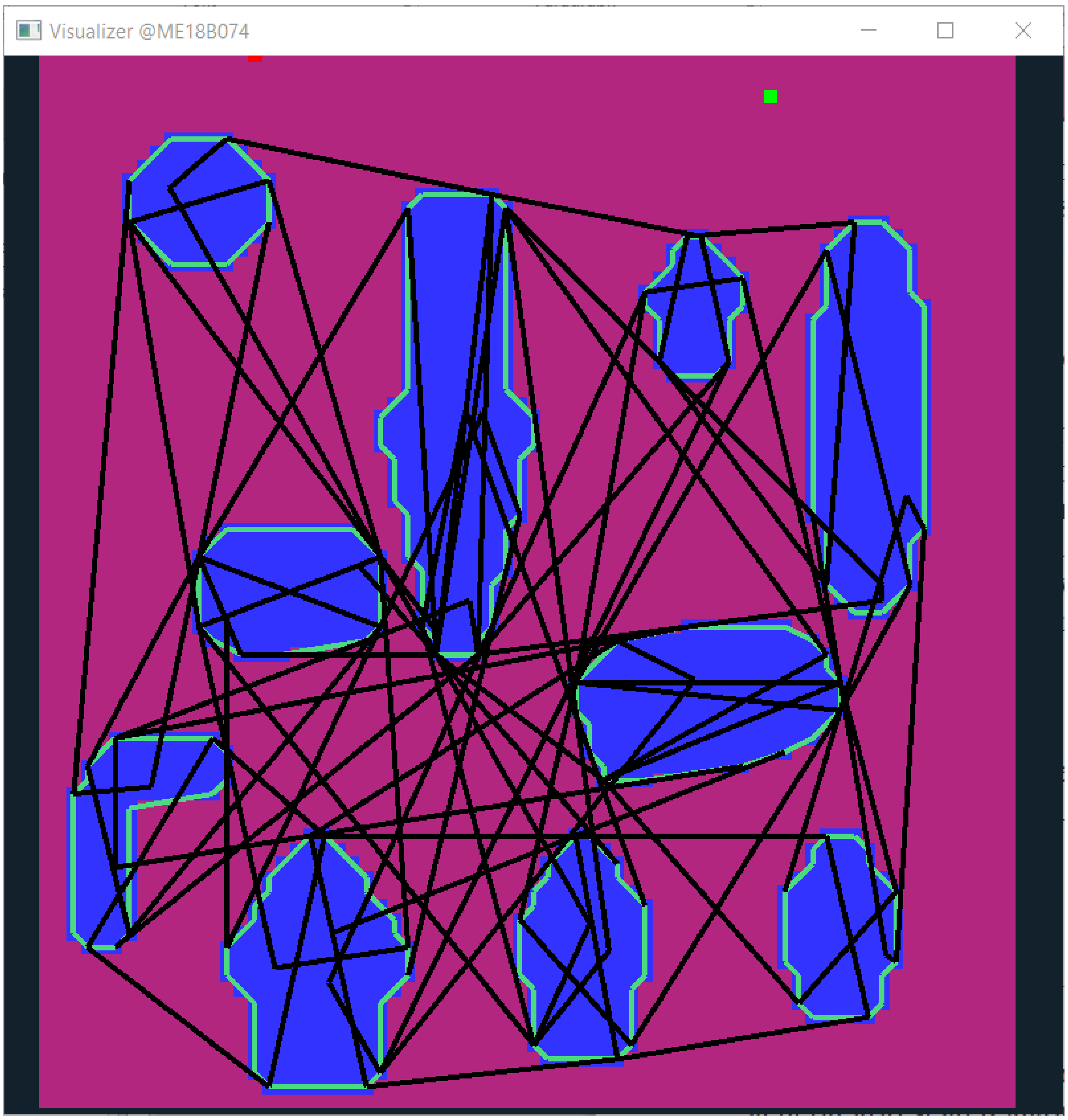


Figure 2: Map representation by quad data structure

Further advancements in this algorithm involve representing these obstacles as ellipses instead of polygons. This will save a lot of computational time in tangent computation step and finding local paths inside the quadrilaterals. And later this algorithm can be extended to 3D world where obstacles will be approximated as ellipsoids instead of ellipses. And quadrilateral units will be replaced by frustums. And 3D paths will be planned in terms of these frustum regions.

