Collision Avoidance Simulation Using Voronoi Diagrams in a Centralized System of Holonomic Multi-agents

Leduin José Cuenca Macas(✉) and Israel Pineda

Yachay Tech University, Hacienda San José s/n Proyecto Yachay, Urcuquí, Imbabura 100115, Ecuador
{leduin.cuenca,ipineda}@yachaytech.edu.ec

Abstract. This work solves the Collision Avoidance problem in a simulation of a centralized system of holonomic multi-agents in a two dimensional space free of static obstacles. For this, we propose an implementation of three modules in an architecture: Threat Assessment Strategy (TAS), Path Planning Strategy (PPS), and Path Tracking Strategy (PTS). The Buffered Voronoi Cells represent the TAS. The PPS modules use two algorithms: the Analytical Geometric Algorithm (AGA) and the Receding Horizons Control (RHC) based on Quadratic Programming (QP) Algorithm. Finally, PTS controls the tracking according to fixed distance magnitudes in each iteration. The analysis of the results considers the computational execution time, the number of steps until convergence, and the calculation of optimal values. Also, these results are compared with the Optimal Reciprocal Collision Avoidance (ORCA) algorithm. In this way, our proposal successfully addresses and solves the collision avoidance problem but takes more execution time and number of steps compared with the ORCA algorithm. Besides, the number of steps of AGA is closer to ORCA, producing promising results with an accuracy of 95%.

Keywords: Collision avoidance · Voronoi diagrams · Convex optimization · Quadratic programming · Path planning · Simulation

1 Introduction

Collision Avoidance (CA) is the process of preventing two or more physical objects from having intersecting boundaries in space-time, taking variables like time and distances into account. In this way, CA is studied due to its practical applications, mainly in path planning for ships [1], autonomous robots [2], aircraft and unmanned aerial vehicles [3], using different mathematical and computational techniques such as geometric analysis, control modeling with optimization, game theory, dynamical systems, and artificial intelligence [4]. Therefore, this problem challenges researchers to simulate the natural ability of complex living beings or processes to avoid physical collisions and react accurately.

J. Herrera-Tapia et al. (Eds.): TICEC 2022, CCIS 1648, pp. 18–31, 2022.
https://doi.org/10.1007/978-3-031-18272-3_2

In the present work, we deal with CA algorithms for centralized autonomous holonomic multi-agent. We use the safe distance-based method as a Threat Assessment Strategy (TAS). A combination of Optimization and Geometrical-based strategies with heuristics to break deadlocks serve as Path Planning Strategy (PPS). Moreover, the Path Tracking Strategy (PTS) uses Euclidean geometry. The project proposal involves using Buffered Voronoi Cells (BVC), Analytical Geometric Algorithm (AGA), and Quadratic Programming (QP) based Receding Horizons Control (RHC) algorithm. The generated algorithms only require detecting the relative positions with a centralizing character. Therefore, it is very suitable for online deployment, as it does not require a concurrent communication network. We demonstrate the capabilities of our algorithm by comparing it to the Optimal Reciprocal Collision Avoidance (ORCA) in a benchmark simulation scenario, and we present the results of over 2160 experimental trials in total. Our work follows the ideas of Zhou *et al.* [5].

2 Related Work

2.1 Collision Avoidance

There are at least two CA Design Control Architectures for autonomous agents [2]: the Multi-Layer CA System divides responsibility for different objectives into layers, and the Unified-Design CA System combines two blocks for integrated objectives and identical control inputs. Both architectures usually comprise several sub-modules or strategies: TAS, PPS and PTS.

