Robot Operating System 2 Toolbox for Formation and Consensus Using Multiple Robots

Thesis submitted by

Rudresh Singh
2022EEA2007

under the guidance of

Prof. Deepak U. Patil
Indian Institute of Technology Delhi

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

Master of Technology

Department Of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI

May 2024

THESIS CERTIFICATE

This is to certify that the thesis titled **Robot Operating System 2 Toolbox for Formation and Consensus Using Multiple Robots**, submitted by **Rudresh Singh (2022EEA2007)**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under my 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.

Prof. Deepak U. Patil
Associate Professor
Dept. of Electrical Engineering
Indian Institute of Technology Delhi

Place: Hauz Khaz, New Delhi

Date: 30th May 2024

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to my supervisor, **Prof. Deepak U. Patil**, for his continuous guidance and encouragement throughout my MTech project. His mentorship has been invaluable in teaching me how to approach, write about a research problem.

I am immensely grateful to my peers Nitish Bishnoi, Ashwin K M, and Sabyasachi Kundu for their steadfast support and encouragement throughout my academic and research endeavors. Their valuable insights and constructive feedback have played a pivotal role in shaping the concepts explored in this thesis.

Furthermore, I would like to express my heartfelt appreciation to my dear family for their unending support and encouragement during my academic journey. Their unwavering faith in me has been a cornerstone of my accomplishments, and I am deeply indebted to them for their boundless love and encouragement.

Finally, I express my gratitude to all those who have contributed to shaping C++ into the powerful language it is today.

Rudresh Singh

ABSTRACT

This thesis presents a comprehensive study on consensus algorithms and formation control in multi-robot systems, with a focus on achieving consensus in both one-dimensional and two-dimensional spaces in minimum time. Leveraging the ROS 2 architecture and the TurtleBot3 platform for experimentation, efficient algorithms for line and circle formations are developed and validated through rigorous simulations and theoretical proofs. Practical viability and scalability are demonstrated through implementation on the ROS 2 architecture and the TurtleBot3 platform, laying the groundwork for future research in swarm robotics and multi-agent systems. The thesis details algorithms for one-dimensional and two-dimensional consensus in minimum time, along with procedures for line formation and circle formation, each accompanied by theoretical formulations, algorithms, and algorithmic testing.

Contents

List of Figures

Chapter 1

Introduction

This work presents a comprehensive study of consensus algorithms and formation control in multi-robot systems. We focus on achieving consensus in both one-dimensional and two-dimensional spaces in minimum time, leveraging the ROS 2 architecture and the TurtleBot3 platform for experimentation. The primary contributions of this work entail the creation of streamlined algorithms for line and circle formations, substantiated by thorough simulations and theoretical validations. While existing literature discusses Swarm Behavior [1] and formation strategies [2] utilizing robots, few delve into achieving formations in minimum time. Thus, this study focuses on addressing the challenge of achieving formation and consensus in minimum time.

The relevance of this work lies in its potential applications across various domains, such as automated industrial processes, search and rescue missions, drone shows, and space exploration. By optimizing the time required for robots to form specific configurations, our algorithms can significantly enhance the efficiency and responsiveness of robotic swarms in dynamic environments.

Moreover, the implementation of these algorithms on the ROS 2 architecture and the TurtleBot3 platform demonstrates the practical viability and scalability of our approach. The impact of this work can be far-reaching, providing a foundational framework for future research in swarm robotics and multi-agent systems.

In this work, our objective is to align robots both in a circular formation and along a straight line within the minimum time possible. Given the equation $t = \frac{d}{s}$, where t represents time, d denotes distance, and s represents velocity, our focus lies on minimizing the distance of each robot to the computed circle and line as we assume a constant velocity for all robots throughout the operation.

Chapter 2

1 Dimensional Consensus in Minimum Time.

2.0.1 Theory

Consider robots that can move only along a single dimension, and we aim to align them in a straight line in the shortest possible time. In one dimension, the reachability space can be represented as a line segment extending in both directions. The point where the reachability spaces of all the robots first overlap is the point of consensus, achieved in minimum time. Robots are constrained to move only in positive and negative x direction.

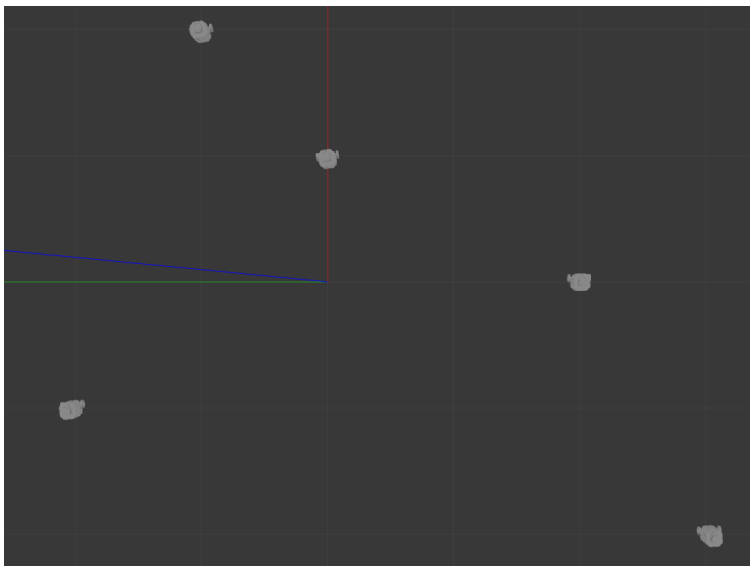


Figure 2.1: Turtlebot3 Spawned in Gazebo Environment

The last two reachability spaces to intersect will belong to the robots that are the farthest apart. Therefore, we will calculate the average of the x-coordinates of these two robots and aim to align all other robots to this x-coordinate. This method ensures achieving 1-D consensus in the minimum possible time. We can find this solution in $O(n)$, where n is the number of robots.

2.0.2 Algorithm

Algorithm

```
1: procedure FIND1DCONSENSUS(robot_coords)
2:     n = length of robot_coords
3:     for i = 1 to n do
4:         if robot_coords[i] < min_x then
5:             min_x = robot_coords[i]
6:         end if
7:         if robot_coords[i] > max_x then
8:             max_x = robot_coords[i]
9:         end if
10:    end for
11:    target_x = (min_x + max_x)/2
12:    return target_x
13: end procedure
```

Chapter 3

2 Dimensional Consensus in Minimum Time.

3.0.1 Theory

The smallest-circle problem is a mathematical problem of computing the smallest circle that contains all of a given set of points in the Euclidean plane. The corresponding problem in n-dimensional space, the smallest bounding sphere problem, is to compute the smallest n-sphere that contains all of a given set of points.

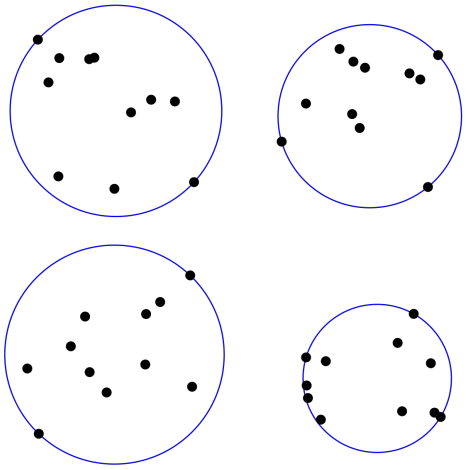


Figure 3.1: Minimum enclosing circle

3.0.2 Procedure

a) Select any 2 points and make them the diameter of the circle.

b) Check if all other points, other than these 2 points, lie in that circle or not.
   i) If it does, we store the value of radius and center (mid-point).
   ii) If it does not, we select the next set of 2 points.

c) If we are not able to find such a point then go to d) else exit.

d) Now select a set of 3 points and make a circle passing through all of them.

e) Check if all other points, other than these 3 points, lie in that circle or not.

f) Store the radius of the circle if no point lies outside that circle.

g) The answer would be the circle with the minimum radius.

