Path planning algorithm for cluttered environments using computational geometry approaches

Project Report

Submitted by
Shreyash Patidar
ME18B074

in partial fulfillment 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 **ShreyashPatidar (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.

	ShreyashPatidar
Prof. M. Ramanathan	(ME18B074)
Name of Guide:	**Name of student**
Signature:	Signature

ABSTRACT

KEYWORDS: Path Planning, Computational Geometry, Voronoi Diagram, 3D environment

Path planning is one of the most crucial technologies that make a robotic agent take decisions autonomously. However, Path planning by computational geometry means is not so frequently visited topic.This project work involves developing a new algorithm for robotic Path Planning using Computational Geometry approaches. There are existing algorithms in the field of Path Planning. Some of them decompose obstacle course into a grid of evenly sized cells and others consider the geometry of each obstacle. Optimization methods and search algorithms are applied to it to find the best possible paths. There is a well-known issue with geometry-based algorithms, their computational complexity. At the same time, it can generate results with little to no approximations. This work is an effort to crack a balance between these approximations and complexity. This algorithm approximates obstacles as ellipses, runs nearest-neighbors tests to get Voronoidiagram-like results, and runs path search traversals to find optimal paths. As the algorithm focuses more on geometry, the algorithm will find its use in cluttered environments or in places where obstacle geometry is of the order of size of the robotic agent and obstacle geometry plays a crucial role.And the approximations as ellipses will boost the speed manyfold. Parametric representation of ellipses will bring down tangent computation complexities from order 3 and 4 to constant time operations. Similarly, it'll bring down complexities in the path search and optimization step.Some uses-cases of the algorithm can include robots navigating busystreets and moving packages in small-scale, unorganized warehouses. Later the algorithm can be extended for 3D environments where it will find use in planning missions for underwater caves, low-height search operations for drones, simulating machine movements for rescue operations, and operations in tightspaces like mines.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Path planning is one of the most crucial technologies that make a robotic agent 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 deal with finding the orientations and actions that the agent should perform to reach its goal pose.

This project work involves designing a new datastructure to represent nodes and obstacles, and building graphs to handle obstacles in the 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 use elliptical approximations to address the problem of higher Complexity of Geometry based path planning algorithms.

1.1 Background

The work started with understanding existing geometry-based algorithms and especially the Voronoi diagram-based algorithm. It was followed by experimenting with datastructures, checks, and conditions and ultimately shaping into a new algorithm.This algorithm starts with approximating the obstacles as ellipses (based on the order of accuracy). Internal tangents or non-extreme tangents between each pair of ellipses are used to identify the nearest neighbors of each elliptical obstacle. For each pair of these nearest neighbors, 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 the 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 a 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.

CHAPTER 2

PROBLEM STATEMENT

The project work aims at creating a new algorithm in Path Planning using Computational Geometry approaches. And building a complete end-to-end package, that can read obstacle/map data from input visual feeds/ images and output planned path along with all metrics and details that might be required by the robotic agent.

This involves several smaller algorithms to load, and handle obstacle data. And representations for easy traversals and core geometric concepts to ease out complexities from existing steps in conventional algorithms. Some important algorithmic aspects include:

- Approximation of obstacles as ellipses within accuracy limits.
- Geometry-based checks to filter out "nearest-neighbors" obstacle pairs.
- Creating a new quad data structure to efficiently handle and use nearest-neighbor information.
- Strategies to assign weights to these quads based on their geometry and geographical position.
- Extending the algorithm to make it compatible with 3D environments.

CHAPTER 3

LITERATURE REVIEW AND RELATED WORKS

Path planning algorithms are used by mobile robots, unmanned aerial vehicles, and autonomous cars in order to identify safe, efficient, collision-free, and least-cost travel paths from an origin to a destination. Based on the use case, several approaches can be used to plan paths. These algorithms can be categorized in many different ways.

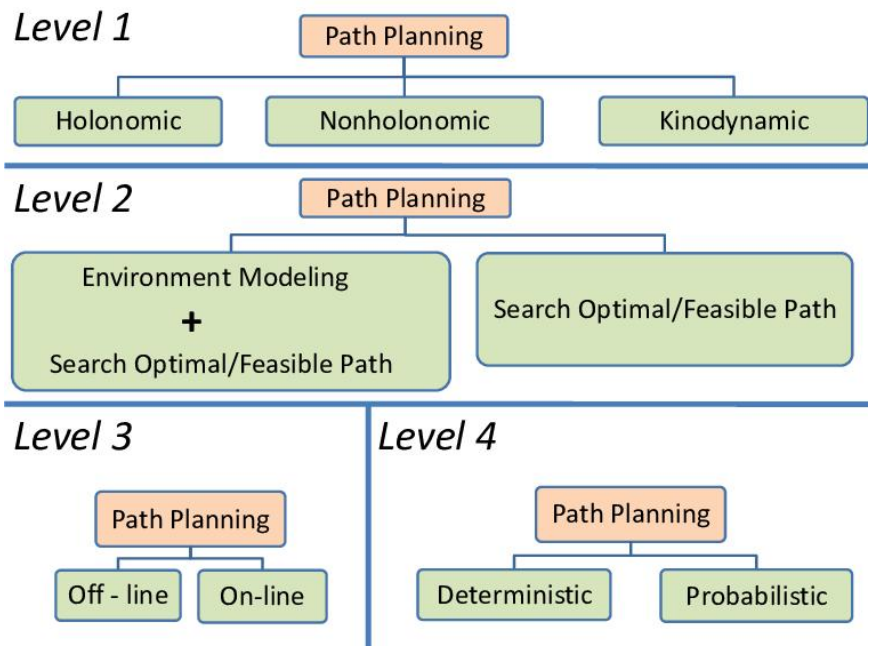


Figure 1 : Various classifications of Path Planning Algorithm

Based on the nature of the algorithm, broadly these can be classified as:

- Grid Based Search
- Geometry Based Algorithms
- Sampling-Based Algorithms
- Artificial Potential Algorithm

3.1 Visibility Graph

A visibility graph is a graph of inter-visible locations, typically for a set of points and obstacles in the Euclidean plane. Paths are searched in terms of line segments instead of points or gridcells.