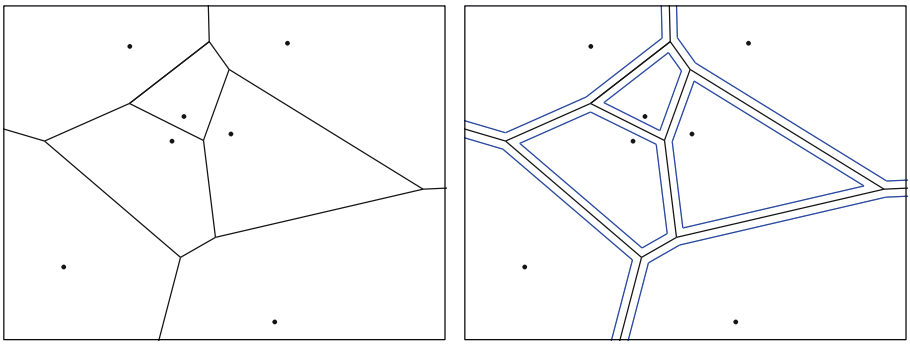
TAS provides an assessment and subsequent warnings of the potential threat to CA. The result of the TAS calculation is the key to triggering the subsequent actions of the CA architecture. In this way, TAS feeds the decision-making strategy on the appropriate action of the moving object. In addition, TAS takes care of the threshold or tolerance limits around obstacles or any physical object in the environment. Once this object violates certain conditions, the CA system activates the path planning module to re-plan the current trajectory. The risk can be measured by any means, including distance, speed, and acceleration of the moving object relative to the elements of the environment [4].

The PPS re-plans a collision-free route while the vehicle moves once the TAS identifies the potential collision threat. This new route may differ from the previous route planned by the PPS. In addition, an ideal PPS considers the risk of collisions involved in changing the current kinematics of the vehicle. First, the strategy guarantees enough space for the mobile to maneuver without frontal or side collisions. Then it needs to make sure that there is no potential risk with another obstacle after the maneuver. Finally, the new trajectory must consider the mechanical limitations and internal implications of the moving object [2].

PTS algorithms act as a path following controller to ensure the vehicle or mobile robot successfully avoids collisions. An enfficient PTS timely tracks the reference re-planned path by producing the required low-level control actions and output suitable interventions. For this reason, different scenarios demand particular CA actions [2].

2.2 Voronoi Diagrams

Given a set of two or more but a finite number of distinct points in the Euclidean plane, we associate all the locations in that space with the closest members of the set of points concerning the Euclidean distance. The result is a tessellation is called *Voronoi Diagram* generated by the point set, and the regions constituting the Voronoi diagram are called *Voronoi cells*. New compressed cells can be generated within each cell according to the safety radius distance. These new cells are the *Buffered Voronoi Cells*. Figure 1 shows this description graphically.



(a) Voronoi Diagram (b) Buffered Voronoi Cells

Fig. 1. Example of Voronoi diagrams

2.3 Multi-agent Navigation

We can categorize the multi-agent systems based on the different views of planning approaches into two classes: centralized and decentralized [6]. We call Centralized Policy for multi-agents to a plan to generate collective actions starting from the states of the global system, solving the problems of the agents in a unified way. On the other hand, in a decentralized system, the agents partially observe the global system state and make local decisions. Thus, the planned mapping from local knowledge to local actions is called a Decentralized Policy for multi-agent.

3 Methodology

3.1 Model Proposal

We propose a simulation architecture for a holonomic multi-agent system in a two-dimensional environment free of obstacles. Our proposed architecture uses three main components: TAS, PPS, and PTS. The simulation uses Voronoi Diagram as the core of its TAS. The PPS component can use either AGA or the

QP-based RHC algorithm; both algorithms have the same input parameters, but with the difference that the QP-based RHC algorithm has a solver for the optimization problem. Additionally, the PPS has heuristics to break deadlocks during navigation. We use Euclidean distance for PTS. The two architecture variations (AGA and QP-based RHC) are compared with the ORCA algorithm to contrast result metrics.