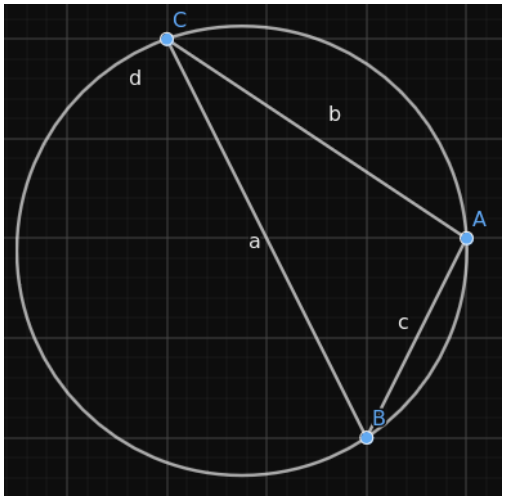


Figure 3.2: Minimum enclosing circle

Suppose the center of the given circle is (x, y) then, where (x_1, y_1), (x_2, y_2), and (x_3, y_3) are the coordinates of the three points, and a, b, and c are the lengths of the sides of the triangle opposite the three points, respectively. Then :

$$x = \frac{a^2(b^2+c^2-a^2)x_1 + b^2(a^2+c^2-b^2)x_2 + c^2(a^2+b^2-c^2)x_3}{2\left(a^2(b^2+c^2-a^2) + b^2(a^2+c^2-b^2) + c^2(a^2+b^2-c^2)\right)}$$
$$y = \frac{a^2(b^2+c^2-a^2)y_1 + b^2(a^2+c^2-b^2)y_2 + c^2(a^2+b^2-c^2)y_3}{2\left(a^2(b^2+c^2-a^2) + b^2(a^2+c^2-b^2) + c^2(a^2+b^2-c^2)\right)}$$

Once we have the coordinates of the circumcenter, we can use the distance formula to find the radius of the circle. Overall time complexity is $O(n^4)$.

3.0.3 Algorithm

Algorithm

```
1: procedure MinimumEnclosingCircle(points)
2:     min_radius = ∞
3:     best_center = None
4:     for all (p1, p2) in combinations of points taken 2 at a time do
5:         center = MidPoint(p1, p2)
6:         radius = Distance(p1, center)
7:         if All other points lie within or on the circle then
8:             if radius < min_radius then
9:                 min_radius = radius
10:                best_center = center
11:            end if
12:        end if
13:    end for
14:    if best_center is None then
15:        for all (p1, p2, p3) in combinations of points taken 3 at a time do
16:            center, radius = Circumcircle(p1, p2, p3)
17:            if All other points lie within or on the circle then
18:                if radius < min_radius then
19:                    min_radius = radius
20:                    best_center = center
21:                end if
22:            end if
23:        end for
24:    end if
25:    return best_center, min_radius
26: end procedure
```

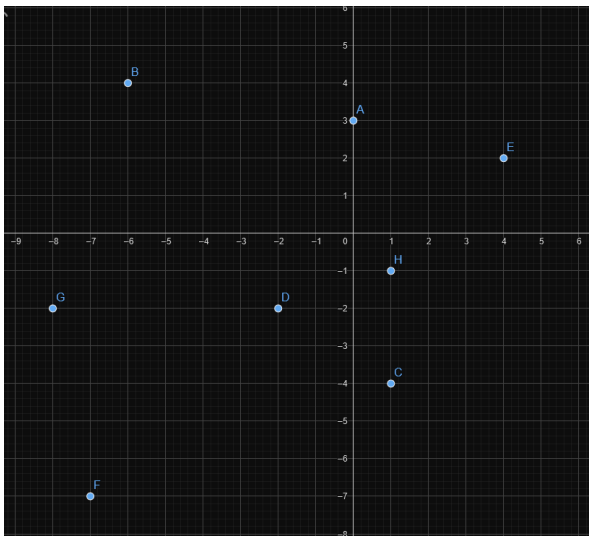
Chapter 4

Line Formation In Minimum Time

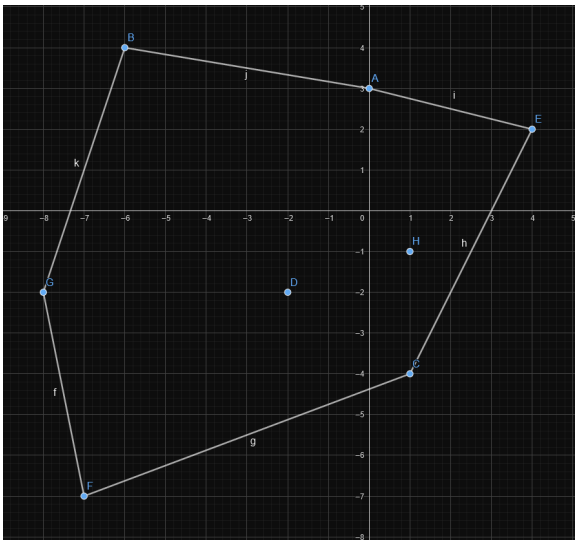
4.0.1 Procedure

We are trying to align all our robot on a line in minimum time. It is a kind of min(max()) problem. We have our robot spawned in an empty world and we know about their Coordinates. Suppose we have n robots.

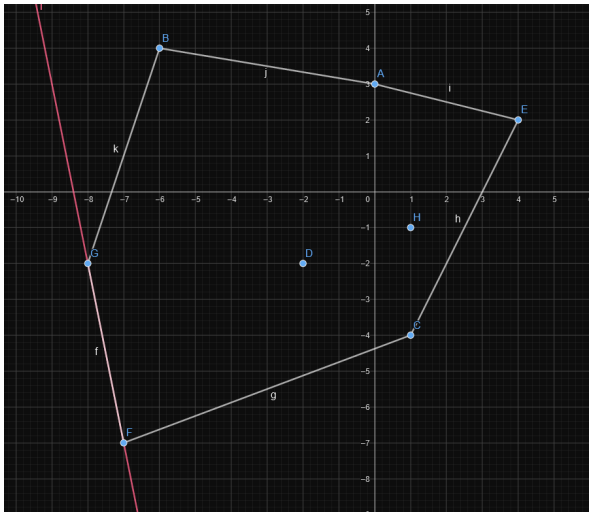
1. First, we need to obtain the Convex Hull of the coordinates of the robots, which can be achieved in $O(n \log n)$ or $O(n^2)$ worst case time complexity.
2. Once we have the Convex Hull, we iterate over the points on the hull, selecting two consecutive points and forming a line.
3. We then identify the point on the hull that is farthest from this line and store both the point and the distance value.
4. Next, we select the next two consecutive points on the hull and repeat the procedure described above.
5. After iterating through all the points on the hull, we select the point and the line with the minimum distance between them.
6. To obtain the minimum time line, we shift the line obtained in the previous step halfway towards the corresponding point with the minimum distance.
7. The resulting line, after the shift, represents our minimum time line.
8. Even if we utilize $O(n \log n)$ to obtain the convex hull, the process of selecting two points and iterating over all other points on the hull would be the bottleneck for the algorithm.
9. The overall worst case time complexity of the algorithm is $O(n^2)$.