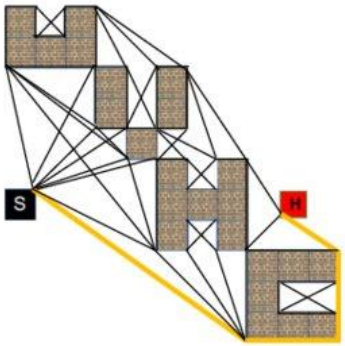


Figure 2 : Visibility graph

3.2 Voronoi Graph

The Voronoi diagram is a partition of a plane into regions close to each of a given set of objects and the smallest area entity resulting from the intersection of edges is called a Voronoi cell. The path search methods to plan the path in terms of edges and cells from this Voronoi diagram are termed as Voronoi-based graph search algorithms.

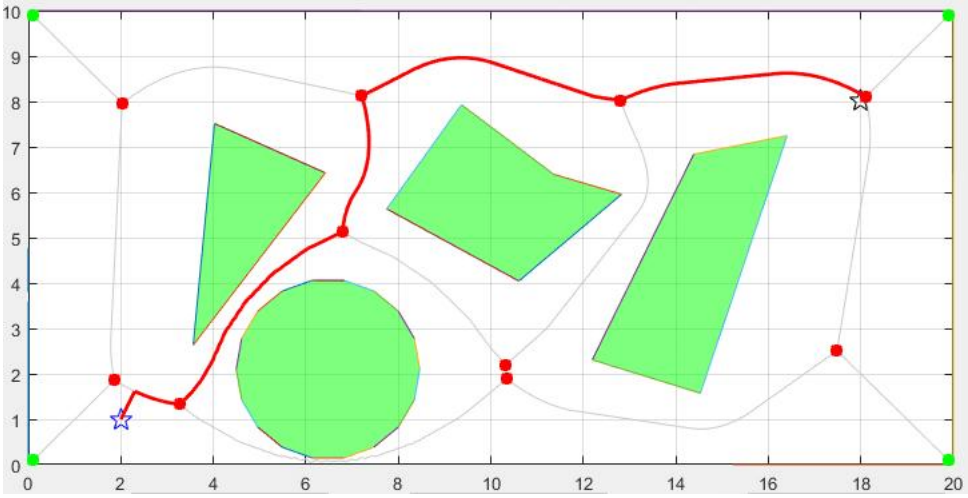


Figure 3 : Voronoi Graph

In our algorithm, some features have been picked from these algorithms to formulate our version. In visibility graphs (or VGRAPH), free space is represented as line segments, and paths are planned in terms of these lines. Similarly, a quad structure is used to represent freespace in our algorithm, and graphs are constructed accordingly. In the Voronoi graph method, a Voronoi diagram is generated first to get information regarding neighboring obstacles of a given obstacle. Here we are using some geometrical checks to rule out far-off obstacles and identify neighboring obstacles.

The idea of representing obstacles as ellipses has already been thought off and worked upon. LounisAdouane, Ahmed Benzerrouk and Philippe Martinet(2011) in their paper on **Mobile Robot Navigation in Cluttered Environment using Reactive Elliptic Trajectories** have approximated walls and other obstacles as ellipses.

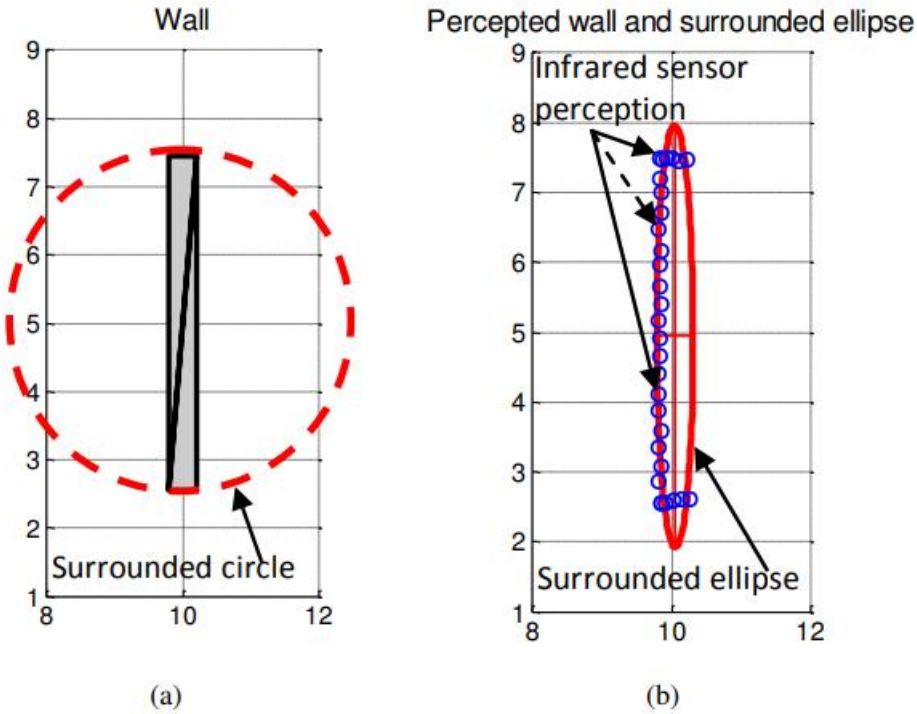


Figure 4 : Interpolated wall using a circle and an ellipse shape

Taking inspiration from this work, we are going to approximate obstacles similarly. But rather than approximating each obstacle as 1 ellipse, we are using a set of differently sized intersecting ellipses to capture the geometry more efficiently.

