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

May 2023

THESIS CERTIFICATE

This is to undertake that the Project Report titled **PATH PLANNING ALGORITHM FOR CLUTTERED ENVIRONMENTS USING COMPUTATIONAL GEOMETRY APPROACHES**, submitted by me to the Indian Institute of Technology Madras, for the award of **Master of Technology**, is a bona fide record of the research work done by me under the supervision of **Prof. M. Ramanathan**. The contents of this Project Report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Chennai 600036 **Shreyash Patidar**

Date: May 2023

Prof. M. Ramanathan
Professor
Department of Engineering Design
IIT Madras

ACKNOWLEDGEMENTS

I am incredibly grateful to my project guide, **Prof. M. Ramanathan** for suggesting this research topic, motivation and continuous guidance throughout the work. I am grateful to Advanced Geometric Computing Lab for providing the computational resources to complete my work. I am also indebted to my friends and family for holding me up in the most challenging times and motivating me to work harder.

Shreyash Patidar

ME18B074

ABSTRACT

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

This work proposes a new algorithm for robotic path planning that uses computational geometry approaches to optimize the decision-making process of a robotic agent. Geometry-based algorithms usually result in high computational complexity. This work is an effort to crack a balance between approximations and complexity. The algorithm approximates obstacles as ellipses to reduce computational complexity and improve speed. It runs nearest-neighbors tests to obtain Voronoi diagram-like results and runs path search traversals to find optimal paths. The focus on geometry makes this algorithm suitable for cluttered environments or places where obstacle geometry plays a crucial role; the algorithm will find its use in cluttered environments, or places where obstacle geometry is of the order of size of the robotic agent and obstacle geometry plays a crucial role. The algorithm will find its use in tight environments, such as navigating busy streets or moving packages in small-scale, unorganized warehouses. The proposed algorithm can be extended to 3D environments, making it useful in planning missions for underwater caves, low-height search operations for drones, simulating machine movements for rescue operations, and operations in tight spaces such as mines.

CONTENTS

LIST OF TABLES

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 data structure 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.

The obstacles are estimated as a set of varying number of overlapping ellipses based on the order of accuracy. This will enable quick path hunts or coarse path planning followed by more and more fine path planning as the agent enters tight spaces.

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. Also, the output result will be in a format that is easier to process for kinodynamic planning.

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.
- Graph search and post-processing of planned path.
- Extending the algorithm to make it compatible with 3D environments.

CHAPTER 3

LITERATURE SURVEY AND RELATED WORK

Path planning (Mandalika (2021)) 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. Omar Souissi (2013) categorizes these algorithms as:

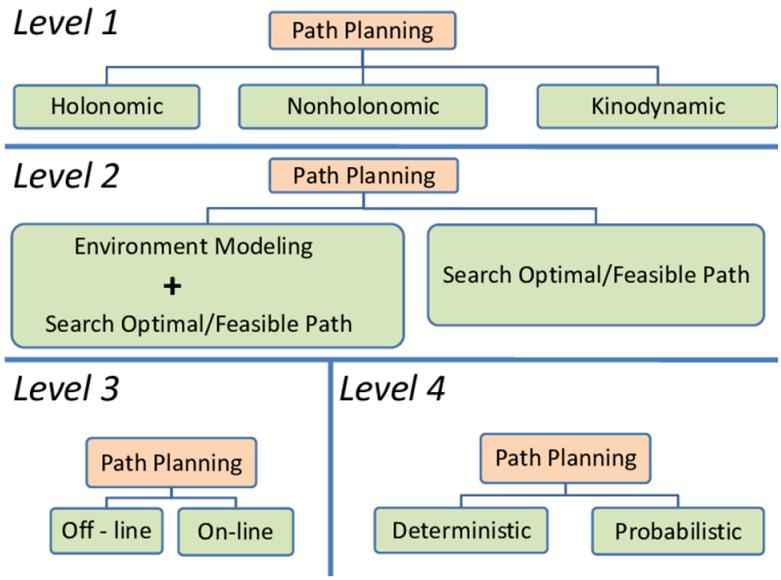


Figure 3.1: Various classifications of Path Planning Algorithm

Based on above categorization, this project work involves environment modelling as well apart from usual path search and optimization. It can be deployed for both on-line and off-line works. The solutions generated are deterministic in nature and convenient to be used for kinodynamic planning(Donald (1993)).

Further, 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 VISISBILITY 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 (Omar Souissi (2013), Inkulu (2022)).

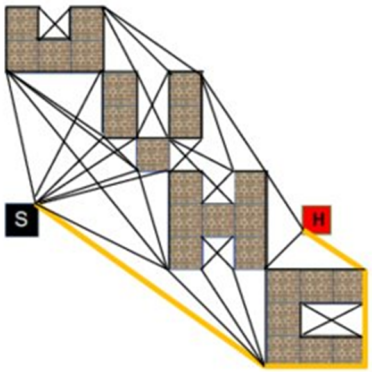


Figure 3.2: Visibility Graph

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 (Magid (2017)) graph search algorithms.

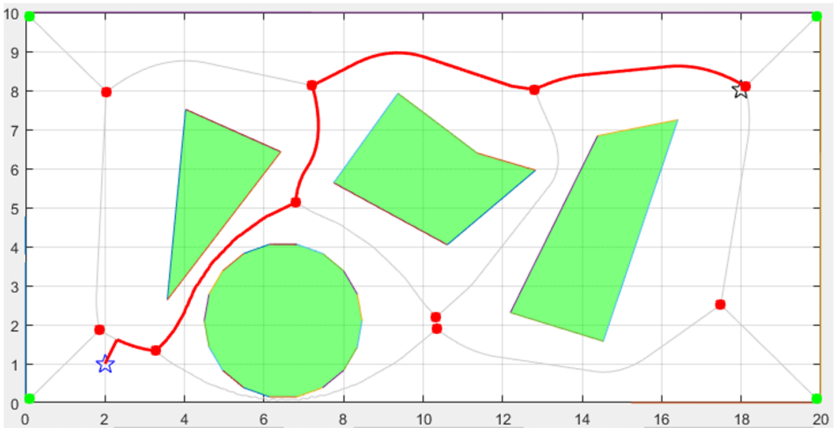


Figure 3.3: Voronoi Graph

3.3 ELLIPSE APPROXIMATION

The idea of representing obstacles as ellipses has already been thought off and worked upon. LounisAdouane (2016) in their paper on **Mobile Robot Navigation in Cluttered Environment using Reactive Elliptic Trajectories** have approximated walls and other obstacles as ellipses.