3.2 Two-Dimensional Environment

The environment is represented by a two-dimensional Cartesian plane. Both main axes contain the real set of numbers for the coordinates generation. Thus, the position of the robots and the Voronoi vertices are any real coordinates. The initial and final positions of the simulated agents are antipodal and form a circular figure centered at the point (0, 0). Let N be the number of robots; the distance from each position to (0,0) equals $4 \times \sqrt{N}$ plus an offset belonging to the set $[-\frac{1}{0.4 \times N}, 0)$. The distance formula is selected to generate configurations similar to Zhou *et al.* [5]. There is an approximation for Voronoi infinity vertices for practical purposes. The magnitude generated for their respective infinity edges is equal to 20 times the number of simulated robots; for example, if there exist five robots, then the magnitude is equal to 100. There are no static obstacles or passages, just moving robots defined as common obstacles. The precision of the numeric values depends on the programming language of the implementation; in this case, Python.

3.3 Heuristics to Deal with Deadlock

Deadlock is an imminent problem in CA. It happens when some robots block the paths of each other so that they cannot reach their goal. For those robots whose goal positions are not inside their own BVC, in a deadlock situation, each robot must be at the closest point to the goal position on its BVC. The closest point in the BVC of robot i, g_i^*, to the goal $p_{i,f}$, must be either at a vertex or on edge such that a line from $p_{i,f}$ to g_i^* is perpendicular to this edge.

No existing algorithm can provably avoid deadlock without central computation to our best knowledge. Instead, most distributed algorithms attempt to alleviate the problem through sensible heuristics. Similarly, we propose two heuristic methods that perform well in solving deadlock. At the same time, we establish a deadlock threshold value due to the limited range of numerical data types of programming languages.

The first heuristic is the Right-Hand Rule, which detours each robot from its right side when encountering other robots in deadlock situations. If the application of this heuristic causes the robot to leave its BVC, then we prefer not to move the robot in that step. This preference is made until the movement of other robots can break the deadlock or allow the movement of the blocked robot.

The second one considers the previous positions, ensuring a high level of breaking deadlock situations, abrupt and zigzag movements. In this case, we analyzed the distance between previous positions with the closest point in the BVC.

3.4 QP-Based Receding Horizons Control Algorithm

The Quadratic Programming Based Algorithm has its foundations on the Receding Horizon Control, also known as Model Predictive Control. Its applications include scenarios such as industrial and chemical process control, supply chain management, stochastic control in economics and finance, revenue management, hybrid vehicles, automotive and aerospace applications [7].

With RHC, an optimization problem is solved at each time step to determine a plan of action over a fixed time horizon. Then, the first input from the plan is applied to the system. Next time we repeat the planning process, solving a new optimization problem with the time horizon shifted one step forward. The optimization problem estimates future quantities based on available information at each time step. The control policy involves feedback since real-time measurements determine the control input [8]. In this way, we model the following optimization problem,

Problem 1 (Receding Horizon Path Planning).

$$\begin{aligned} \min_{\overline{p}_1,\ldots,\overline{p}_T} J_i = & \sum_{t=0}^{T-1} \left(\left(\overline{p}_{i,t} - p_{i,f}\right)^{\intercal} Q \left(\overline{p}_{i,t} - p_{i,f}\right) + u_{i,t}^{\intercal} R u_{i,t}\right) \\ & + \left(\overline{p}_{i,T} - p_{i,f}\right)^{\intercal} Qf \left(\overline{p}_{i,T} - p_{i,f}\right) \end{aligned} \tag{1}$$

subject to:

$$\overline{p}_{i,t+1} = A\overline{p}_{i,t} + Bu_{i,t},\ t = 0, \cdots, T-1, \tag{2}$$