CHAPTER 4

APPROACH

This algorithm is inspired by Voronoi diagrams to a great extent, it runs very close to method of creating voronoi diagrams. Voronoi diagrams of the polygon are created by first assuming each line is a separate entity. In the next pass, each Voronoi edge falling inside the polygon or intersecting the polygonal edge is cleared out. Thus resulting in vornoi diagram for 2D polygons.

This involves unnecessary computation as well, computation of voronoi edges between far-off points. Our algorithm tries to find pairs of nearest neighbour polygons first and run the voronoi edge algorithm to these selected points. Thus saving on computational power.
These nearest neighbours are identified using a simple geometrical test, which is developed as a experimental finding during the course of this project work.

4.1 Geometrical Test to Filter out far-off polygons

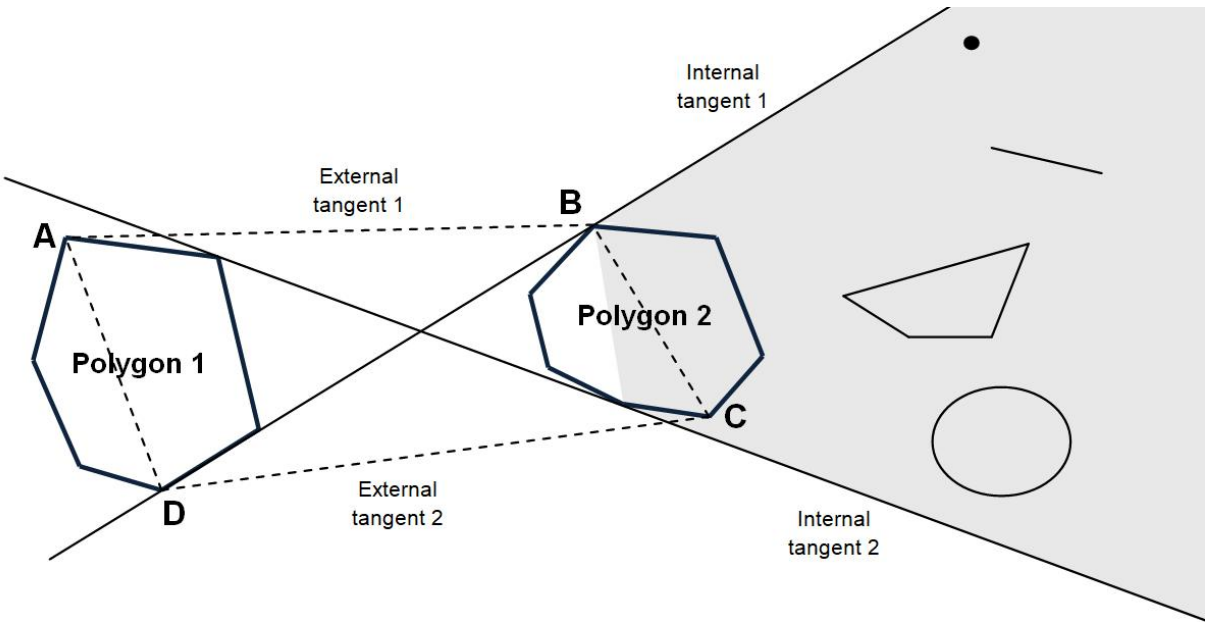


Figure 5 : Polygon Shielding

For a **chosen polygon (polygon 1 in Fig 5)**, any other polygon (polygon 2 in Fig 5) shields all the features in the highlighted area such that any portion of its voronoi edge (in the voronoi diagram constructed for all the polygons as a whole) with polygon 1 will not lie inside the quadrilateral ABCD.

For any point inside the quadrilateral ABCD, polygon 2 is closest compared to any shielded feature. Hence the shielded feature (point/ line/polygon/curve) cannot have its Voronoi region inside quadrilateral ABCD. Thus, for Voronoi computations inside region ABCD, these features can be safely ignored.

4.2 Quad data structure

Each pair of closest neighbouring polygons can be represented by a quadrilateral formed by their external tangents and the respective chords. The tangents here are highlighted in green, and the chords in red colour (in Fig 6).

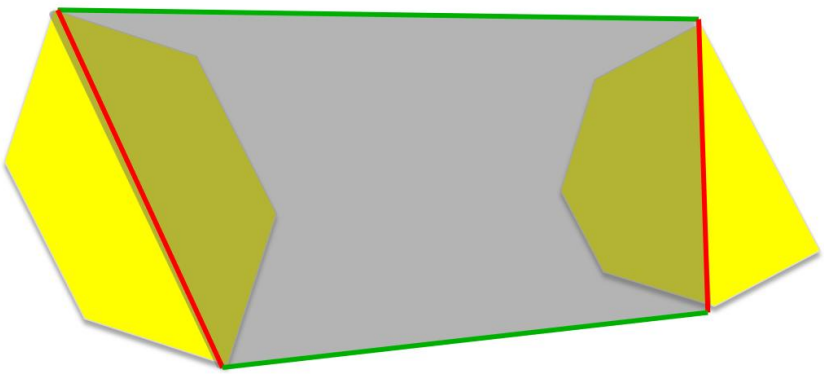


Figure 6 : Quad Representation

Constraints involved:

- A robotic agent can enter or exit this quadrilateral region through green edges only.
- It can never cross/enter or exit through red edges.
- Local paths inside this region are determined by the geometry of the two polygons, i.e., the closeness and geometry of these polygons influence the maneuverability of the robotic agent inside this region.