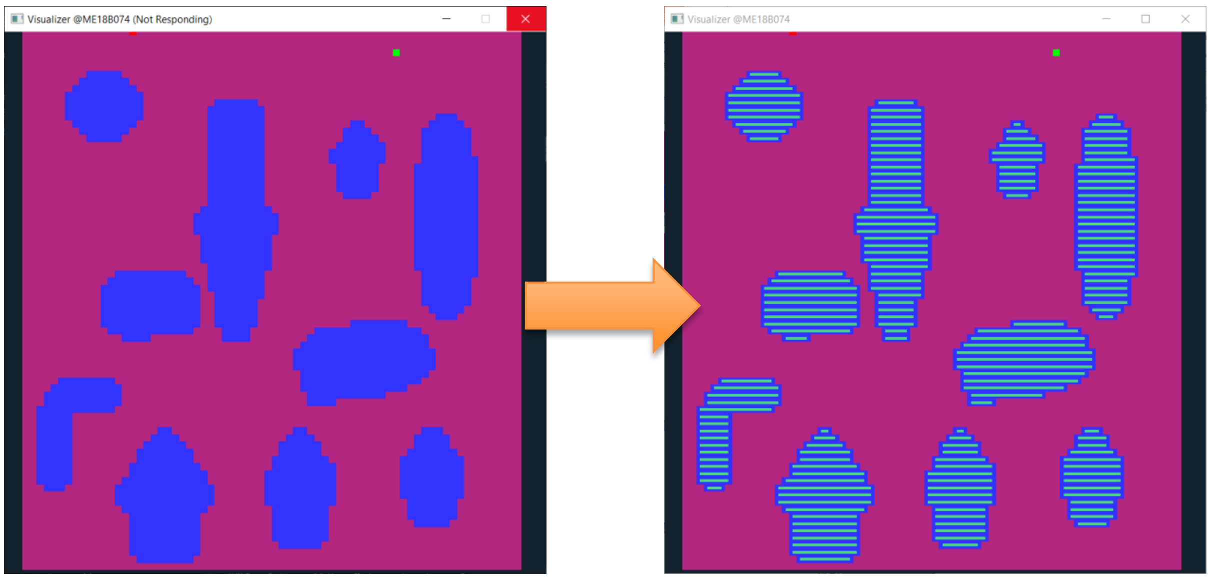
Algorithm

Inputs:

- It takes binary images of obstacle course as a input where obstacles will be represented by black colour and white colour represents free area.
- For real-time applications, video frames will be passed to image processing libraries to get masked images which can then be read by this algorithm.

Step I: reading the data and storing it as polygons

- Read image row by row
- Identify the strip-obstacles in each row; store these strip data in vectors
- Traverse through these vectors top down; and keep merging the strips