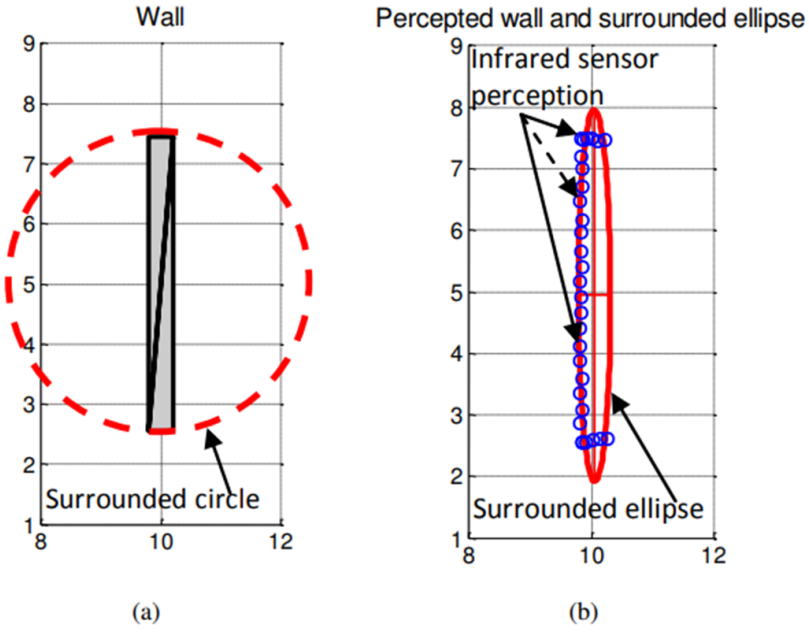


Figure 3.4: Interpolated wall using a circle and an ellipse shape

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. The idea is to develop a method to represent an object by a set of varying number of overlapping ellipses based on the order of accuracy. This type of estimation requires the division of existing polygonal obstacles into several convex polygons. Later, ellipses are fit over these convex clustered points.

Ellipse fitting is finding the best-fit ellipse to a set of points in a 2D plane. One common method for ellipse fitting is the least-squares approach, which involves minimizing the sum of squared distances between the data points and the ellipse. This can be done using matrix algebra and eigenvalue decomposition to solve for the parameters of the ellipse. Other methods include the Hough Transform method, Levenberg-Marquardt algorithm, and the Powell algorithm. Some standard techniques, their implementation, and their limitations have been discussed by Milos Stojmenovic (2007), in their paper **Direct Ellipse Fitting and Measuring Based on Shape Boundaries**.

$$AX^2 + BXY + CY^2 + DX + EY + F = 0$$

$$a, b = \frac{-\sqrt{2\left(AE^2 + CD^2 - BDE + (B^2 - 4AC)F\right)\left((A + C) \pm \sqrt{(A - C)^2 + B^2}\right)}}{B^2 - 4AC}$$

$$x_\circ = \frac{2CD - BE}{B^2 - 4AC}$$

$$y_\circ = \frac{2AE - BD}{B^2 - 4AC}$$

$$\theta = \begin{cases} \operatorname{arccot} \dfrac{C - A - \sqrt{(A - C)^2 + B^2}}{B} & \text{for } B \neq 0 \\ 0 & \text{for } B = 0,\ A < C \\ 90^\circ & \text{for } B = 0,\ A > C \end{cases}$$

Figure 3.5: Generalized equation of ellipse

The approach followed in this work is to represent ellipses in its generalized form of equation, expressed as a second-order equation in x and y. The parameters of these equations estimated by matrix algebra and ellipse parameters i.e., size(major axis), eccentricity, rotation, and center are extracted from the parameters.

In some cases, this algorithm might not return any result. The same generalized equation represents a parabola, a hyperbola, an ellipse, and a pair of straight lines. And thus, it fits one of these curves based on the skewness and completeness of clustered points.

3.4 CONVEX CLUSTERING

Convex clustering is a technique that divides a polygon into several convex polygons. A convex polygon is one with all convex internal angles(interior angles are less than or equal to 180 degrees). Here the dataset is pixelated, and hence small and alternate convex-concave edges are inevitable, forming small ears.

Ears (O'Rourke (1997)) are small semi-loop-like structures in 2D polygons. In our case, small ears resulting from pixelated lines are to be ignored, whereas bigger ears are considered for clustering.

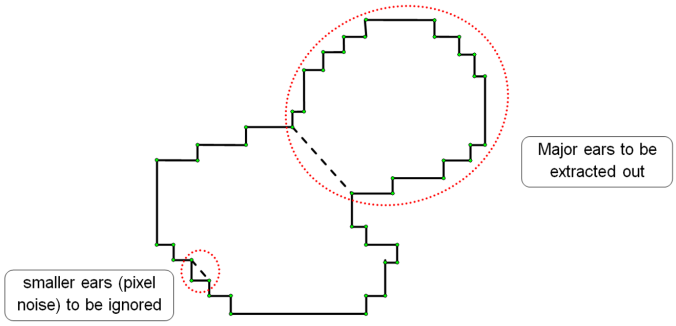


Figure 3.6: Ear clipping for convex clustering

A quick check to distinguish between this pixel noise and major ears is the area measure; any ear with the area under a set threshold is considered as noise. Another measure is the ear area ratio to the support chord's length (shown in the dashed line). Large

values of this ratio mean more bulkiness of the blob(ear) compared to the supporting chord. And hence more detached is that shape from the geometry. Such chords are an excellent parting line to divide into convex shapes, and the same has also been used in the algorithm.

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

For a **chosen polygon (polygon 1 in Fig 4.1)**, any other polygon (polygon 2 in Fig 4.1) 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.

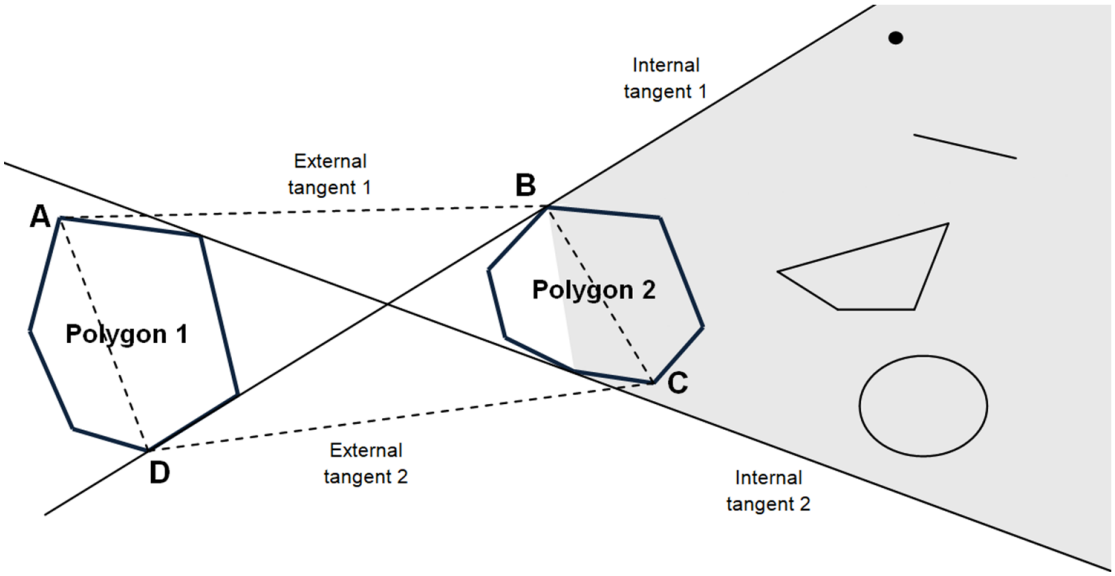


Figure 4.1: Polygon Shielding

Proof-

Consider a degenerate case where polygon 2 is deflated into a line segment along line PQ. Line PQ is same as the chord BC' formed by the two internal tangents at polygon 2.