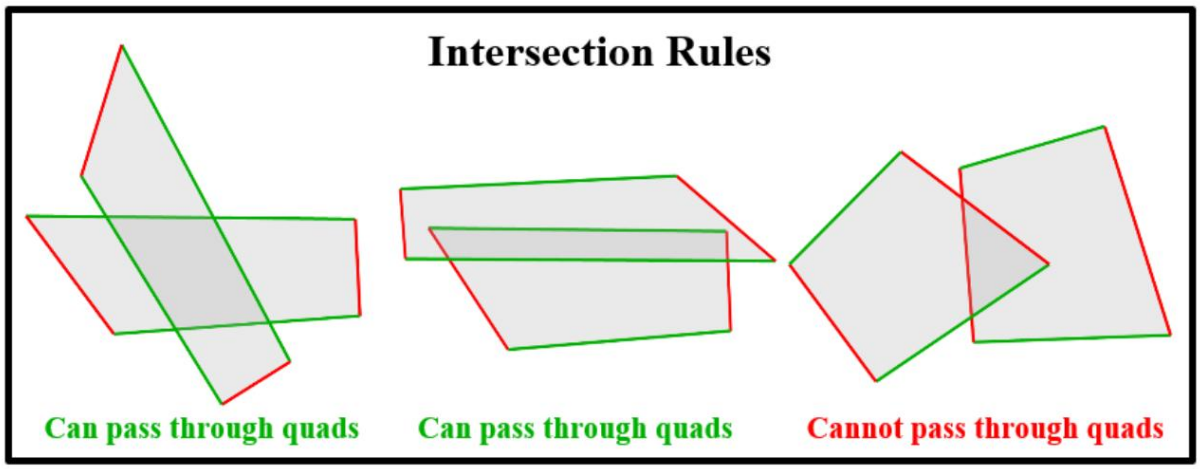


Figure 7 : Intersection Rules – Quad Structures

Intersection Rules :

- If the quads intersect each other on the green edges, the robotic agent can pass from one quad to another (as shown in case I in Fig 7)
- If quads intersect at red edges, it cannot pass through the other quad (case III in Fig 7)
- The quads can intersect at a maximum of four points; if green lines are involved in at least 2 of these intersection points, the agent can pass from one quad to another (case II in Fig 7)

CHAPTER 5

ALGORITHM

5.1 Inputs:

- It takes binary images of an obstacle course as input where obstacles will be represented by black color and white color represents the 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.

5.2 Step I: reading the data and storing it as polygons

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

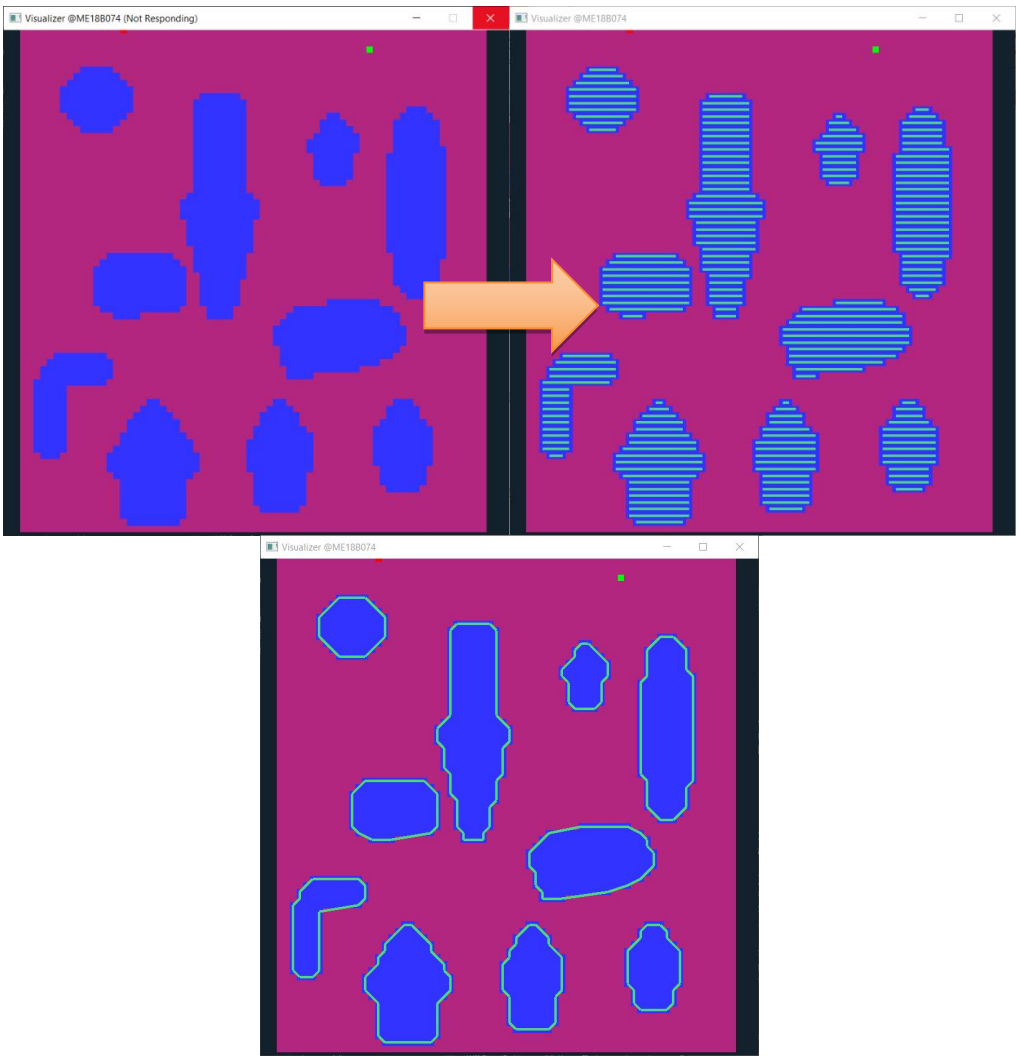


Figure 8 : Locating obstacles from the binary image

5.3 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 neighbors of chosen obstacle.
        - If this obstacle contributes to the Voronoi diagram of chosen obstacles; add it to the "neighbors" list of chosen obstacle.