$$\overline{p}_{i,t} \in \overline{\mathcal{V}_i},\ t = 1, \cdots, T, \tag{3}$$

$$\overline{p}_{i,0} = p_i, \tag{4}$$

$$\|u_{i,t,x}\| \leq u_{x,max},\ t = 0, \cdots, T-1, \tag{5}$$

$$\|u_{i,t,y}\| \leq u_{y,max},\ t = 0, \cdots, T-1. \tag{6}$$

In Problem 1, the cost function J_i is a summation of the intermediate state and terminal costs. In Eq. 1, $p_{i,f}$ is the final position, $\overline{p}_{i,0}$ to $\overline{p}_{i,T}$ and $u_{i,0}$ to $u_{i,T-1}$ are the path and inputs to be planned, respectively. The positive definite or semi-definite matrices Q, R, and Q_f are weight factors to balance the three costs. The decision variables for this standard QP problem are $\overline{p}_{i,1}$ to $\overline{p}_{i,T}$. Constraint 2 ensures that the path is feasible with the dynamics of the robots. In the present work, the holonomic robots do not need to fix the matrix values because they can move in any direction, and their velocity is the same. Constraint 3 restrains

the planned path inside the corresponding BVC $\overline{\mathcal{V}_i}$ of robot i. This constraint can be written explicitly in the form of a set of linear inequalities:

$$\overline{p}_{i,t}^{\intercal}\epsilon_j \leq 0, \qquad t = 1, \cdots, T, j \in \{1, \cdots, N\}, j \neq i, \tag{7}$$

where ϵ_j is a vector representing an edge of $\overline{\mathcal{V}_i}$ that separates robot i from robot j. Constraint 4 makes sure the planned positions start from the current position of the robot, and, finally, the lower and upper bounds for the input $u_{i,t}$ are written component-wisely in constraints 5 and 6.

We use the Python-embedded modeling language CVXPY to solve the QP problem [9]. It allows expressing a convex problem naturally that follows the mathematical conventions rather than in the restrictive standard form of solvers. However, the usage of CVXPY is similar to a usual Python library. Algorithm 1 contains the most fundamental parts of the pseudocode to solve the QP problem and get the closer coordinate in the robot cell to its final position.

Algorithm 1: QP Solver

Data: agent, r, T
Result: $\overline{p}_T$
initialize m, A, B, R, Q, Q_f, $\overline{p}$, u, $u_{max,x}$, $u_{max,y}$;
cost = 0;
constraints = [ ];
p_i = agent initial position;
p_f = agent final position;
Add $\overline{p}_{i,0} = p_i$ to constraints;
for *t in range(T)* **do**
    Sum $(\overline{p}_t - p_f)^{\intercal} Q (\overline{p}_t - p_f) + u_t^{\intercal} R u_t$ to cost;
    Add $\overline{p}_{t+1} = A\overline{p}_t + Bu_t$ to constraints;
    Add $\|u_{0,t}\| \leq u_{max,x}$ to constraints;
    Add $\|u_{1,t}\| \leq u_{max,y}$ to constraints;
    for *each position of agent neighbors* **do**
        p_j = neighbor position;
        Add $p_t - \frac{p_i + p_j}{2}^{\intercal} (p_j - p_i) + (r * \|p_j - p_i\|) \leq 0$ to constraints;
    end
end
Sum $(\overline{p}_T - p_f)^{\intercal} Q_f (\overline{p}_T - p_f)$ to cost;
for *each position of agent neighbors* **do**
    p_j = neighbor position;
    Add $p_T - \frac{p_i + p_j}{2}^{\intercal} (p_j - p_i) + (r * \|p_j - p_i\|) \leq 0$ to constraints;
end
Solve the CVXPY problem();

3.5 Analytical Geometrical Algorithm

The QP-based RHC Algorithm generates a control policy optimal over the planning horizon at the expense of solving a QP problem online at each time step. Furthermore, we can solve the QP with an Analytical Geometric Algorithm for the particular case of no intermediate cost terms, which executes much faster than the QP, while CA is still guaranteed.

Consider the case where the intermediate state and the control input costs are equal to zero, and the terminal cost exists. However, all the constraints of the optimization problem are the same. This simplification can be considered as a one-step greedy strategy that drives the robot to move to its goal position as soon as possible. With this simplification, the moving object should direct towards a point in the convex polygon borders closest to its goal position [5].