So, the polygon is deflated to take the shape of this chord BC', keeping all other geometries and tangents same.

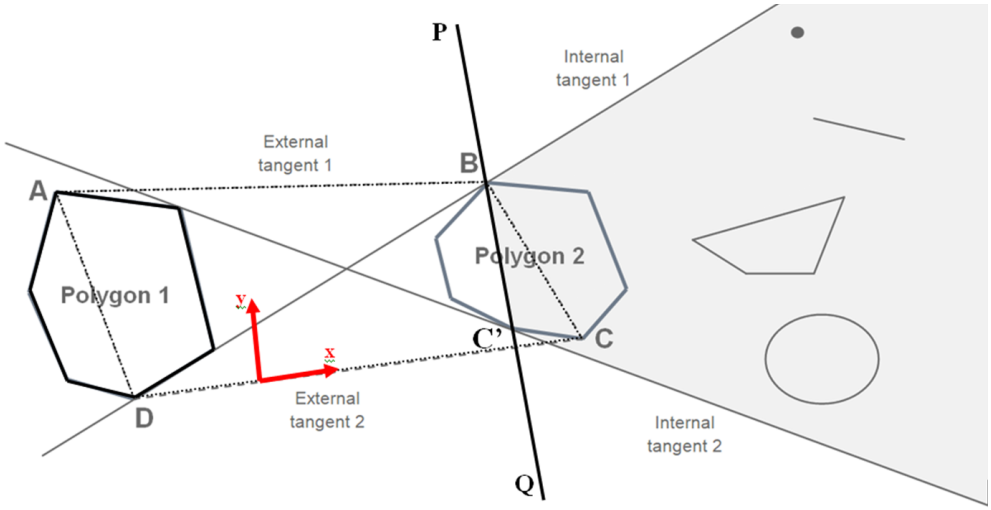


Figure 4.2: Polygon Shielding

Now, if we could prove that BC' is enough to shield the area highlighted in grey to have any influence on Voronoi edges inside region ABCD, we can always say that polygon 2 will also shield all the geometric entities inside the grey- region to have any Voronoi edge with polygon 1 inside region ABCD.

Setting the coordinate system along External tangent 2 (CD); for any point inside region ABCD, chord BC' is closer than any other point in the grey zone. Hence, any point inside the grey zone can never have any part of its Voronoi region inside the quad. ABCD. And therefore, no Voronoi edge is shared with polygon 1 inside region ABCD.

4.2 QUAD DATA STUCTURE

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).

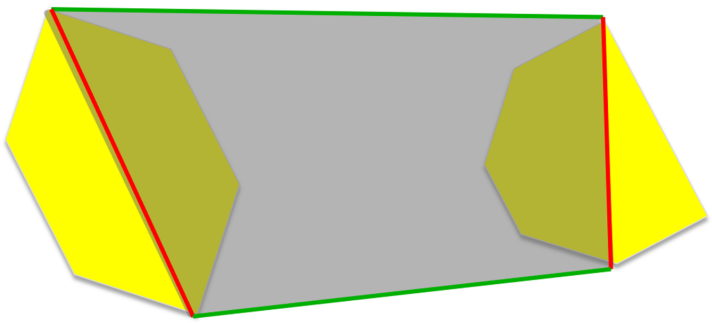


Figure 4.3: Quad Representation

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.

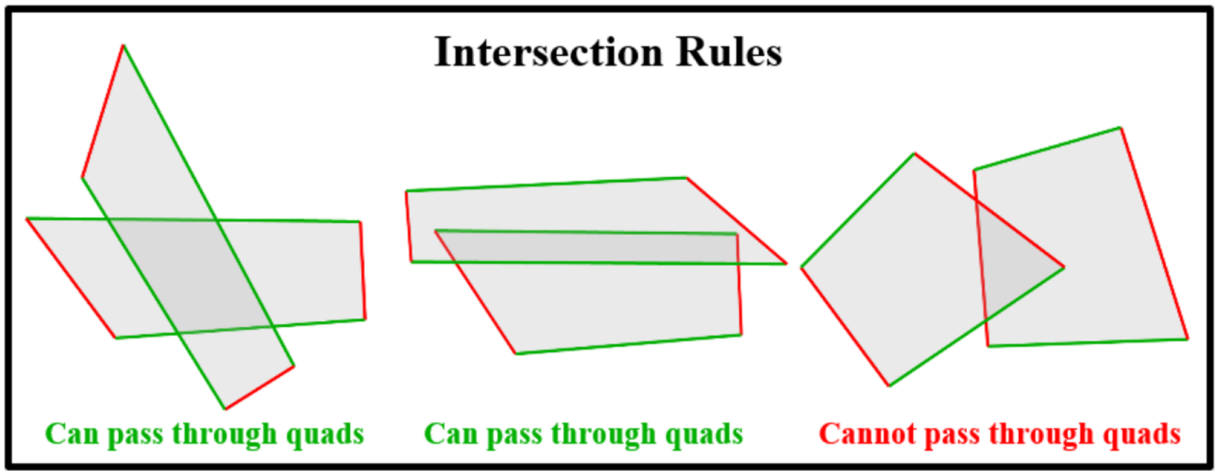


Figure 4.4: Intersection Rules – Quad Structures

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)

4.3 PRELIMINARIES

4.3.1 Left-On Predicate

A simple, constant-time operation to check whether a point lies on the left of a given line segment (O'Rourke (1997)).

For a point (x_0, y_0) and line segment (x_1, y_1) ; (x_2, y_2), leftOn predicate is given by the mathematical expression :

$$LP = x_1 * (y_2 - y_0) + x_2 * (y_0 - y_1) + x_0 * (y_1 - y_2)$$

4.3.2 Convex Hull

Convex hull is the smallest convex shape encompassing a set of given points. In 2D, it is the smallest convex polygon surrounding the set of given 2D points. Similarly, in 3D, it is a convex polyhedron with a set of points falling strictly inside, where a convex polyhedron is a 3D polyhedron with all its adjacent faces making an internal angle of less than 180 degrees.