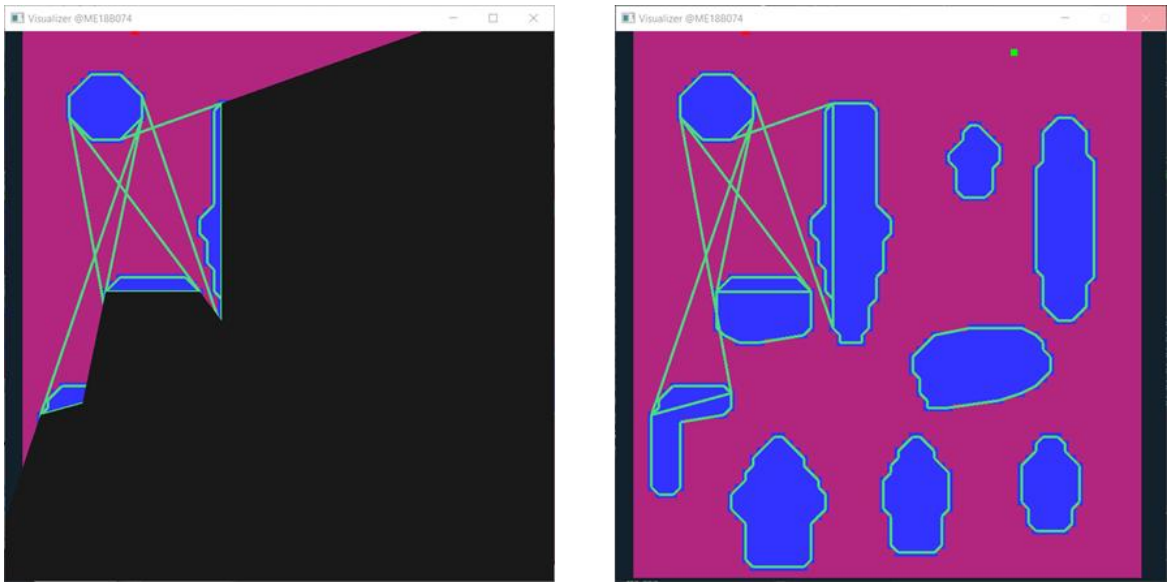


Figure 9 : Obstacles falling in the black region don't contribute to Voronoi edge with the first obstacle

Shielding Check :

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

- For the next coming iterations; the remaining obstacles need 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 the kth obstacle fall fully inside the shielding region formed by internal tangents; the kth obstacle will no more share the Voronoi edge with obstacle 1 (next page).
- In the 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 the Voronoi edge. Otherwise in case of partial or complete exposure; add the kth obstacle to the neighbours list of chosen obstacle (obstacle 1).

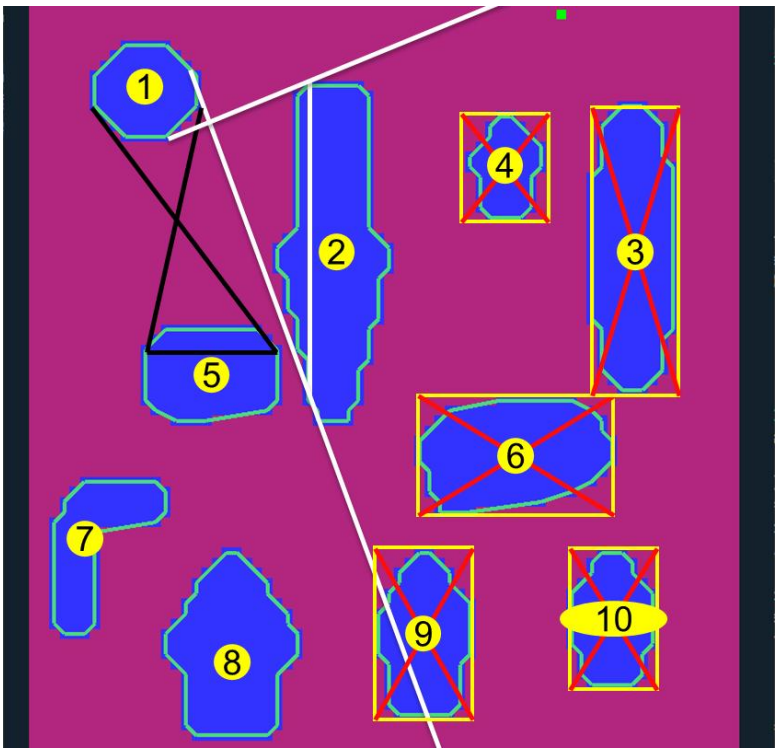


Figure 10 : Shielding Check (Complete Shielding)

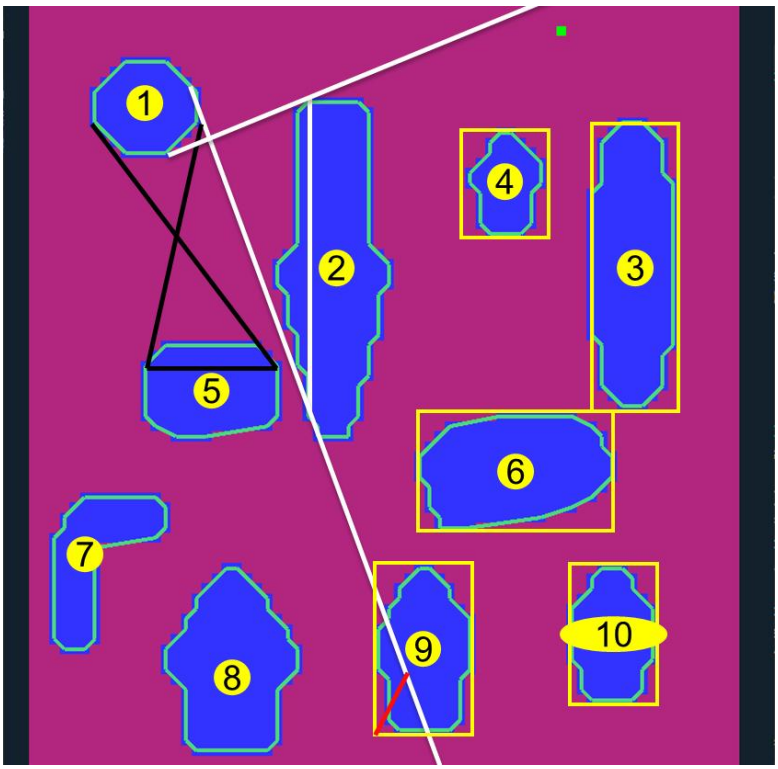


Figure 11 :Shielding Check (Partial Shielding)