Algorithm 2 outlines the basic AGA procedure. The input parameters are the number of robots N, the magnitude of the safety radius r, the magnitude of movement m in each step, the magnitude of deadlock tolerance δ, the magnitude of movement ϵ to break deadlock, and the number of previous positions ω to evaluate if a deadlock situation exists.

Algorithm 2: Analytical Geometric

```
   Data: N, r, m, δ, ϵ, ω, robots
 1 if not Collision Free Configuration then
 2 |   Change the initial or final configuration;
 3 |   break;
 4 end
 5 while not current positions = final positions do
 6 |   Generate Voronoi Diagram;
 7 |   Generate BVC;
 8 |   for agent in robots list do
 9 |   |   if not Final position inside BVC then
10 |   |   |   Get the closest point inside BVC;
11 |   |   |   Check deadlock;
12 |   |   |   cycle = 1;
13 |   |   |   while cycle < ω and not deadlock do
14 |   |   |   |   if deadlock then
15 |   |   |   |   |   Apply right-hand heuristic;
16 |   |   |   |   end
17 |   |   |   |   Increment cycle in 1;
18 |   |   |   end
19 |   |   else
20 |   |   |   The closest point is the final position;
21 |   |   end
22 |   |   Move the robot to its closest point;
23 |   end
24 end
```

The explanation of Collision Free Configuration and generation of the closest point to final in a cell mentioned in the procedure is below.

– **Collision Free Configuration.** For the group of N robots with the same safety radius r, *A collision free configuration* is one where the distance between positions of robot p_i and robot p_j satisfies:

$$\|p_i - p_j\| \geq 2r, \forall i, j \in 1, 2, \cdots, N, i \neq j. \tag{8}$$

– **BVC Closest Point.** Let $\mathcal{V} = (\epsilon, e)$ represents a convex polygon in ^{2}, where ϵ is the set of edges and e is the set of vertices. For any point $g \in^2$, the closest point $g^* \in \mathcal{V}$ to g is either g itself, or on an edge ϵ^* of $\mathcal{V}$, or is a vertex e^* of $\mathcal{V}$.

3.6 Optimal Reciprocal Collision Avoidance Algorithm

The task is for each robot A to independently (and new simultaneously) select a new velocity v_A^{new} for itself such that all robots are guaranteed to be collision-free for at least a preset amount of time r when they would continue to move at their new velocity. As a secondary objective, the robots should select their new velocity as close as possible to their preferred velocity. The robots cannot communicate with each other and can only use observations of the current position and velocity of the other robot. However, each robot may assume that the other robots use the same strategy to select a new velocity. Note that this problem cannot be solved using central coordination, as the robot itself only knows the preferred velocity of each robot [10].

In this work, we use the RVO2 library as the ORCA algorithm implementation [11]. It is an open-source implementation and has a simple API for third-party applications. In this way, the user specifies static obstacles, agents, and the preferred velocities of the agents. The simulation is performed step-by-step via a simple call to the library. Thus, the simulation is fully accessible and manipulable during runtime. Furthermore, the algorithm ensures that each agent exhibits no oscillatory behaviors.

3.7 Analysis Method

We defined the ways to study the model proposed in this subsection. First, a series of experiments evaluate and compare the performance of the model proposed in Sect. 4.2. These experiments were designed with specific research intentions:

1. We are executing pseudo-random spatial configurations of the agents to check the consistency in the construction of the Voronoi Diagram.
2. Next, suitable values for parameters in AGA and QP-Based RHC Algorithm appear.
3. Finally, the proposed algorithms and ORCA are compared using the best performing AGA and QP-Based RHC Algorithm.

We defined a set of measures for testing the algorithms. These measures helped us understand the performance of algorithm in terms of duration and calculation of optimum values. The performance measures are the following.