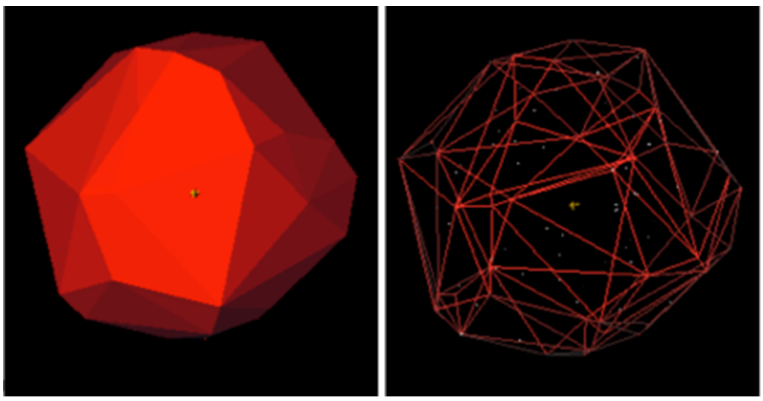


Figure 4.5: Convex Hull

4.3.3 Monotone Mountains

A monotone mountain is a monotone polygon (O'Rourke (1997)) with one of its two monotone chains as a single segment, the base. And along this base, each vertice on the other chain is in sorted order along the base axis. The other chain never turns back along the base axis.

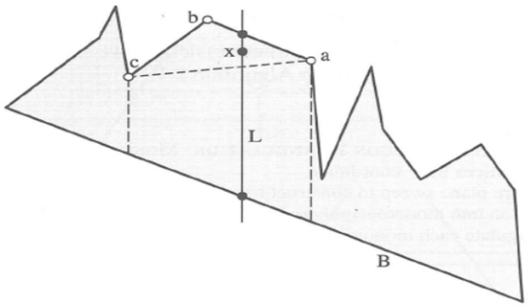


Figure 4.6: Monotone Mountain

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 ALGORITHM

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

- Read the 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

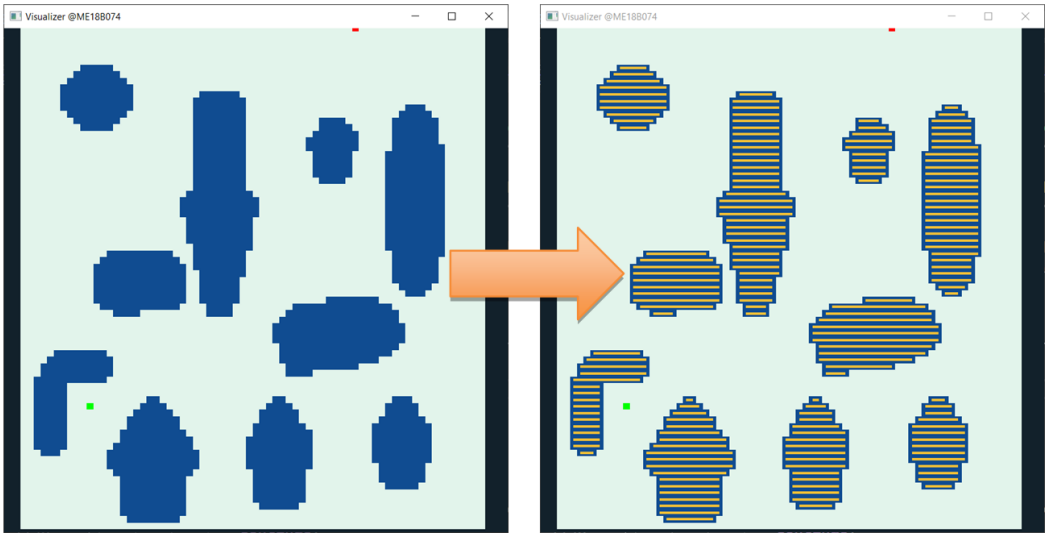


Figure 5.1: Locating obstacles from the binary image

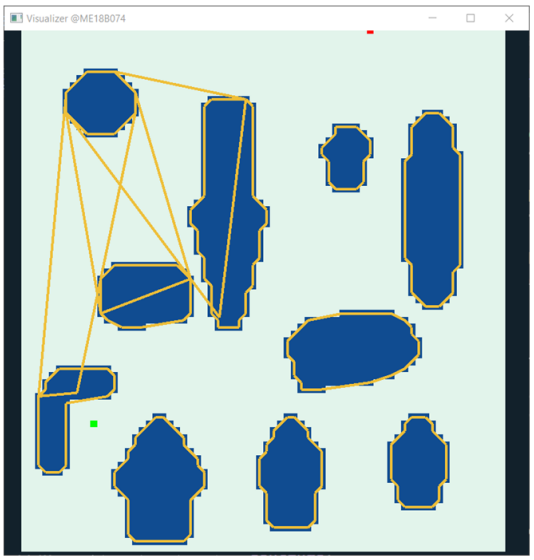


Figure 5.2: Locating obstacles from the binary image

5.2.2 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.