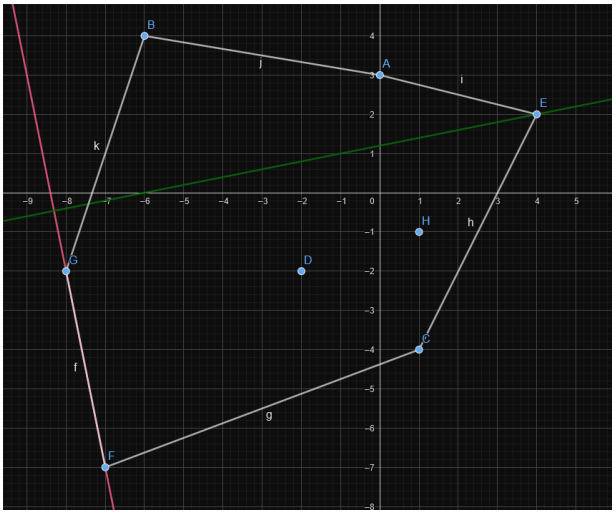
(a) Robots Spawned in an empty world



(b) Convex hull

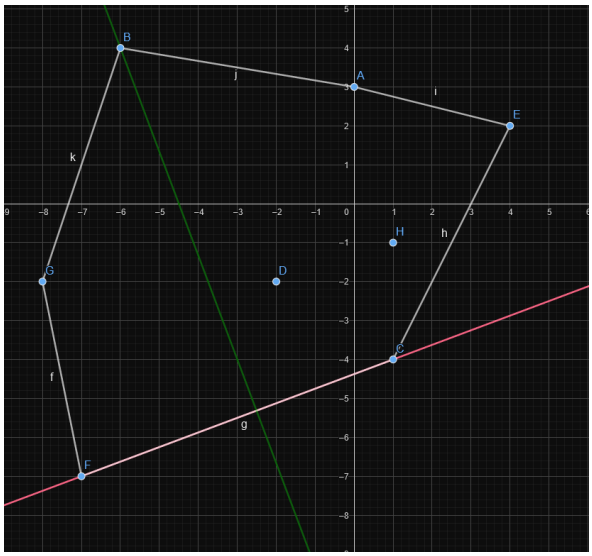


(c) Taking two adjacent points on Hull and creating a line

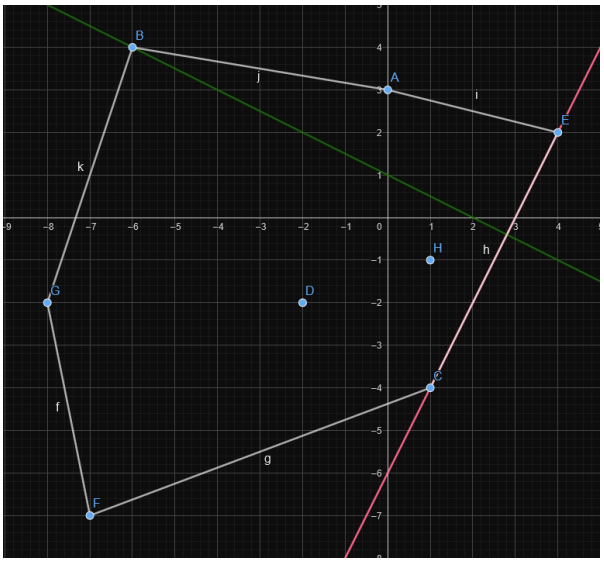


(d) Finding the point which is farthest from this Line

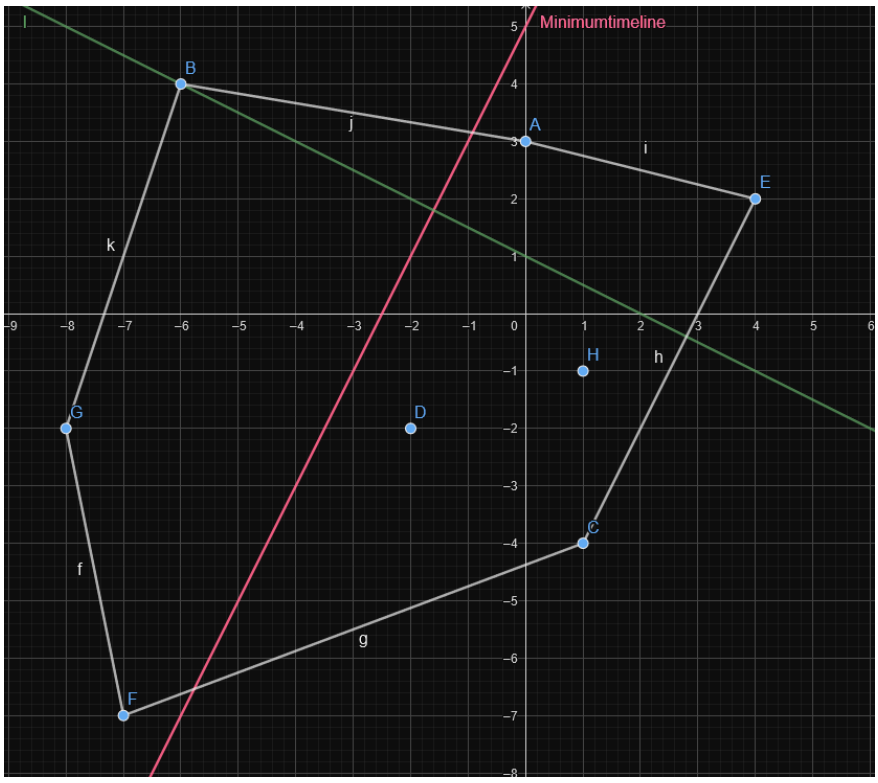
Figure 4.1: Illustration of the algorithm process.



(e) Selecting the next line and finding the point farthest from this line



(f) After all iteration, Line(C,E) and point B are the min(max(distance))



(g) Shift the Line(C,E) halfway towards the point B to obtain Minimum Time Line

Figure 4.2: Illustration of the algorithm process.

4.0.2 Algorithm

Algorithm

```
1: procedure FindConvexHull(robots_coordinates)
2:     Compute convex hull from robots_coordinates
3:     return convex_hull
4: end procedure
5: procedure FindMaxDistancePoint(convex_hull, line)
6:     max_distance = 0
7:     max_distance_point = None
8:     for point in convex_hull do
9:         distance = CalculateDistance(point, line)
10:        if distance > max_distance then
11:            max_distance = distance
12:            max_distance_point = point
13:        end if
14:    end for
15:    return max_distance_point
16: end procedure
17: procedure CalculateDistance(point, line)
18:    return distance of point from line
19: end procedure
20: procedure MinLine(convex_hull)
21:    min_max_distance = ∞
22:    min_max_line = None
23:    for i = 1 to length of convex_hull do
24:        line formed by i^th and (i + 1)^th point in convex_hull
25:        max_distance_point = FindMaxDistancePoint(convex_hull, line)
26:        max_distance = CalculateDistance(max_distance_point, line)
27:        if max_distance < min_max_distance then
28:            min_max_distance = max_distance
29:            min_max_line = line
30:            min_max_point = max_distance_point
31:        end if
32:    end for
33:    Shift min_max_line halfway towards min_max_point
34:    return shifted_min_max_line
35: end procedure
```

4.0.3 Algorithm Testing

To demonstrate how this problem can be solved, we'll employ a brute force method initially. This involves trying out different possibilities to find the shortest time line. Once we've identified this shortest possible time line, we'll compare it with the line produced by our algorithm. When we compare the two lines, we'll specifically examine the error in distances for the point that lies farthest from each line. For each test case we do the procedure below:

- We will first generate six random point which will denote the coordinate of our robots.
- To randomize the points, we will utilize `std::random_device`, which is a random number generator that produces non-deterministic random numbers, often utilizing hardware entropy sources.
- We have put a constrained on robot coordinates as $x, y \in [-5, 5]$.
- Now we will randomly generate values of m and c for min time line $y = mx + c$
- We will generate 2000 points for case 1 and 20000 points for case 2 each for m and c .
- $m, c \in [-100, 100]$.
- For each m we will iterate over all values of c to obtain line.
- For each $Line(m, c)$ we will iterate over all the generated robot coordinate and store the point which is farthest from this line and its distance value.
- Now we will try to find the minimum of the all maximum distance.
- Once we have this minimum distance , we will compare it with our algorithm's minimum distance. Then we will try to find the value of error function.
- Our error function is $F(e) = D_{bf} - D_a$, where D_a is the minimum distance given by our algorithm and D_{bf} is the minimum distance given by brute force.
- For our algorithm to be correct the value of $F(e)$ should always be greater than or equal to 0.