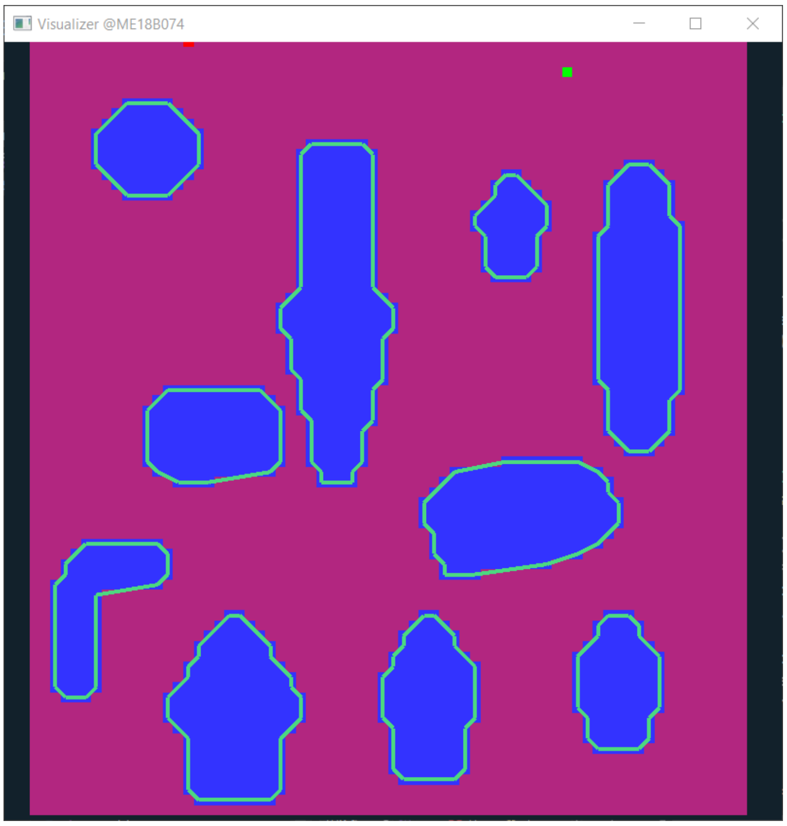


Figure 3: Locating obstacles from binary image

Step II: "Nearest-neighbour computation":

- Iterate through the list of obstacles:
    - For each obstacle, iterate through the remaining n-1 obstacles in order of the least distance from the selected obstacle
        - Check if this obstacle gets shielded by the neighbours of chosen obstacle.
        - If this obstacle contributes to the voronoi diagram of chosen obstacle; add it to the neighbours list of chosen obstacle.

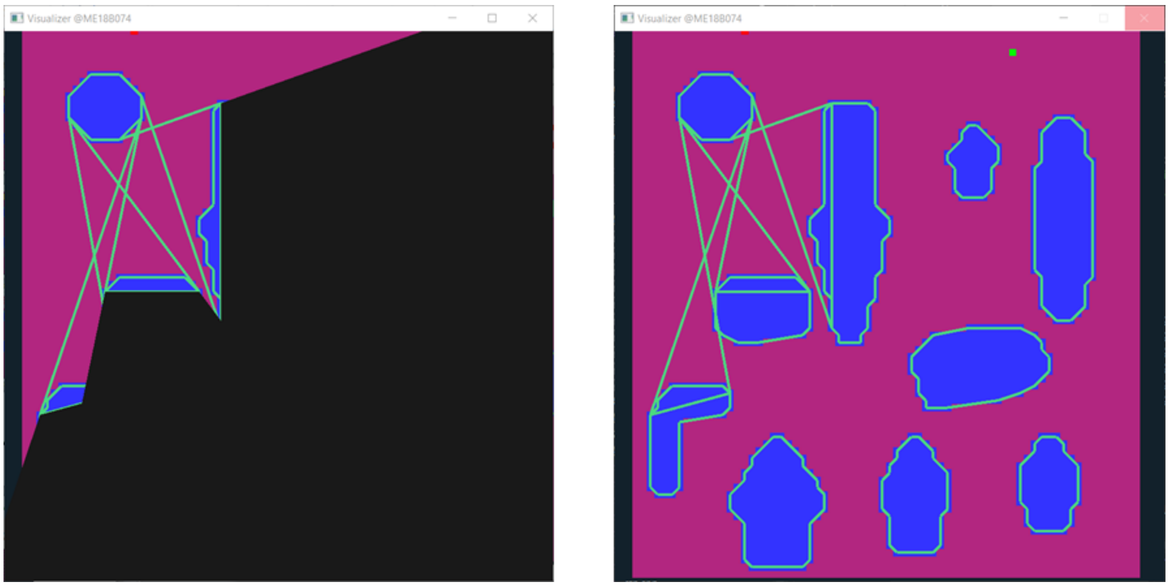


Figure 4: Obstacles falling in black region doesn't contribute to Voronoi edge with the first obstacle

Shielding Check :

Lets say, in outer loop; obstacle 1 is being iterated through.
(remaining n-1 obstacles are to be iterated in order of least distance from obstacle 1).
Lets assume that obstacle 2 and obstacle 5 have been already iterated through resulting in addition of these obstacles to neighbours list of obstacle 1.