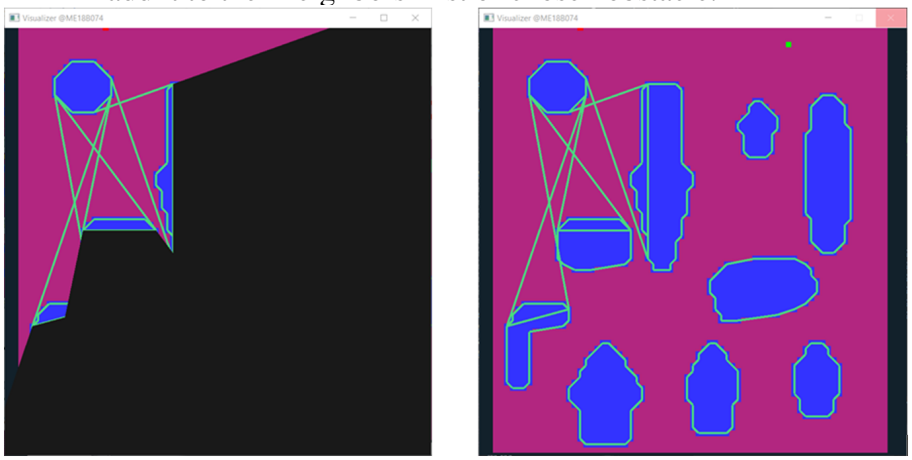


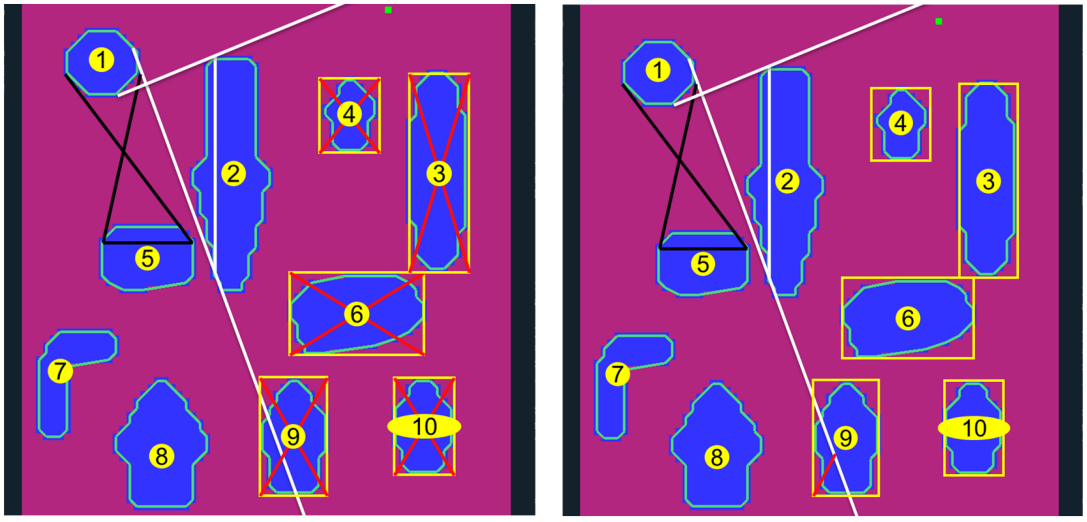
Figure 5.3: Obstacles falling in the black region don’t contribute to Voronoi edge with the first obstacle

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).



Complete Shielding Partial Shielding

Figure 5.4: Shielding Check

Outputs at the end of step 2

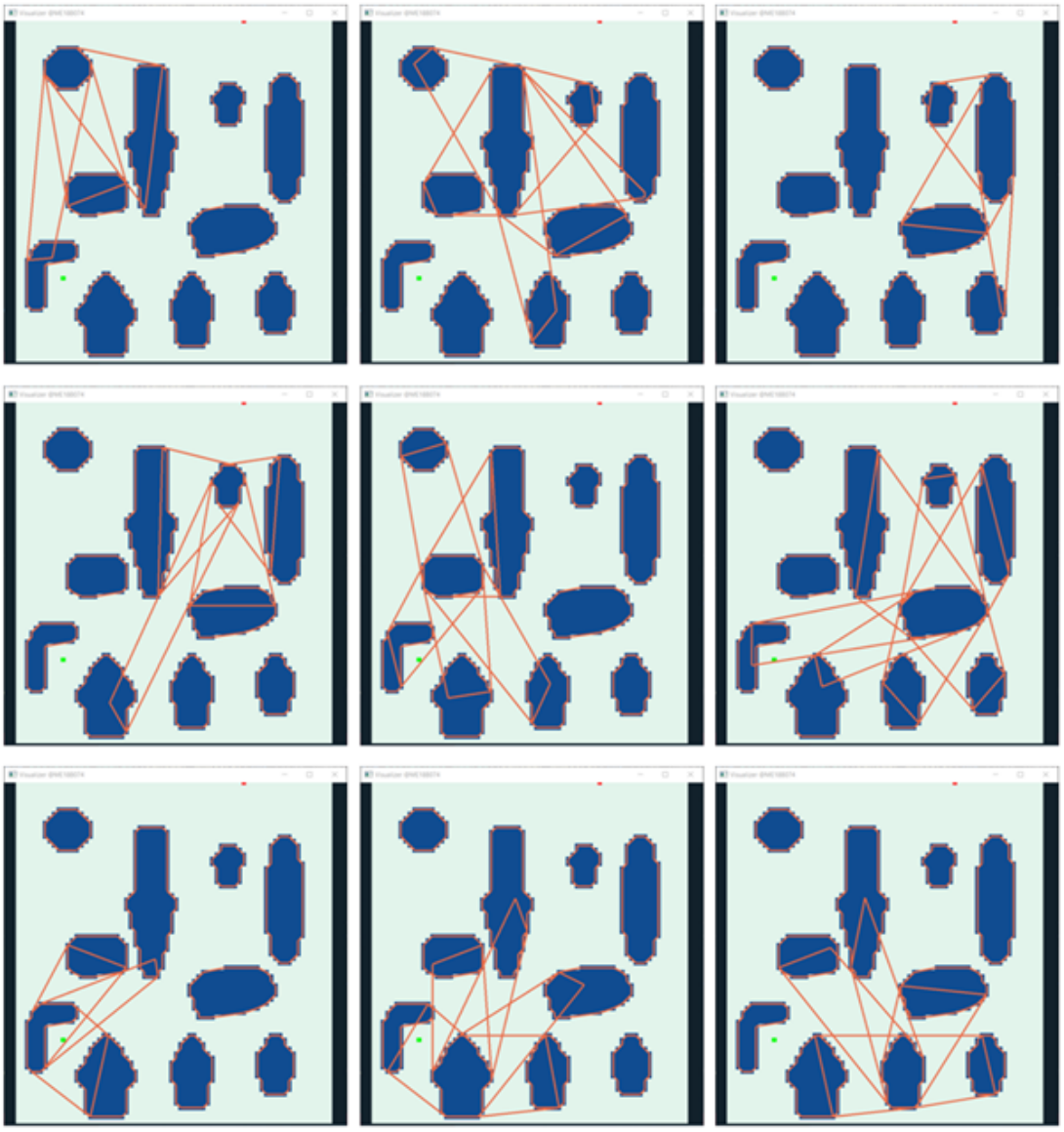


Figure 5.5: Outputs from Step II

Quad Data Structure

- For each pair of nearest neighbours, a quadrilateral region is defined by joining the chords formed by external tangents between these polygons.
- This quad data structure 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 nearest neighbours.

5.2.3 Step III : Graph Building using quad data structure

- 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 for upcoming steps.

5.2.4 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.

- Some of the node-specific heuristic parameters for these quads include the free area estimate of a quad, estimated free length, and estimated path width.

- Node-node movement cost is determined in terms of certain parameters like the area of the common polygon between two quads, estimated length of the common chord (acts as width of entry or exit from one quad to other).

- The final planned path is returned as an ordered list of quads to be traversed through in order to reach the goal.

5.2.5 Step V : Path Post Processing

- The returned path is iterated for each quad; if the start-point lies inside any of the quads, all the quads before it are deleted from the list.

- Similarly, if the goal-point lies inside any of the quads, all the subsequent nodes (quads) are removed from the list.

- If any of the quad has good enough overlap with a quad that is further down in the list, all the nodes in between these two nodes are deleted.

- 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 the number of turns, straightness, width, and length of the path; the best path is sent as output.

5.3 ELLIPTICAL APPROXIMATION METHOD

5.3.1 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

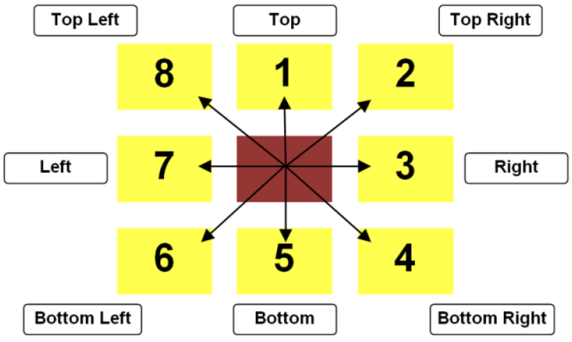


Figure 5.6: Traversal order for boundary points generation

  – Termination step:

    * If all the 8-neighbours are either occupied or all non-occupied cells. Then return.