4.0.4 Results

We have obtained the Distance Error for each test case, it took around 3 minutes to compute 1000 test cases. We can see that the value of distance error is always greater than or equal to 0.

Case 1: 2000 points for m and c.

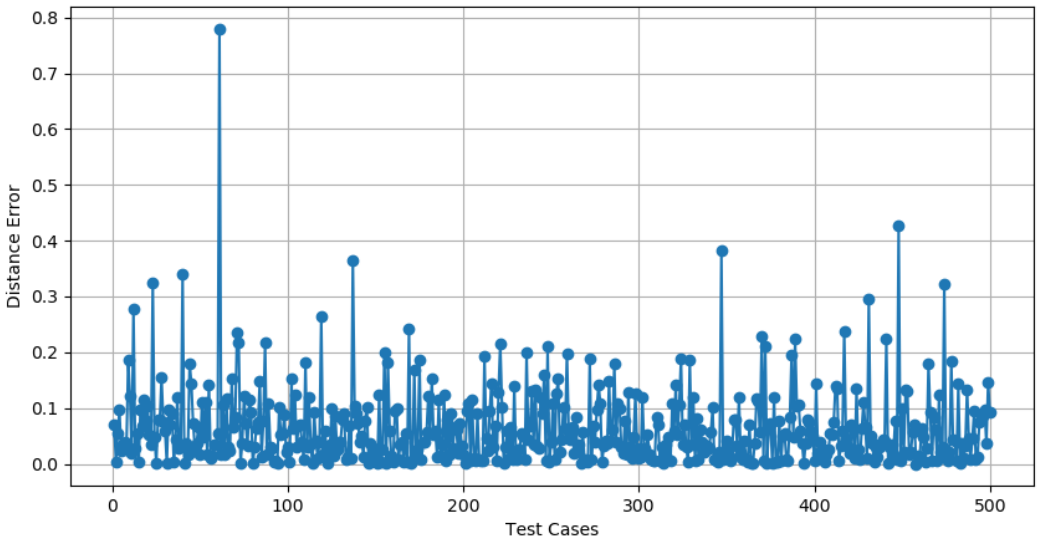


Figure 4.3: Test cases vs the distance error , Average Distance Error = 0.064

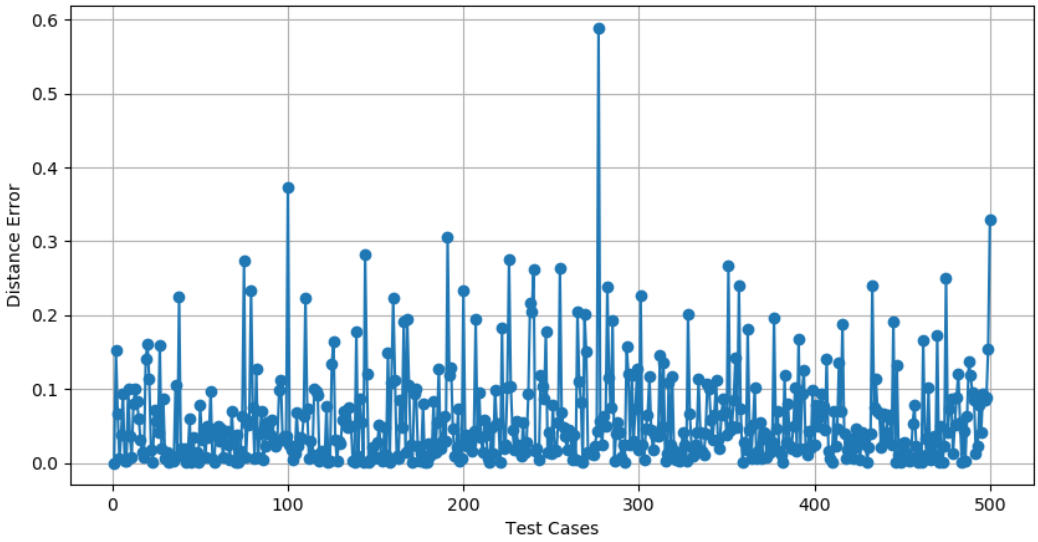


Figure 4.4: Test cases vs the distance error , Average Distance Error = 0.058

In this case it took around 3 hours to compute 500 test cases each. We can see that the value of distance error is always greater than or equal to 0 and the Average Distance Error also decreases.

Case 2: 20000 points for m and c.

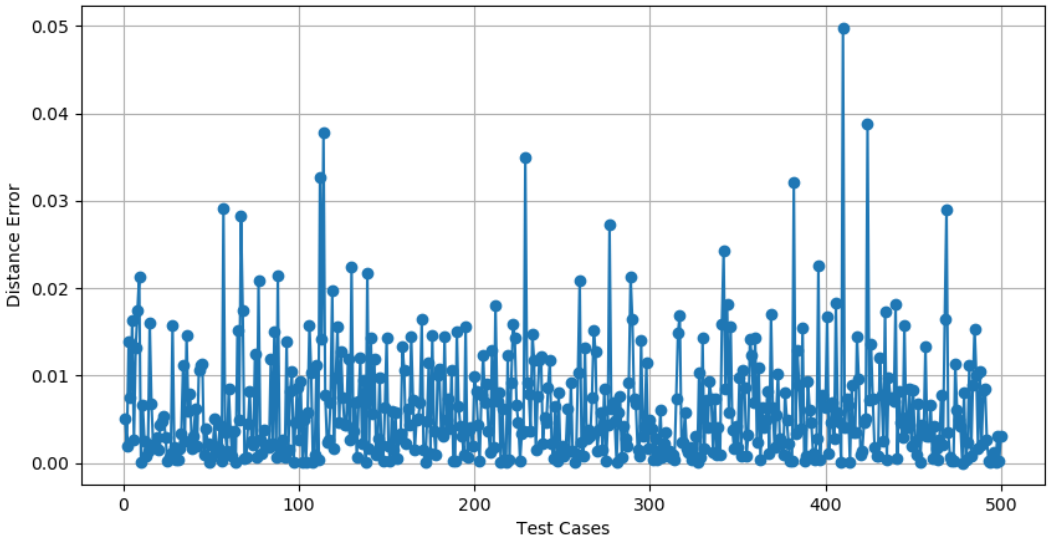


Figure 4.5: Test cases vs the distance error , Average Distance Error = 0.0063

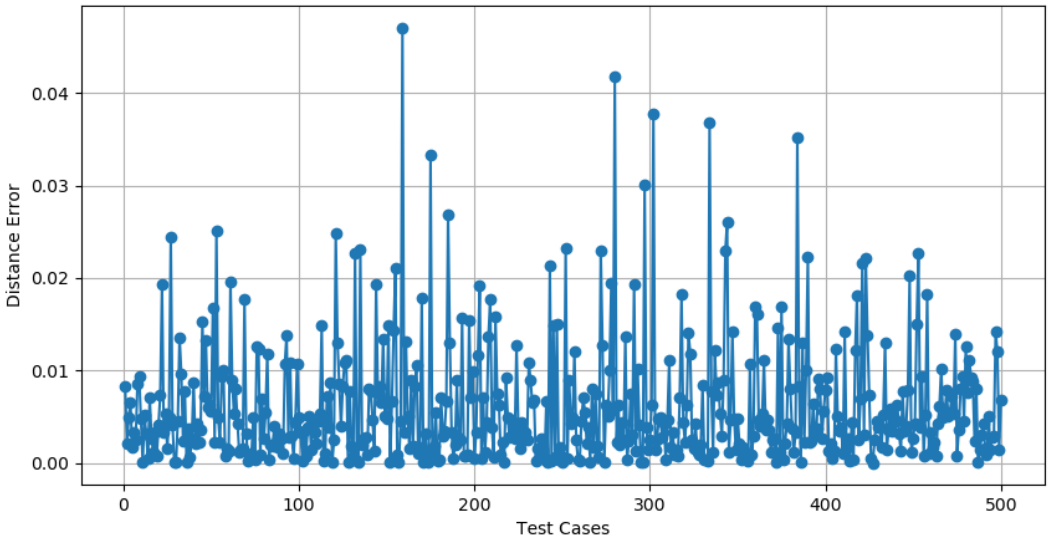


Figure 4.6: Test cases vs the distance error , Average Distance Error = 0.0061

We can see that as we increase the number of points for m and c the average distance error decreases.