Outputs at the end of step 2:

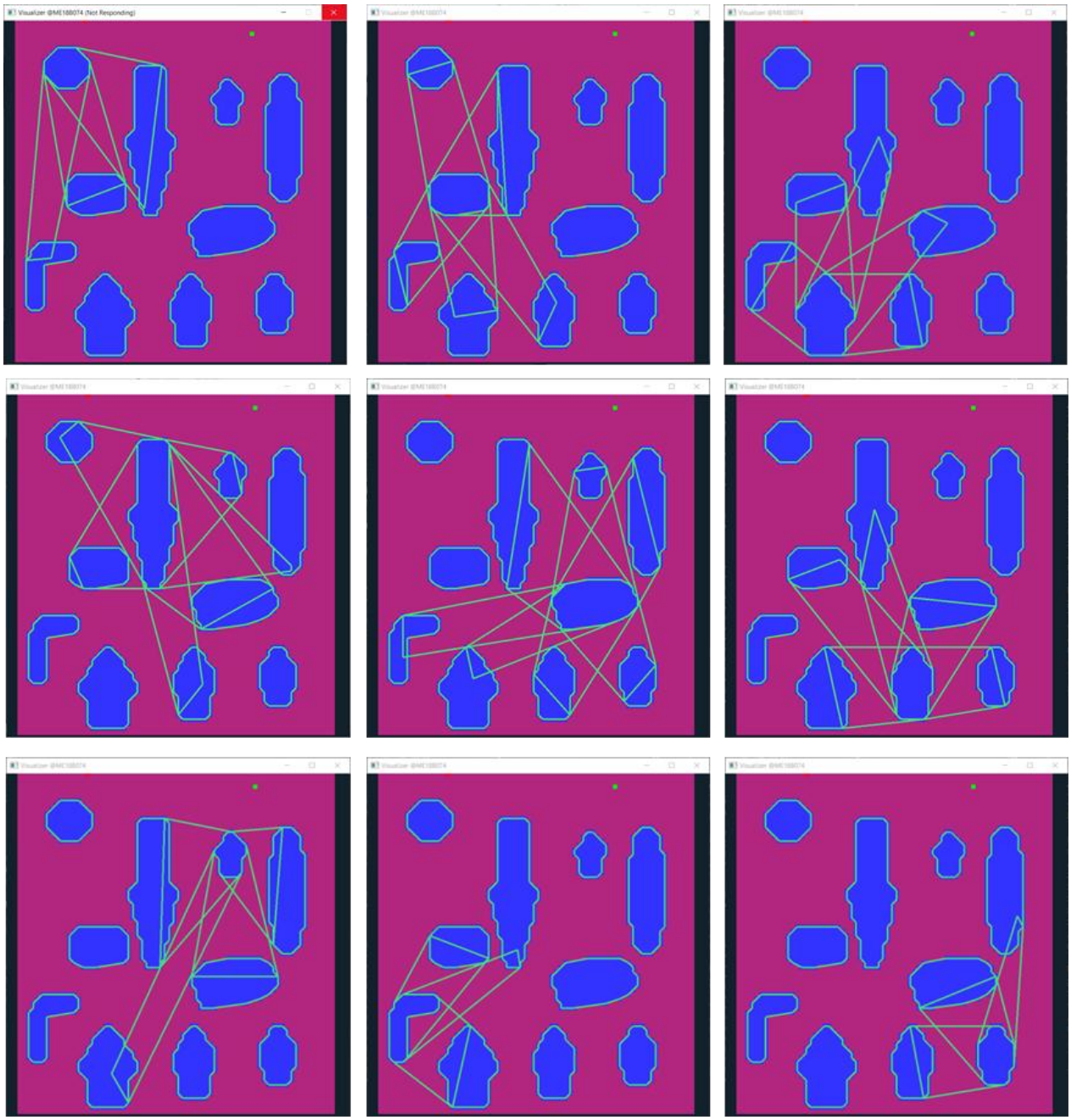


Figure 12 :Outputs from Step II

Quad Data-structure:

- For each pair of nearestneighbours, a quadrilateral region is defined by joining the chords formed by external tangents between these polygons.
- This quad datastructure is special in the fact that; one pair of opposite sides represents obstacles and the 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 the path inside this quad is only defined by the geometry of the nearestneighbours.

5.4 Step III: Graph building using quad datastructure:

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

*** All the steps till here are a one-time computation. And it eases the path search forupcoming steps.

5.5 Step IV: Path search

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

5.6 Elliptical Approximation Method

Step I : Reading data and Sorting polygonal points in clockwise-direction

- Read the image row by row.
- As we get the first occupied cell, a recursive algorithm is used to locate all the boundary points of the obstacle.
- Recursive algorithm:
    - Base case:
        - Start if the current cell is occupied cell. Else continue.
    - Continuation step :
        - For an occupied cell; look for the first neighbouring occupied cell in this order, add that to the boundary point list, and call the recursive function for that cell.