5.3.2 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.

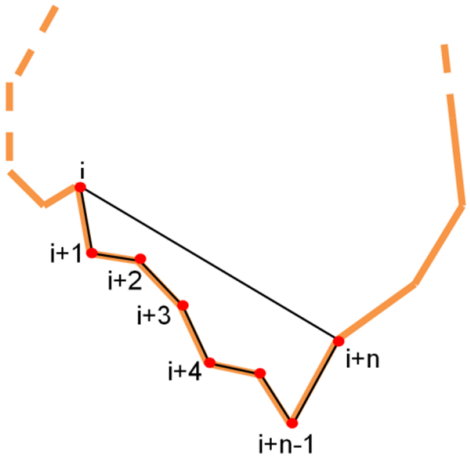


Figure 5.7: Ear Detection by area measure

- 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/ai,i+n is used to filter out concave vertex from pixel-noise.

5.3.3 Fitting ellipses on 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. ax2 + 2hxy + by2 + 2gx + 2fy + c = 0

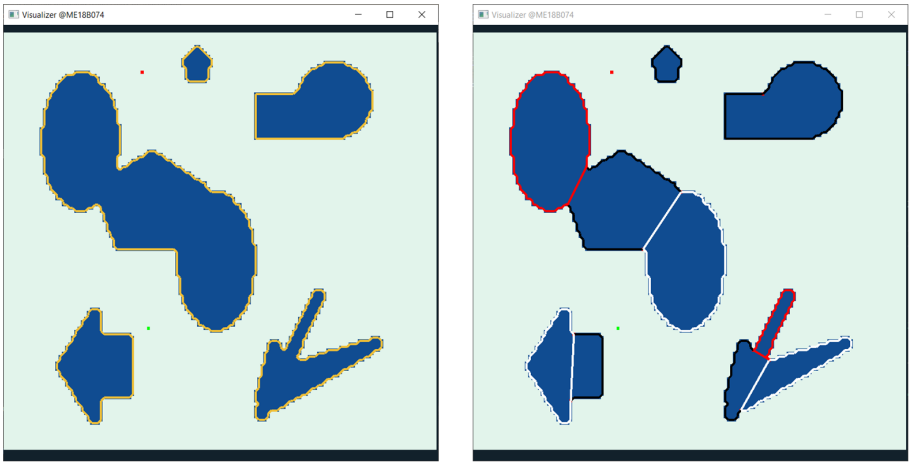
With the fitted ellipses, continue with step II as discussed in section 5.2.2.

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.

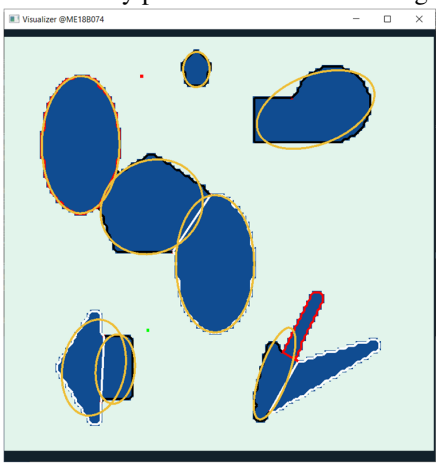
- In the tangent computation step, **O(m2n2)** (m, n are the order of 50-70) is replaced by a **constant** time operation. A similar computational advantage will be seen in the path-finding stage as well.

Algorithm Summarized

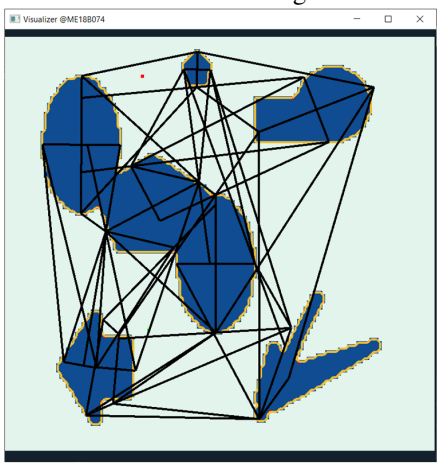


Boundary point detection and sorting

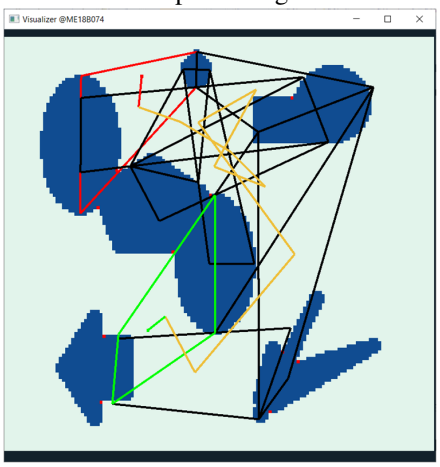
Convex clustering



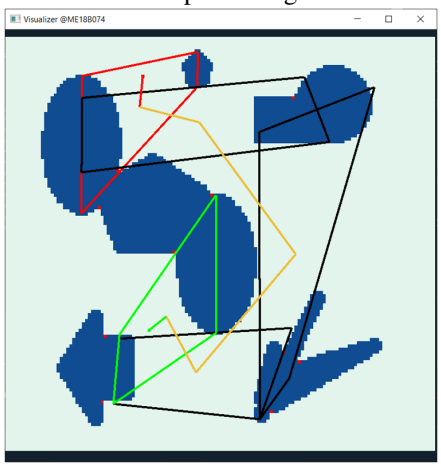
Ellipse Fitting



Map Building



Path Planning



Post Processing

Figure 5.8: Elliptical Approximation

CHAPTER 6

RESULTS AND DISCUSSION

6.1 RESULTS

Performance comparison with existing algorithms:

7 different algorithms from Open Motion Planning Library(OMPL) are compared against this algorithm

- RRT* (Rapidly Exploring Random Trees) (ompl/geometric/planners/rrt/RRTstar.h)
- PRM (Probabilistic RoadMap) (ompl/geometric/planners/prm/PRM.h)
- FMT (Fast Marching Tree) (ompl/geometric/planners/fmt/FMT.h)
- EST (Expansive Space Trees) (ompl/geometric/planners/est/EST.h)
- RLRT (ompl/geometric/planners/rlrt/RLRT.h)
- SST (Sparse Sampling Trees) (ompl/geometric/planners/sst/SST.h)
- STRIDE (Stride based algorithms) (ompl/geometric/planners/stride/STRIDE.h)