4.0.5 Proof

Mathematical Formulation : Consider a line L defined by $y = mx + c$.

Now, let's introduce the function :
$d_i(x_i, y_i, L) = \frac{|y_i - mx_i - c|}{\sqrt{1+m^2}}$, where i ranges from 1 to n.

Hence, the objective is to determine suitable values for m and c that minimize the expression $\min_{\forall L} (\max_{\forall i}(d_i(L)))$.

Suppose we have 3 drones all on same z-axis, when we see from above, then we can have their reachability space as the circle which will grow with time as radius of all those circle is proportional to time.

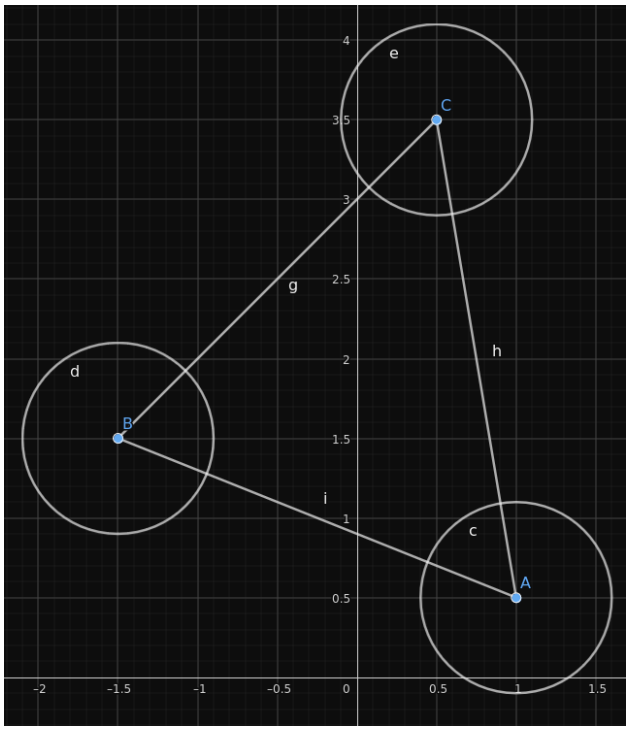
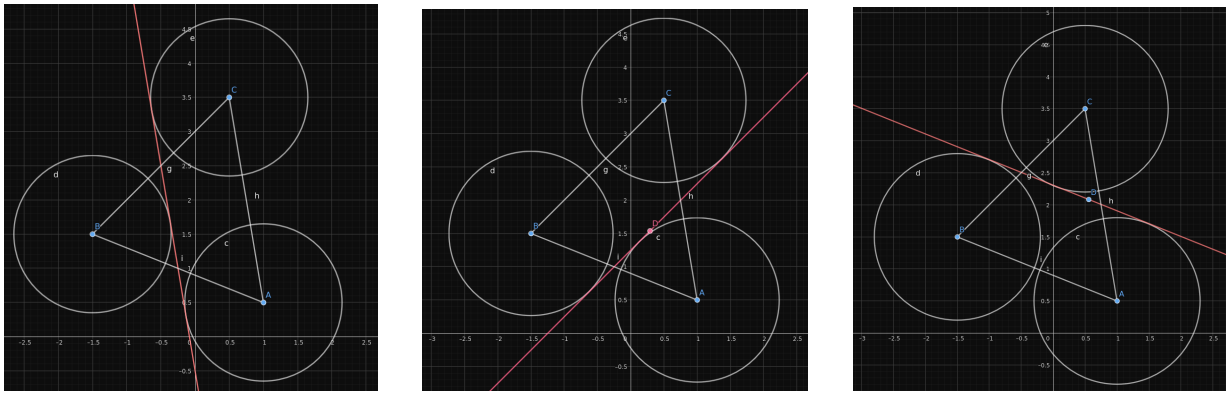


Figure 4.7: For r = 0.6 the reachability space of drones



(a) For r = 1.15 the reachability space of drones with the minimum-time line

(b) For r = 1.23 the reachability space of drones with a line

(c) For r = 1.3 the reachability space of drones with a line

Figure 4.8: Reachability space of drones with different radii

For three points, we can have three lines which can be formed and will touch all the three circles. For the minimum-time line, the minimum radius for which we can form the line touching all the circles (reachability space) is the line we are actually looking for. Since radius is proportional to time, the minimum radius will give us the minimum-time line.

When we have more than 3 points, even then the minimum time line is governed by only 3 points which lie on convex hull and the line would be parallel to one of the side of the convex hull. Here the line is parallel to segment(D,B).

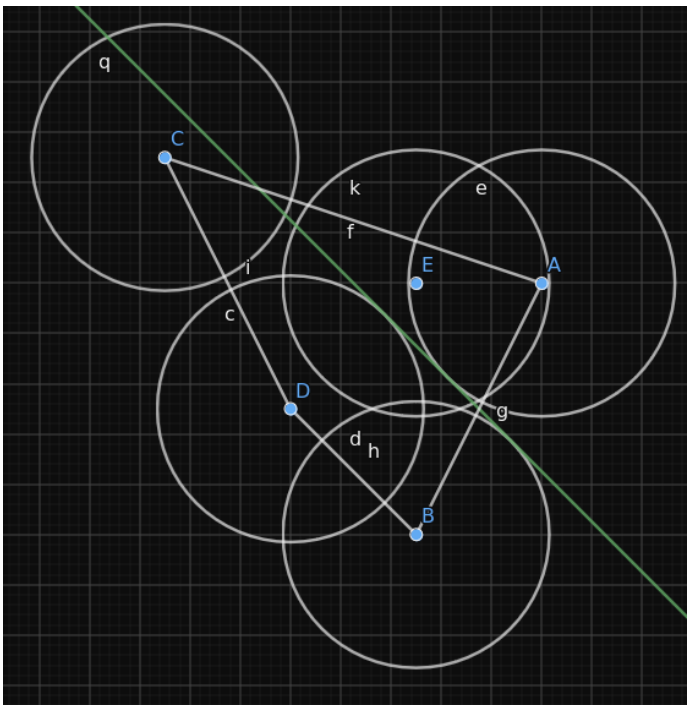


Figure 4.9: Randomly generated 5 points with their reachability space and Minimum-time Line

Reason why one of the side of the convex hull will be parallel to the Minimum-time line is because when we drop a perpendicular on line from point B and D , it will form a rectangle with sides as D,G,F,B. We will not be considering any point which is inside the Convex Hull as they will always be closer to the minimum-time line.