- **Execution Time (ET)** is a measure of execution duration in units of time. It is the time from starting movement until all robots reach the final positions. The execution time is represented as follows:

$$ET = t_f - t_i. \tag{9}$$

  Thus, ET is the total execution time, t_f is the final time, and t_i is the initial time.
- **Steps Number (ST)** is a measure of duration in terms of iterations. It is the number of iterations until all robots reach the final position. This measure depends on the algorithm design, mainly of the movement magnitude.
- **Effectiveness in Distance Traveled (ED)** measures how close the path traveled is to the shortest distance, understood as a straight line from the starting point to the end. This effectiveness is represented as:

$$ED = \frac{\sum_{i=1}^{N} a_i}{\sum_{i=1}^{N} b_i}. \tag{10}$$

  In this equation, N is the total number of robots, a_i represents the shortest distance for robot i from its started position to the final position, and b_i is the total distance traveled by robot i. This measure is only used for AGA parameters analysis.
- **Sum Cost (SC)** represents the sum of the means of all the previous measures for the selection of the best AGA parameter value. The SC follows the next model:

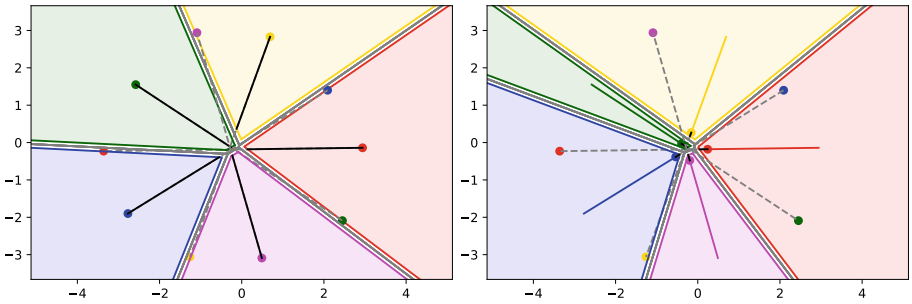
$$\min \left\{ \frac{ET_i}{\sum_{j=1}^{k} ET_j} + \frac{ST_i}{\sum_{j=1}^{k} ST_j} + \frac{1 - ED_i}{\sum_{j=1}^{k} (1 - ED_j)} \right\}, \tag{11}$$

  where k represents the number of values that one of the parameters can take and $i \in \{1, \cdots, k\}$.

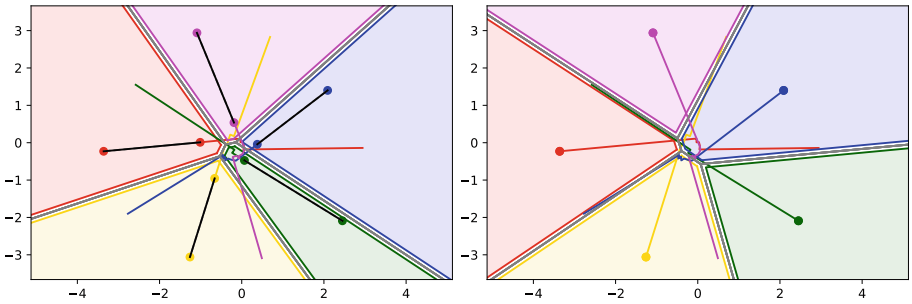
4 Results and Discussion

4.1 Construction of Voronoi Diagram

Figure 2 shows one experiment with all the corresponding geometric structures. The Voronoi Diagram is drawn with thick gray lines. The dashed gray lines are the straight lines from the current positions of agents to their goal positions. Also, the executed trajectories, BVC, and goal positions have the same color as the robot, and the thick dark lines are the planned paths from our algorithm for each robot in its cell. We can see in Fig. 2a an initial configuration with five robots and a safety radius equal to 0.3. In Fig. 2b, we can see the robots in the middle of the execution. In Fig. 2c, we can see all robots direct to their goal position. Finally, Fig. 2d shows the final state of the system.