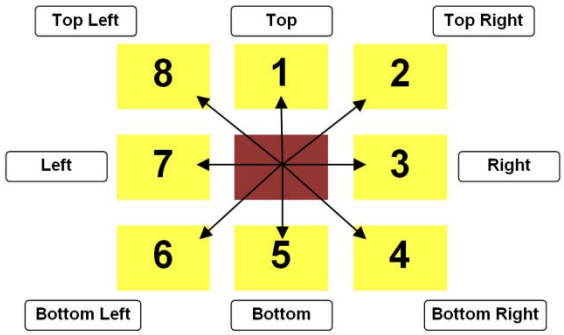


Figure 13 : Traversal order for boundary points generation

    - Termination Step :
        - If all the 8-neighbours are either occupied or all non-occupied cells. Then return.

Step II : Clustering and dividing polygons into convex polygons

- The data is already in pixelated form, so all the straight lines will appear as staggered lines. And it will be a mix of convex and concave vertexes. So the concept of strictly convex polygons won't stand well here.

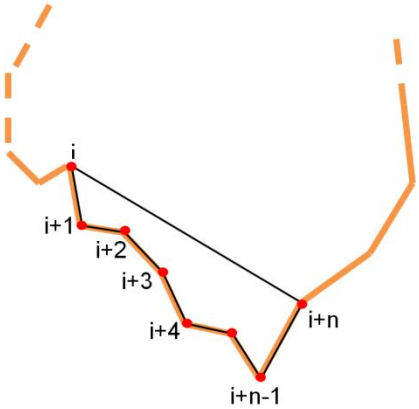


Figure 14 : Ear Detection by area measure

- Consecutive n-points are sampled together, area of the resulting polygon is compared. In clockwise order, the positive area represents a convex turn. The negative area represents a possible concave vertex in-between.
- Another parameter area/$a_{i,i+n}$ is used to filter out concave vertex from pixel-noise.

Step III : Fitting ellipses on the convex polygons

- Various methods of curve fitting or data-specific models like Gaussian mixing models can be used to fit ellipses over previously clustered convex data points.
- As a simpler alternative, Ax=b minimization can fit n data-points (n>6) in the following general equation.

$$ax^2 + 2hxy + by^2 + 2gx + 2fy + c = 0$$

With the fitted ellipses, continue with step II as discussed in section 5.3.

Advantages of this pre-processing step:

- Complex polygons, represented by at least 50-70 data-points are now defined using five parameters of ellipses, resulting in a memory efficient algorithm.
- In the tangent computation step, $\mathbf{O(m^2n^2)}$ (m, n are the order of 50-70) is replaced by a **constant** time operation. A similar computational advantage will be seen in the path-finding stage as well.

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) The 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 15 : Algorithm Summary

CHAPTER 6

RESULTS AND DISCUSSION

6.1 Results

- A bare minimum and functional algorithm is ready which considers obstacles as polygons (no approximations) and not ellipses.
- The current version of the coded algorithm is not optimized to its full potential, and thus it doesn't guarantee significant accuracy.
- But it acts as a proof of concept and validates the proper functioning of nearest-neighbours and quad data-structure part of the algorithm.
- The quad-ranking strategy needs to be thought of to eliminate unnecessary back-and-forth turns in the planned paths.
- There are certain quads, which get covered up by 2-3 other quads as a group. This redundancy needs to be resolved to further speed up traversals.

6.2 Venues of Improvement and Further Research

- Approximating convex polygons using multiple-intersecting ellipses.
- Improving the method of ranking quads to achieve better, shorter paths in even lesser time.
- Extending the algorithm to 3D environments.

6.3 Project Timeline

(Work since Jan 2023)

- Elliptical approximation – under improvement and optimization stage
- Division and clustering of polygons into convex polygons – done
- Integration of elliptical module into previous algorithm – ongoing

Outputs from some failed and successful attempts (polygonal version) :