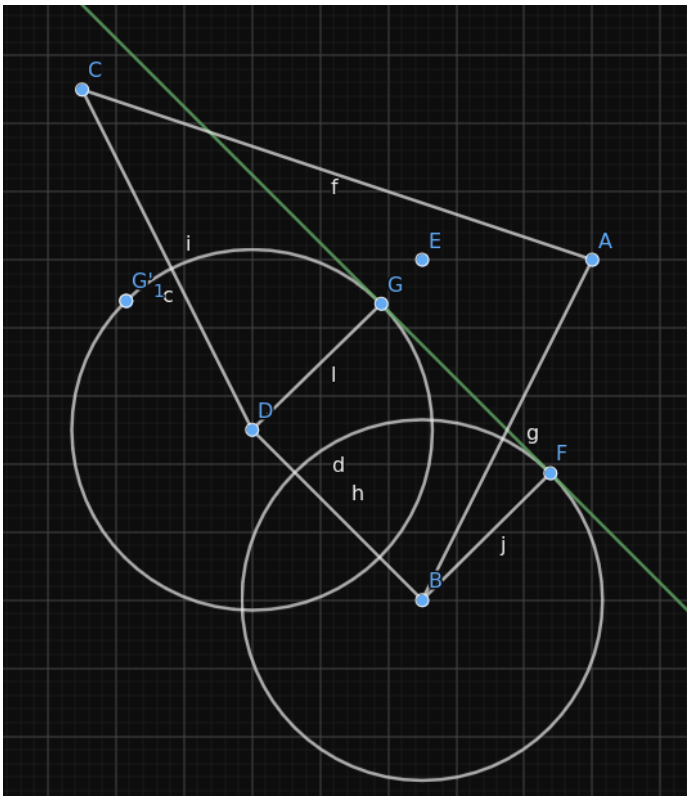


Figure 4.10: Reachability space of robot D,B forming a rectangle D-G-F-B

Chapter 5

Circle Formation in Minimum Time

5.0.1 Procedure

We are attempting to arrange all our robots on a circle in minimum time. Our robots are initially positioned in an empty world, and we possess information about their coordinates. Let the number of robots be represented by n. For $n \leq 3$, it is always possible to obtain a circle passing through all the provided robot coordinates.

For $n \geq 4$,

1. We select two pair of two robot coordinates.
2. Calculate the perpendicular bisector for each pair.
3. Determine the intersection point of these two perpendicular bisectors.
4. The intersection point serves as the center of minimum time circle when the difference between maximum and minimum distances from all points to this center is minimum for all combinations.
5. If the condition in above step holds true, then the intersection point becomes the center of the minimum time circle, with its radius calculated as average of the maximum and minimum distances of all the point from this centre.
6. If not then we will go to step 1 and select two new pairs.

The overall worst case time complexity of the algorithm is $O(n^5)$.

For a set of 7 randomly generated points, we determine the minimum-time circle, positioned equidistantly between two concentric circles. The center of this minimum-time circle (MTC) can be identified by the intersection of two perpendicular bisectors formed by points A, D, and points B, C.

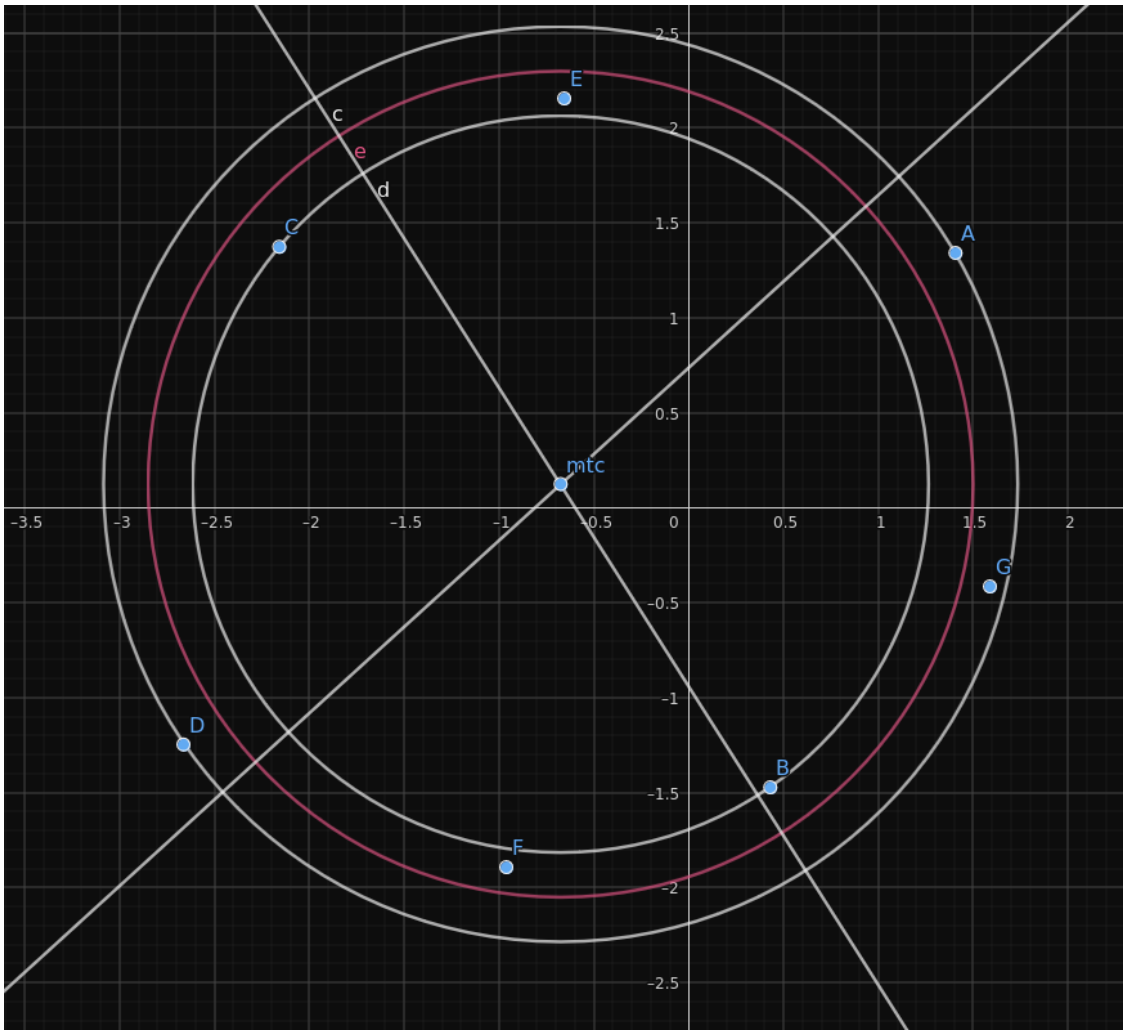


Figure 5.1: Minimum-time Circle is shown in pink

5.0.2 Algorithm

Algorithm

procedure IntersectionPoint($line1, line2$)
    Solve the system of equations formed by $line1$ and $line2$ to find the intersection point
    return the intersection point
end procedure
procedure PerpendicularBisector($point1, point2$)
    Compute the midpoint between $point1$ and $point2$
    Determine the line perpendicular to the line segment connecting $point1$ and $point2$, passing through the midpoint
    return the perpendicular bisector line
end procedure
procedure MinimumTimeCircle(robo_coor)
    $min\_difference = \infty$
    min_time_center = None
    $min\_time\_radius = 0$
    for $i = 1$ **to** length of robo_coor **do**
        for $j = 1$ **to** length of robo_coor **do**
            for $k = 1$ **to** length of robo_coor **do**
                for $l = 1$ **to** length of robo_coor **do**
                if $i \neq j$ and $i \neq k$ and $i \neq l$ and $j \neq k$ and $j \neq l$ and $k \neq l$ **then**
                $line1$ = PerpendicularBisector(robo_coor[i], robo_coor[j])
                $line2$ = PerpendicularBisector(robo_coor[k], robo_coor[l])
                $center$ = IntersectionPoint($line1, line2$)
                Calculate the max and min distances from all robo_coor to $center$
                $difference$ = max_distance − min_distance
                if $difference < min\_difference$ **then**
                $min\_difference = difference$
                $min\_time\_center = center$
                min_time_radius = (max_distance + min_distance)/2
                end if
                end if
            end for
        end for
    end for
    return the min_time_center and the min_time_radius
end procedure