- For the next coming iterations; remaining obstacles needs to be checked if they get shielded by neighbours of obstacle 1, ie, obstacle id 2 and 5.
- For shielding check; find the internal tangents between obstacle 1 and obstacle 2.
- If the rectangular bounds of kth obstacle falls fully inside the shielding region formed by internal tangents; kth obstacle will no more share the voronoi edge with obstacle 1.
- In case of partial shielding; look for the diagonals of rectangular bounds. Delete the part of these diagonals which is shielded by obstacle 2. And continue with the algorithm.
- While iterating through neighbours of chosen obstacle (obstacle 1), if these diagonals get completely deleted at the end. The kth obstacle will no more contribute to voronoi edge. Otherwise in case of partial or complete exposure; add the kth obstacle to neighbours list of chosen obstacle (obstacle 1).

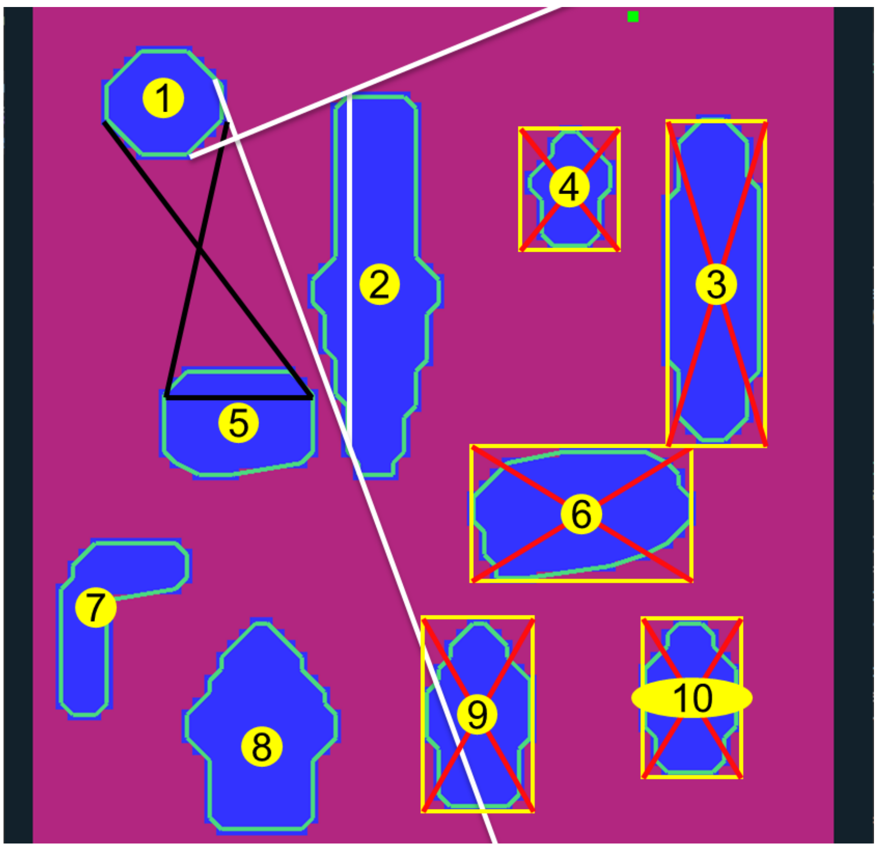


Figure 5: Shielding Check (Complete shielding)