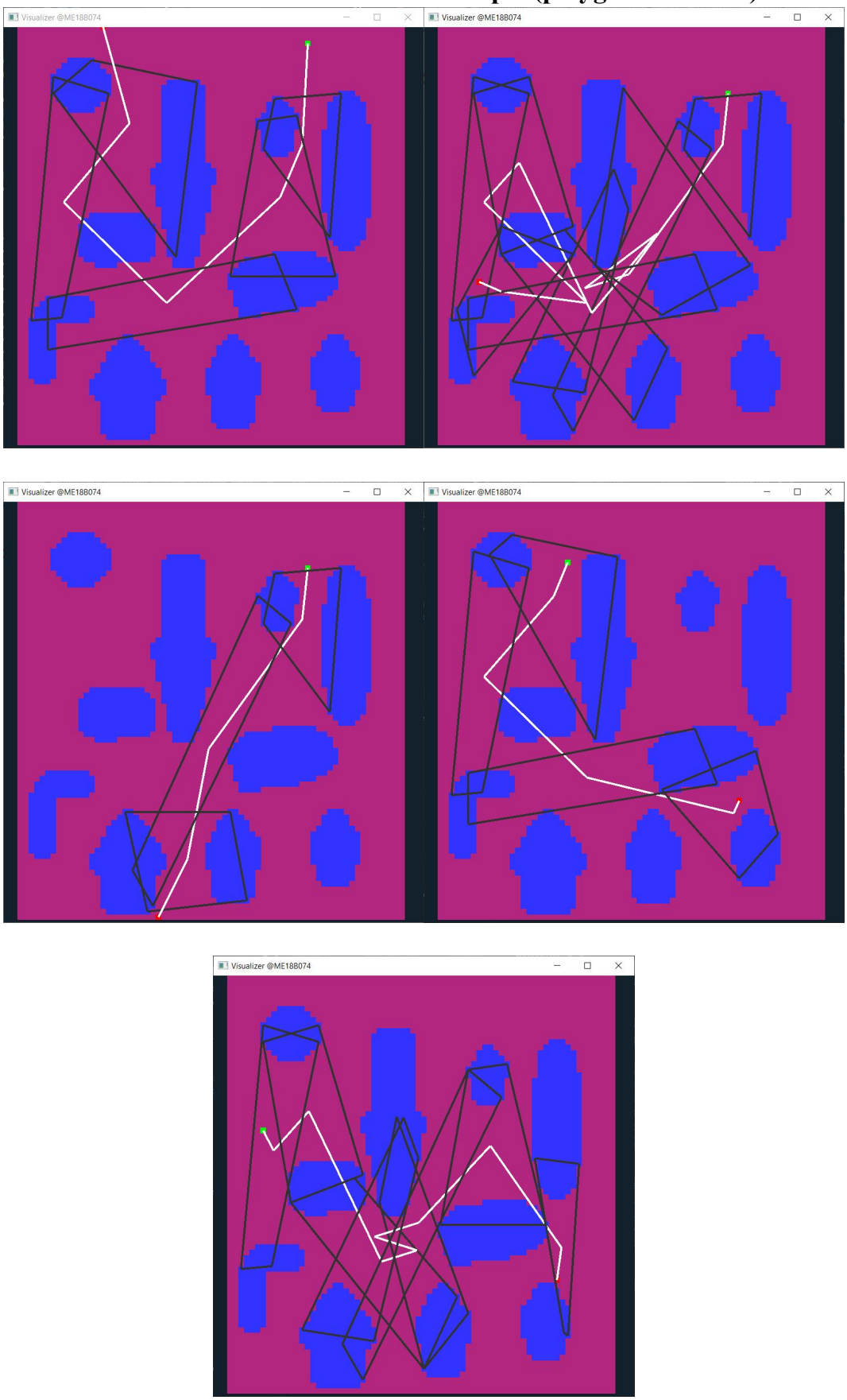
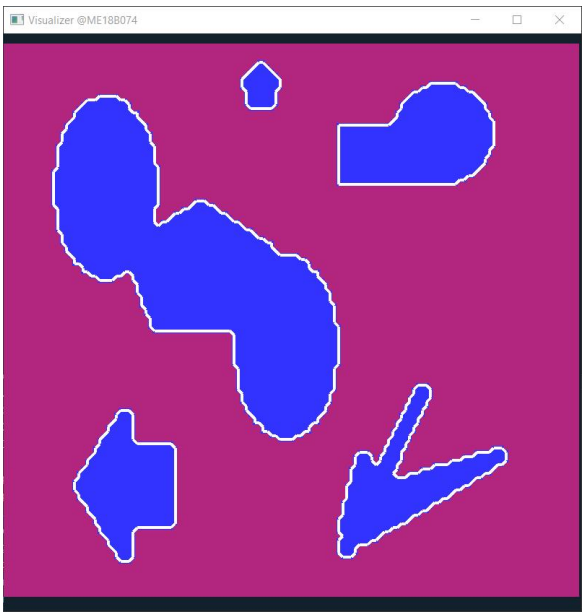
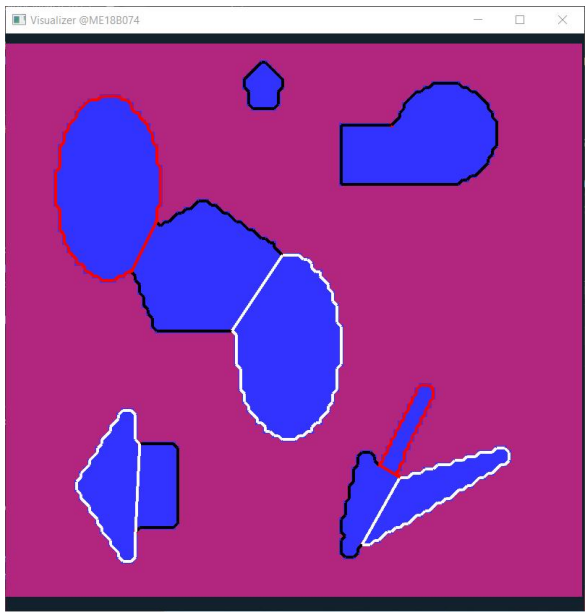


Figure 16 : Results from the polygonal version of the algorithm

Current progress in the elliptical approximation module:



Boundary point detection and sorting



Convex clustering

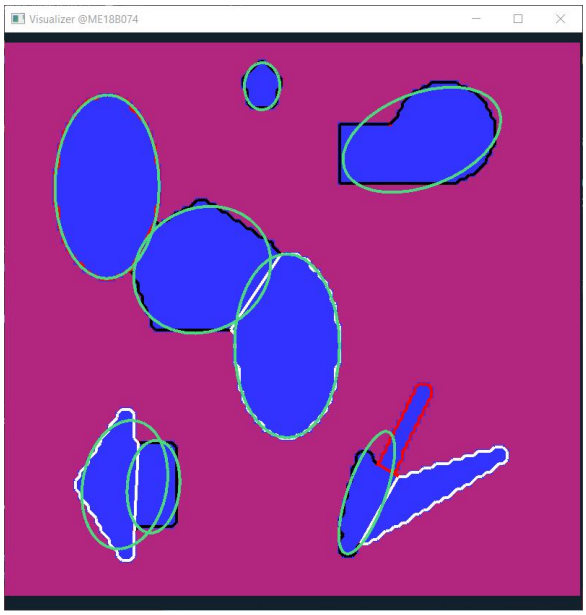


Figure 17 : Elliptical Approximation

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. Magid, Evgeni &Lavrenov, Roman &Afanasyev, Ilya. (2017), **Voronoi-based trajectory optimization for UGV path planning,** 383-387.10.1109/ICMSC.2017.7959506

4. LounisAdouane, Ahmed Benzerrouk, Philippe Martinet, (2016), **Mobile Robot Navigation in Cluttered Environment using Reactive Elliptic Trajectories**, LASMEA, UBP-UMR CNRS 6602, France

5. Omar Souissi, RabieBenatitallah, David Duvivier, AbedlHakimArtiba, Nicolas Belanger, Pierre Feyzeau, (2013), **Path planning: A 2013 survey**, IEEE Conference, Agdal, Morocco, 28-30 October 2013

6. 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.