5.0.3 Algorithm Testing

To illustrate the solution to this problem, we will start with a brute force approach. This involves exploring various possibilities to determine the circle with the shortest time. After identifying the shortest time circle, we will compare it to the one generated by our algorithm. Specifically, we will evaluate the distance error for the point that is farthest from each circle. For each test case, we will follow the procedure below:

- Initially, we will generate ten random points to represent the coordinates of our robots.
- To randomize the points, we will utilize `std::random_device`, which is a random number generator that produces non-deterministic random numbers, often utilizing hardware entropy sources.
- We have constrained the robot coordinates to $x, y \in [-5, 5]$.
- Next, we will randomly generate 100,000 points with coordinates $x, y \in [-5, 5]$ and consider them as potential centers of the minimum time circle.
- For each assumed circle center, we will incrementally increase the circle radius r in steps of 0.05 within the range $r \in [0, 5]$.
- With each circle formed, we will determine the farthest distance of a point from this circle.
- Upon reaching $r = 5$, we will select the next randomly generated point from a pool of 100,000 points and repeat the process.
- Once we have obtained the minimum distance using this method, we will compare it with the minimum distance calculated by our algorithm. Subsequently, we will evaluate the error function.
- Our error function is defined as $F(e) = D_{bf} - D_a$, where D_a represents the minimum distance given by our algorithm, and D_{bf} represents the minimum distance obtained by brute force.
- For our algorithm to be deemed correct, the value of $F(e)$ should always be greater than or equal to 0.

5.0.4 Result

We have obtained the Distance Error for each test case, it took around 506 seconds to compute 500 test cases. We can see that the value of distance error is always greater than or equal to 0.

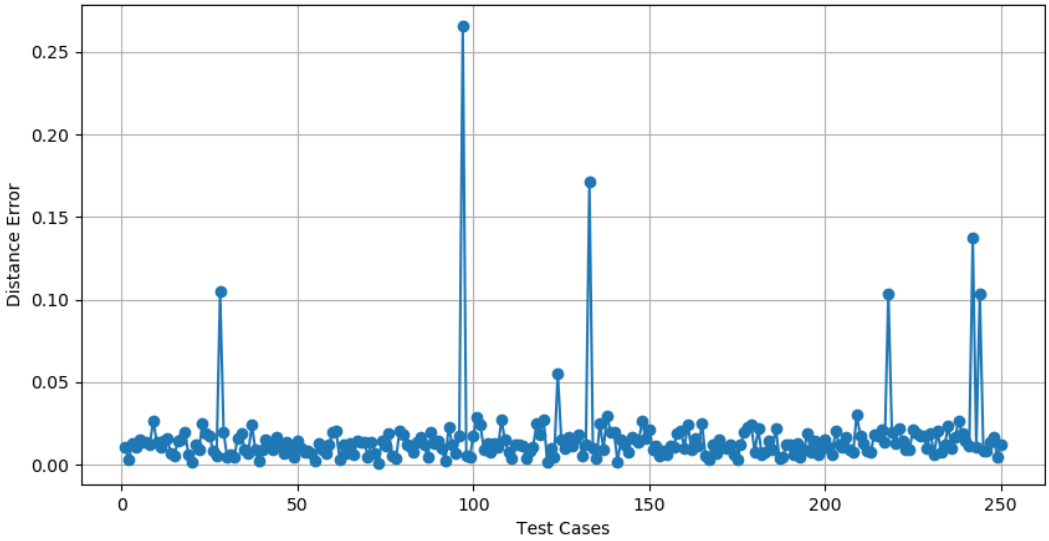


Figure 5.2: Test cases vs the distance error , Average Distance Error = 0.0160

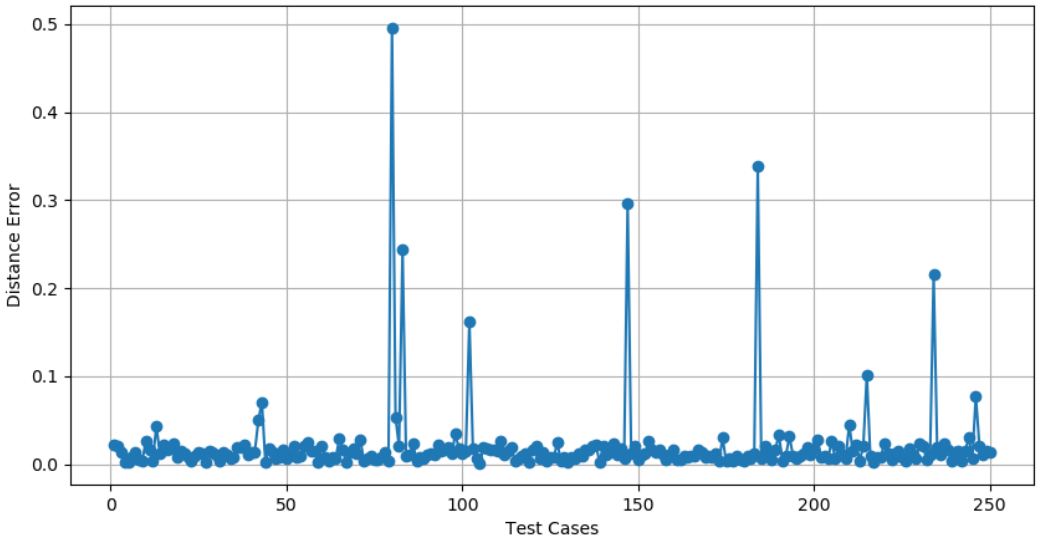


Figure 5.3: Test cases vs the distance error , Average Distance Error = 0.0206

5.0.5 Proof

The task of determining the out-of-roundness error of a production part, represented by a finite set of boundary points, can be mathematically formulated as finding the center and radius of a circle that minimizes the maximum perpendicular distance between these points and the circle [4]. Let

$$S = \{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$$

be the given set of boundary points in a plane. The maximum Euclidean distance between S and a circle with center (x, y) and radius r is

$$f(x, y, r) = \max_{i=1,\ldots,n} \left(\sqrt{(a_i - x)^2 + (b_i - y)^2} - r \right). \tag{1}$$

The minimax center estimation problem refers to finding the center (x^*, y^*) and the radius r^* that makes $f(x, y, r)$ as small as possible. The out-of-roundness error of the inspected part will be $f(x^*, y^*, r^*)$. A problem equivalent to minimizing (1) is to find $(x^*, y^*, \alpha^*, \beta^*)$ that solve

$$\text{P1:} \quad \min \frac{1}{2}(\alpha - \beta) \tag{2}$$

subject to

$$\sqrt{(a_i - x)^2 + (b_i - y)^2} - \alpha \leq 0, \quad i = 1, \ldots, n, \tag{3}$$

$$-\sqrt{(a_i - x)^2 + (b_i - y)^2} + \beta \leq 0, \quad i = 1, \ldots, n. \tag{4}$$

Note that a feasible point (x, y, α, β) to the above problem has the intuitive interpretation of two concentric circles with center (x, y), and with α and β being the radii of the largest and smallest circle, respectively. The points in S belong to the region bounded by these two circles.

The above Problem P1 has the following two properties.

Property 1. The objective function is linear, and the optimal objective function value is greater than or equal to zero. Only the following two cases produce a zero solution:

1. $n \leq 3$: there always exists a circle passing through three or fewer points in a plane. This includes the degenerate case of three collinear points, where the circle has an infinite radius. The two circles defined by the optimal solution of the problem will have the same radius, i.e., $\alpha^* = \beta^*$

1. $n \geq 4$ and all the points in S are on the boundary of a common circle.

Property 2. The feasible region is not convex since the constraints in (4) are defined by concave non-linear functions.

Case 1: $n = 4$

The following theorem gives necessary optimality conditions for this problem.