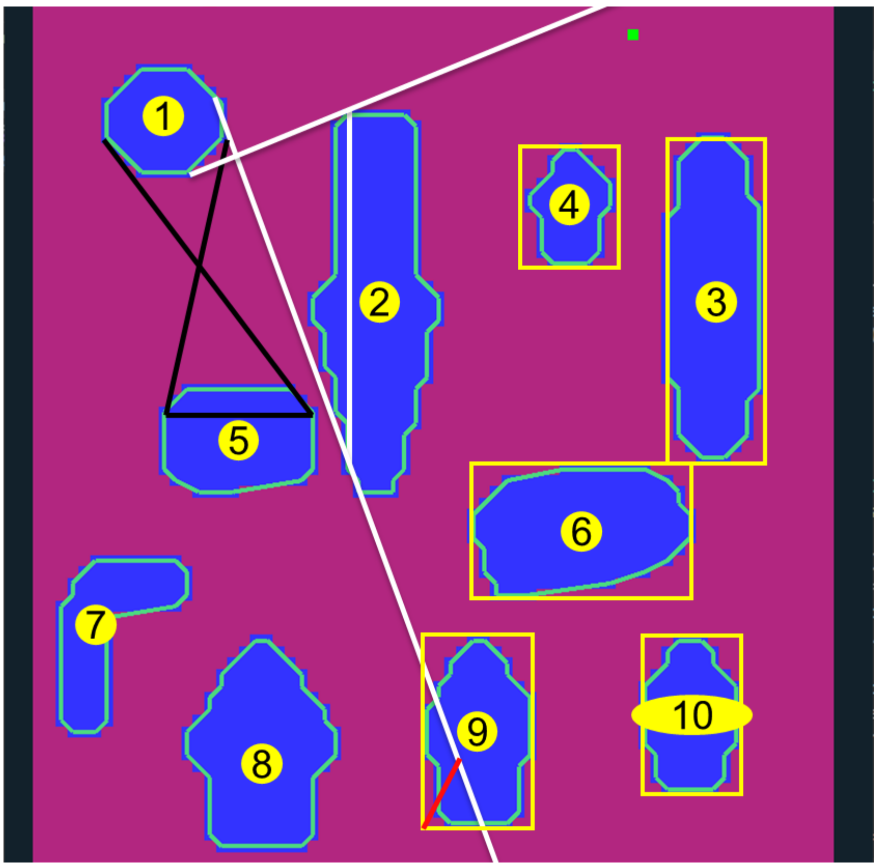


Figure 6: Shielding Check (Partial Shielding)

Outputs at the end of step 2:

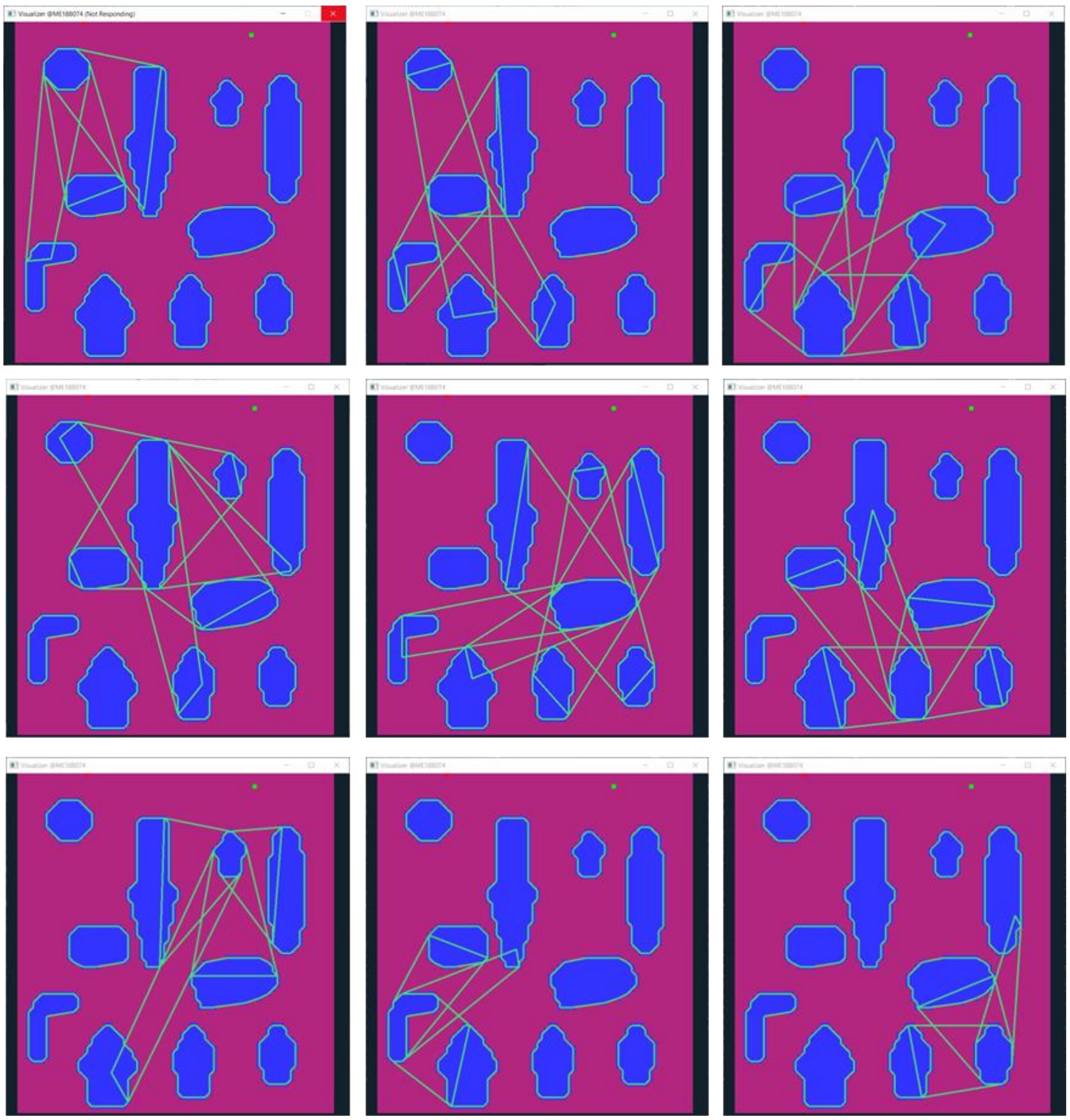


Figure 7: Outputs from Step II

Quad Data-structure:

- For each pair of nearest-neighbours, a quadrilateral region is defined by joining the chords formed by internal tangents between these polygons.
- This quad data-structure is special by the fact that; one pair of opposite sides reperesents obstacles and other pair of opposite sides represents free space.
- Hence a robotic agent can enter this quad space only through a specific pair of opposite sides. Similarly it can go from one quad to another only if the sides corresponding to its free spaces intersects with one another.
- And the shape of path inside this quad is only defined by the geometry of the nearest-neighbours.

Step III: Graph building using quad data-structure:

- Based on intersection criteria of quads, a bi-directional graph is created, taking these quads as the nodes.
- Also, these quads are ranked on the basis of their geometries. And stored in a sorted order according to ranks. Bigger the area of the quad, or the lengths of free sides; better will be the rank of that quad. And in any neighbour list / traversal order, the quad will appear the earlier.

*** <u>All the steps till here are a one-time computation. And it eases the path search for upcoming steps.</u>

Step IV: Path search

- The quad-graph is now searched for suitable paths.
- Preference is given to nodes with higher rank. They have been already sorted according to areas and length of free-side. So first few nodes itself is supposed to cover maximum area, thus attacking the problem in greedy manner.
- Once we get paths in terms of these quad data-structures.
- Local path planning methods are used to plan out subsequent paths inside these quads. And based on criteria such as number of turns, straightness, width and length of path; best path is sent as output.

Outputs from some failed and successful attempts :