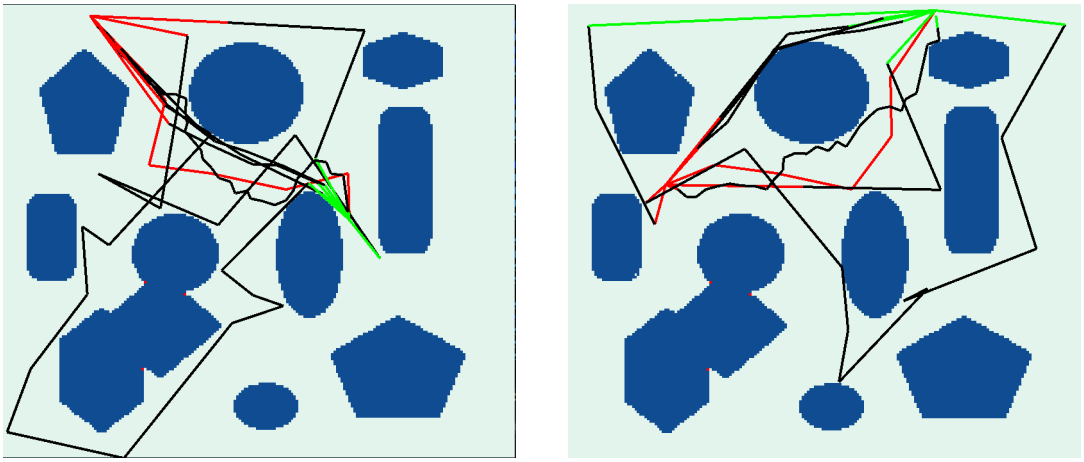


Figure 6.1: Planned paths generated by various algorithms

Planning time comparison:

Frame size	Iterations	This	RRT*	PRM	FMT	EST	RLRT	SST	STRIDE
70x76	172	**0.625**	503.9	511.2	285.6	6.738	45.42	502.8	28.2
138x132	127	**0.502**	504.2	511.7	260.5	5.56	14.05	502.5	41.6
207x182	272	**0.718**	499.3	507.5	271.3	7.01	20.87	498.7	25.2
448x433	136	**1.672**	503.7	514.3	198.5	7.51	9.46	502.7	50.7

Table 6.1: Planning time comparison

** all values are in milliseconds

** these values are averaged out after performing indicated number of iterations with different start and goal states

Path-length comparison :

Frame size	Iterations	This	RRT*	PRM	FMT	EST	RLRT	SST	STRIDE
70x76	172	1.00	0.66	0.66	0.65	1.12	1.70	1.03	1.63
138x132	127	1.00	0.55	0.56	0.55	0.88	1.14	0.68	1.31
207x182	272	1.00	0.64	0.65	0.64	0.95	1.27	0.91	1.42
448x433	136	1.00	0.49	0.50	0.48	0.79	1.10	0.65	1.11

Table 6.2: Planned path-length comparison

** all values are relative output path size wrt. to our algorithm. ie. the ratio of length of path for a certain algorithm to the length of path planned by our algorithm.

** values larger than 1 means our algorithm generates shorter path than that algorithm and vice versa.

** these values are averaged out after performing indicated number of iterations with different start and goal states

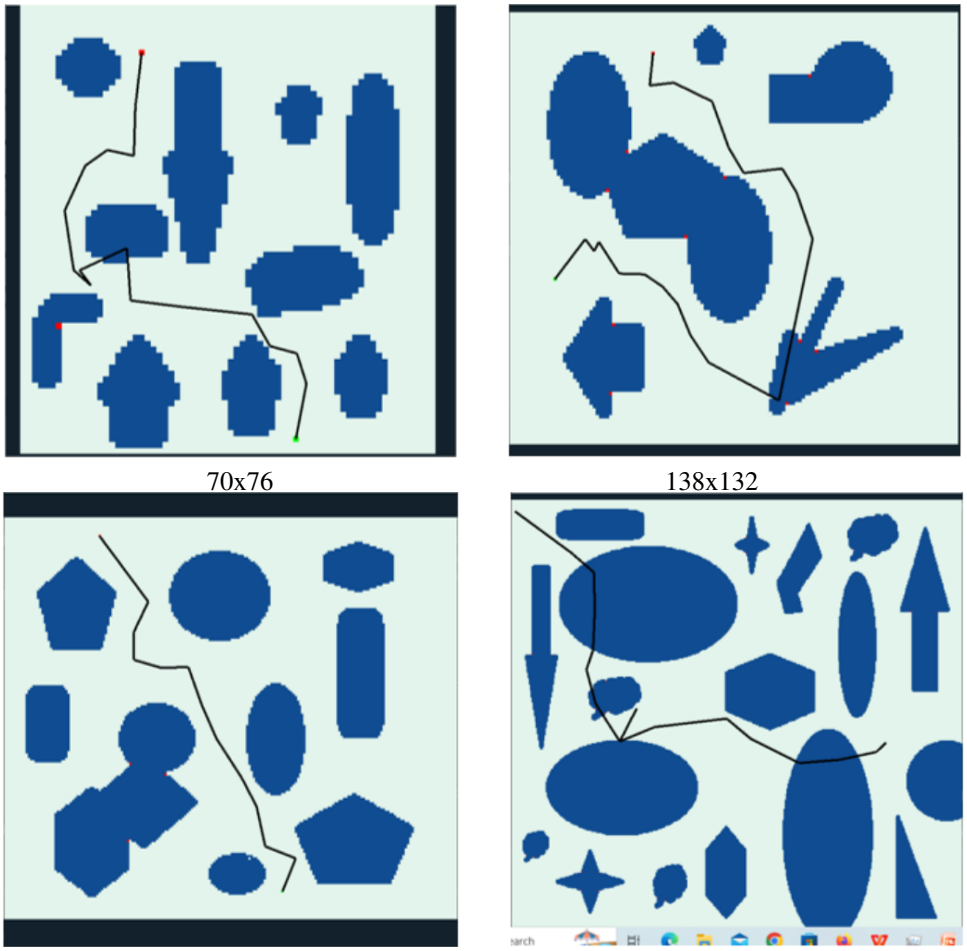


Figure 6.2: Algorithm results on various obstacle maps and sizes

Algorithms performance with real-time sensors :

Frame Size	Number of obstacles	Iterations	Configuration time (ms)	Average time per path search (µs)	Input @30fps	Input @60fps	Input @120fps	Input @240fps
70x76	10	127	26.6	502.4	29.01	59.79	118.30	227.06
138x132	7	172	34.5	624.5	29.44	59.89	119.14	233.32
207x182	11	272	66.5	718.3	28.42	59.68	117.47	221.16
448x433	18	136	1234.1	1671.5	27.13	59.59	116.65	215.59