Theorem 1. Every optimal solution of Problem $P1$ with $n = 4$ satisfies the following two conditions:

1. For $i = 1, \ldots, 4$, either the i-th constraint in (3) or the i-th constraint in (4) is active.

2. At least one constraint in (3) and one in (4) are active.

Case 2: $n > 4$

The following two theorems provide the necessary background for the forthcoming analysis of the minimax problem when there are more than four boundary points.

Theorem 2. Every optimal solution of Problem $P1$ with $n > 4$ has the following properties:

1. At least four constraints, defined by four different points of S, are active.

2. At least one constraint in (3) and one in (4) are active.

Define $P2(k)$ as the minimax problem corresponding to the k-th combination of four points taken from the original set S. Note that there are $\frac{1}{24}n(n-1)(n-2)(n-3)$ possible problems of the form $P2(k)$.

Theorem 3. If (x_k, y_k, a_k, b_k) is an optimal solution to Problem $P2(k)$, then $\frac{1}{2}(a_k - b_k)$ is a lower bound for $P1$.

Since the maximum number of iterations of Algorithm is equal to the number of combinations of n points taking 4 at a time, i.e., $\binom{n}{4} = \frac{n(n-1)(n-2)(n-3)}{4!}$,the complexity of the algorithm is $O(n^5)$. We can further decrease the average case time complexity by utilizing the fact that the outer circle formed by the 2 points will always lie on the convex hull. So, $\Theta(n^3 \cdot h^2)$, where h is the number of points on the convex hull. If $h \ll n$, then $\Theta(n^3 \cdot h^2) \approx \Theta(n^3)$.

Chapter 6

Conclusion

In this work, we conducted a detailed study of consensus algorithms and formation control in multi-robot systems, focusing on achieving consensus in both one-dimensional and two-dimensional spaces in minimum time. Using the ROS 2 architecture and the TurtleBot3 platform for experimentation, we developed and validated efficient algorithms for line and circle formations through simulations and theoretical proofs.

By implementing these algorithms on ROS 2 and TurtleBot3, we demonstrated the practical viability and scalability of our approach. This work lays a foundational framework for future research in swarm robotics and multi-agent systems, enhancing the efficiency and responsiveness of robotic swarms in dynamic environments.Our objective was to align robots in circular and straight line formations within the minimum possible time, assuming a constant velocity for all robots. By minimizing the distance each robot travels to reach the computed circle and line, we achieved our goal efficiently. The algorithms presented in this thesis provide a robust solution for formation control in multi-robot systems, paving the way for further advancements in this field.

6.1 Future Work

- We have not yet incorporated obstacle avoidance in our algorithm. There can be instances where two robots collide with each other when forming a shape.
- Additionally, we can continue to find different shapes which can be formed in minimum time like triangle and so on.

Chapter 7

APPENDIX

7.1 ROS 2 Architecture

Robot Operating System (ROS) 2 [3] is a set of software libraries and tools that help you build robot applications. It is designed to work with multiple platforms, provide real-time capabilities, and support distributed systems. The core components of ROS 2 architecture include nodes, topics, services, actions, and the ROS 2 middleware (RMW) interface. We are using Turtlebot3 Burger for the experiment.

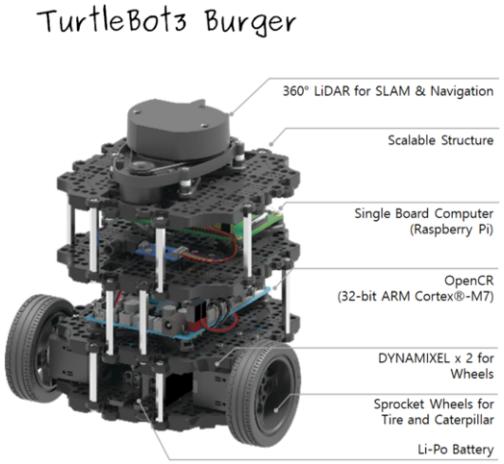


Figure 7.1: Turtlebot3

7.1.1 Nodes

Nodes are the fundamental building blocks of a ROS 2 system. Each node is an independent process that performs computation. Nodes can communicate with each other by passing messages. This modular approach allows for the development of complex robot behaviors by combining simpler, reusable modules.

7.1.2 Topics

Topics are named buses over which nodes exchange messages. A node that wants to share information publishes messages to a topic, while nodes that are interested in that information subscribe to the topic. This publish/subscribe model decouples the production and consumption of information, enhancing modularity and scalability.

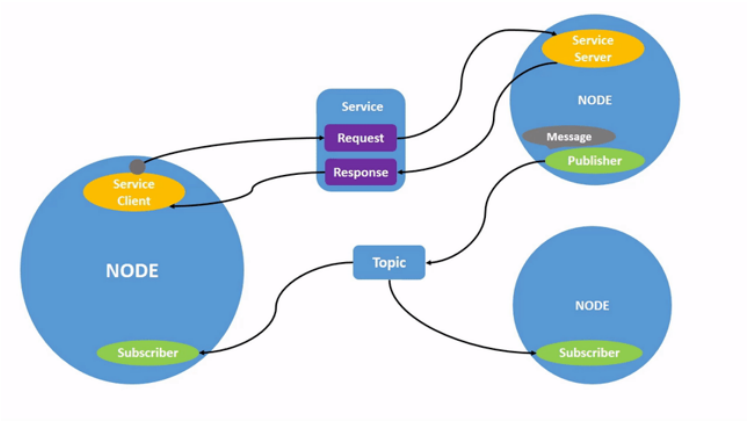


Figure 7.2: Example of ROS 2 Nodes

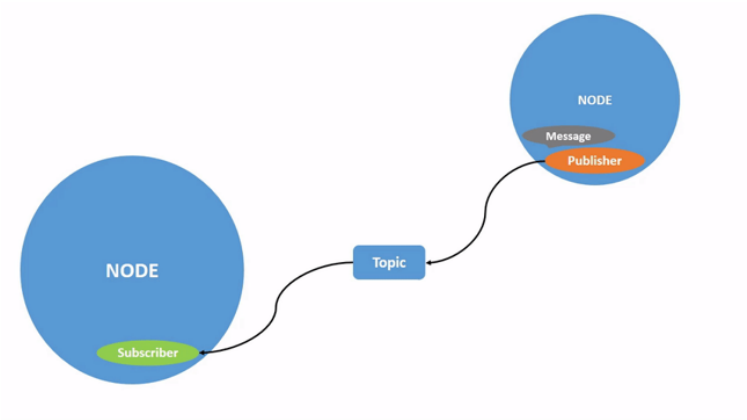


Figure 7.3: ROS 2 Publish/Subscribe Model

7.1.3 Odometry in ROS 2 TurtleBot3

In ROS 2, TurtleBot3 publishes odometry data on the `/odom` topic using the `nav_msgs/Odometry` message type. This data includes:

- **Header**: Timestamp and coordinate frame information.
- **Pose**: Robot's position and orientation.
- **Twist**: Linear and angular velocity.

The TurtleBot3 publishes odometry data by reading sensor inputs (e.g., wheel encoders, IMU). We will only be using odometry data for formation and Consensus.

Bibliography

[1] Tanja Katharina Kaiser, Marian Johannes Begemann, Tavia Plattenteich, Lars Schilling, Georg Schildbach, and Heiko Hamann, "ROS2SWARM - A ROS 2 Package for Swarm Robot Behaviors" in IEEE International Conference on Robotics and Automation (ICRA), 2022.

[2] ChoiRbot: A ROS 2 Toolbox for Cooperative Robotics Andrea Testa , Andrea Camisa , Giuseppe Notarstefano ,IEEE Robotics and Automation Letters , 2021.

[3] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, 2022.

[4] The minimax center estimation problem for automated roundness inspection, Jose A. ventura, Sencer yeralan, European Journal of Operational Research (1989) 64-72.

You can find the code repository for TB3_Task on GitHub: `https://github.com/Ruudddiiii/TB3_Task`.