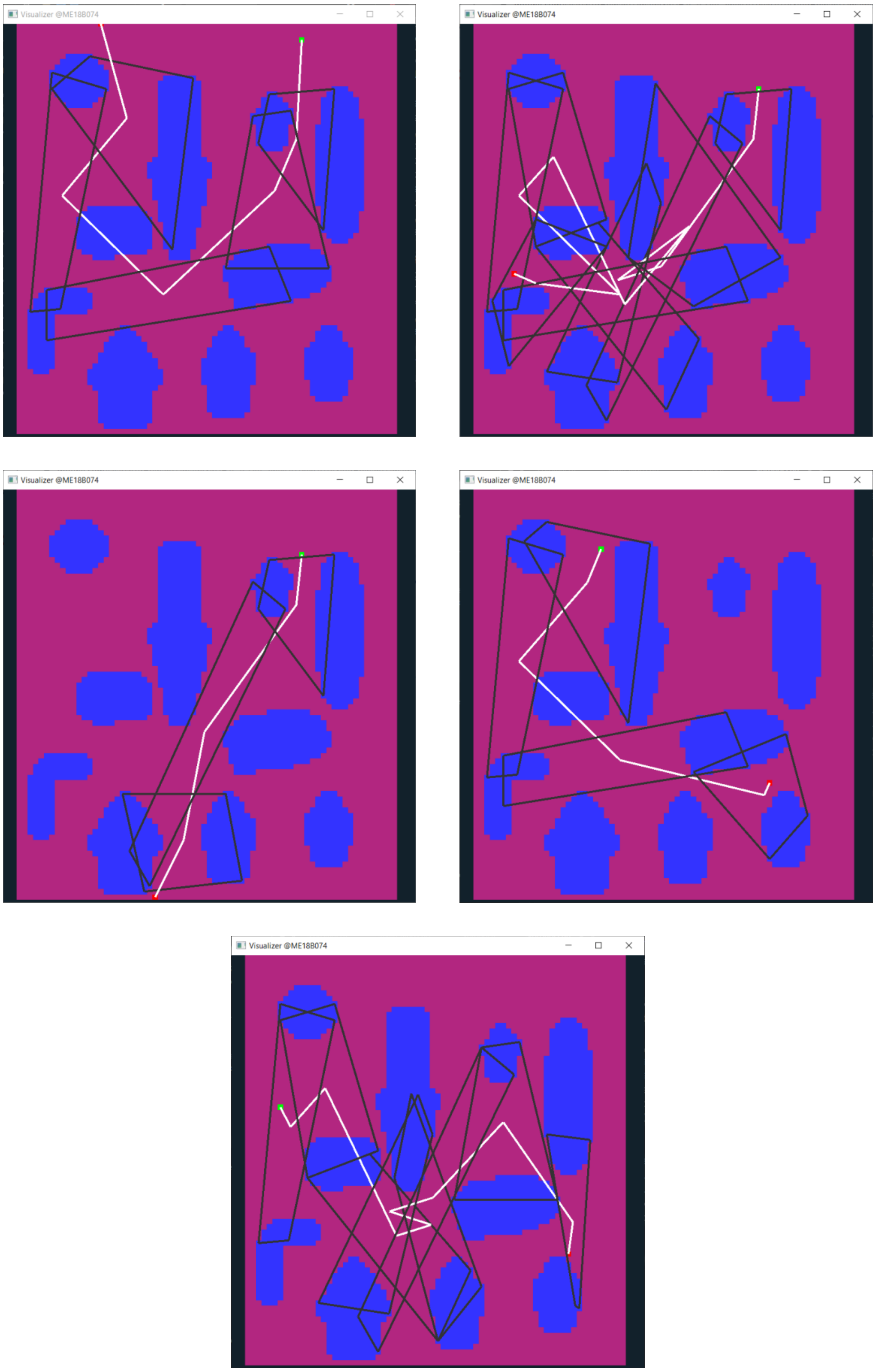


Figure 8: Results from current version of algorithm

A peek into the final optimized algorithm:

Category	Algorithm	Complexity
Pre-processing	Data reading	Complexity : O(m*n) For m*n pixel image
	Storing obstacle data as strips	O(m*n)
	Merging strips into polygons	Complexity : O(m*k) Average number of polygons in one row : k
Nearest-neighbours	Iterating for shielding test	O(p*p*l) p : no. of obstacles l : average number of neighbours of an obstacle
	Tangent Computation	O(v^4); v : number of vertices in contributing polygon(obstacle)
	Shielding test : fully shielded case	O(constant)
	Shielding test : partial shielding case	O(l); l : average number of neighbours of an obstacle
Map Building	Quad intersection check	O(constant)
	Tree Building	
Path Search	Tree Traversal	
	Local Path Planning	

Figure 9: Algorithm Summary

Venues of Improvement and Further Research:

- Approximation of polygons as ellipses without losing geometry information, and keeping the complexity with limits.
- Improving the method of ranking quads to achieve better, shorter paths and in even lesser time.
- Extending the algorithm to 3D environments.

References

1. Sanjana Agrawal & R. Inkulu, (2022), **Visibility polygons and visibility graphs among dynamic polygonal obstacles in the plane**, Journal of Combinatorial Optimization volume 44, pages 3056–3082
2. Mandalika, Aditya Vamsikrishna, (2021), **Efficient Robot Motion Planning in Cluttered Environments**
3. Lounis Adouane, Ahmed Benzerrouk, Philippe Martinet, (2016), **Mobile Robot Navigation in Cluttered Environment using Reactive Elliptic Trajectories**, LASMEA, UBP-UMR CNRS 6602, France
4. Armin Hornung, Mike Phillips, E. Gil Jones, Maren Bennewitz, Maxim Likhachev, Sachin Chitta, (2012),**Navigation in Three-Dimensional Cluttered Environments for Mobile Manipulation**, IEEE International Conference on Robotics and Automation.