Table 6.3: Expected output frame rate for given input streams

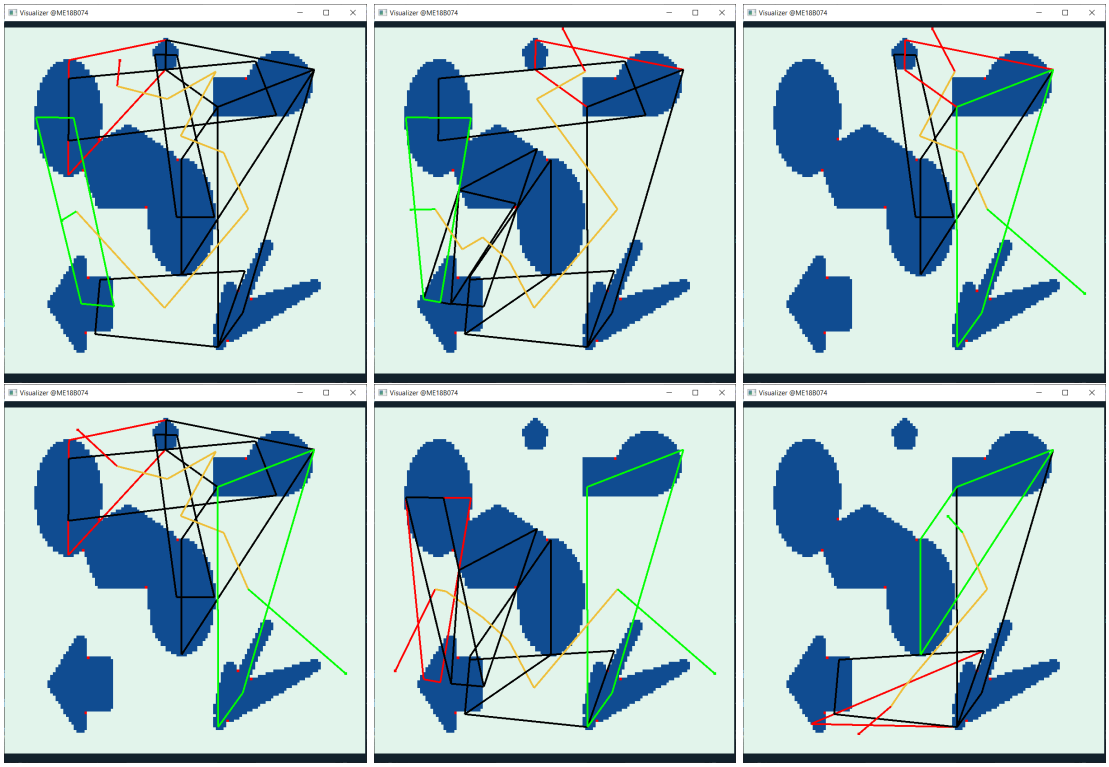


Figure 6.3: Paths planned by the algorithm

** Here yellow line just represents the order of quad units of planned path and not the path itself.

6.2 INFERENCES

- After testing on 4 different maps and 700+ iterations, this algorithm came out to be nearly 10x faster than other algorithms available open-source (OMPL).
- Other algorithms with planning time of the order 500ms keeps on improving the solution with time; but they give out first feasible path within (15-20ms).
- Our algorithm generates paths, smaller than the other fast-processing algorithms (RLRT, STRIDE and EST).
- At this processing speed, the algorithm can process frames from high speed sensors (240 fps) without losing much data/frames.
- For this speed, some accuracy has been compromised, there is need to improve accuracy, many times planned paths breaches inside the obstacle.

- This can be improved by optimizing ellipse fitting and local-path planning step. And the algorithm can still be 2x-3x faster than other algorithms.

- The increase in speed is primarily due to new data-structure and representation. Ex. for a 100x100 pixel image, 10,000 (lighter) nodes are being replaced by 20-30 (heavy on memory) quad structures.

6.3 CONCLUSIONS

- A very fast real-time path-planning algorithm is ready and functional, which returns a feasible path.

- The elliptical approximation is a bottleneck in this algorithm, and the correctness of the planned path is highly dependent on the ellipse fitting operations.

- ”Exact tangent computation of ellipses is possible in near-constant time operations” was a claim made earlier during the project. But contrary to expectations, the computation is bit heavy. Hence, approximated tangents are computed to achieve similar performances.

- Introduction of a new data representation – Quad structure was found feasible, and it makes the solution highly modular for subsequent kinodynamic planning and environment specific optimization.

- Graph search and path post-processing operations are to be optimized for the shortest / quickest paths.

6.4 KEY FINDINGS

- Tangent-based shielding test to rule out far-off polygons from computations.

- Quad data structure to effectively represent and process free-to-move spaces.

6.5 VENUES OF IMPROVEMENTS AND FURTHER RESEARCH

- Approximating convex polygons as multiple overlapping ellipses based on order of accuracy.
- Exact computation of common tangents of two ellipses in a near-constant time operation. (Closed-form solution to compute tangents)
- Extending the algorithm to 3D environments.

6.6 SUB-PROBLEMS WORKED UPON

- 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.
- Graph search and post-processing of planned path.

BIBLIOGRAPHY

1. **Donald, P. C. J. R. J., B.; Xavier** (1993). Kinodynamic motion planning. *Journal of the ACM*. URL https://users.cs.duke.edu/~brd/papers/src-papers/jacm93.pdf.

2. **Inkulu, S. A. . R.** (2022). Visibility polygons and visibility graphs among dynamic polygonal obstacles in the plane. *Journal of Combinatorial Optimization volume 44, pages 3056–3082*. URL https://link.springer.com/article/10.1007/s10878-022-00846-1.

3. **LounisAdouane, P. M., Ahmed Benzerrouk** (2016). Mobile robot navigation in cluttered environment using reactive elliptic trajectories. *LASMEA, UBP-UMR CNRS 6602, France*. URL https://www.researchgate.net/publication/269264462_Mobile_Robot_Navigation_in_Cluttered_Environment_Using_Reactive_Elliptic_Trajectories.

4. **Magid, R. A. I., Evgeni Lavrenov** (2017). Voronoi-based trajectory optimization for ugv path planning. *383-387.10.1109/ICMSC.2017.7959506*. URL https://kpfu.ru/staff_files/F_999381834/2017_Voronoi_Based_Trajectory_Optimization_for_UGV_Path_Planning.pdf.

5. **Mandalika, A. V.** (2021). Efficient robot motion planning in cluttered environments. URL https://digital.lib.washington.edu/researchworks/handle/1773/47042.

6. **Milos Stojmenovic, A. N.** (2007). Direct ellipse fitting and measuring based on shape boundaries. *PSIVT 2007, LNCS 4872, pp. 221–235, 2007*. URL https://link.springer.com/content/pdf/10.1007/978-3-540-77129-6_22.pdf.

7. **Omar Souissi, D. D. A. N. B. P. F., RabieBenatitallah** (2013). path planning: A 2013 survey. *IEEE Conference, Agdal, Morocco*. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6761521.

8. **O'Rourke, J.**, *Computational Geometry In C 2nd ed*. 1997.