(a) Initial configuration (b) Robots at step 27

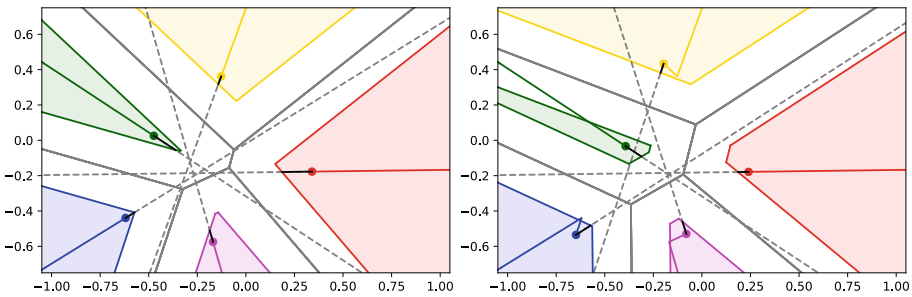


(c) Robots at step 43 (d) Robots at step 72

Fig. 2. Visualization of AGA execution with the respective BVC generation

4.2 Deadlock

The blocking situation was addressed satisfactorily in all the experiments. In Fig. 3a, we can see blue, cyan, and magenta robots in deadlock situation. Next, we can see in Fig. 3b the next state after dealing with deadlock using the right-hand rule heuristic.



(a) Yellow, blue and magenta robots in deadlock. (b) Positions after using the right-hand rule heuristic.

Fig. 3. Deadlock situation. (Color figure online)

4.3 Evaluation of Algorithms

We work with AGA to test geometric structures, deadlock situations, parameters performance and have a basis for the optimal proposal. The algorithm parameters are the same presented in Subsect. 3.5. The default values of each parameter are shown in Table 1. Finally, the number of experiments to evaluate each parameter value was equal to 30. In Fig. 4, we can see the increase of the average execution time along with the number of robots.

Table 1. Default parameters values in AGA evaluation

Parameter	N	r	m	δ	ϵ	ω
Value	5	0.1	0.1	0.05	0.1	1

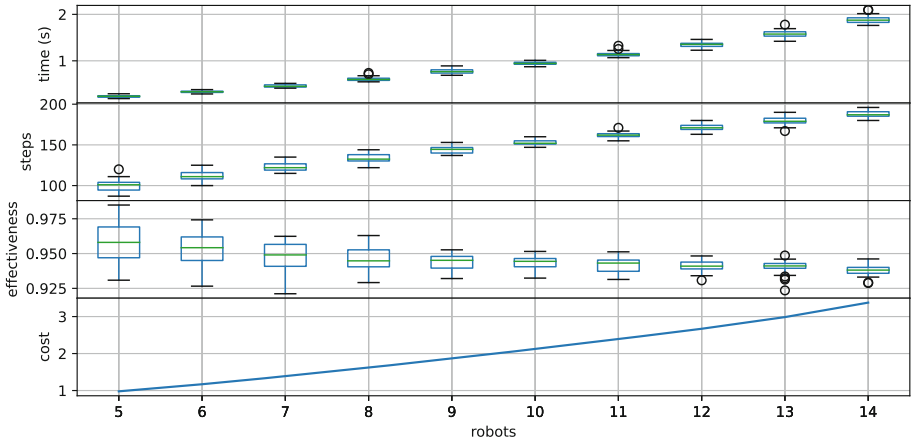


Fig. 4. Time in seconds, number of steps and effectiveness vs number of robots in AGA

The results of the QP-based RHC algorithm are similar to AGA performance since there is a growing trend of the execution time and instability in the step patterns and effectiveness up to a particular value. In the case of steps, the results stabilize from value six. In the effectiveness case, the results tend to be the same from value five onwards. Figure 5 shows the number of system steps according to the number of receding horizons steps in solver.

Figure 6 shows the performance of the ORCA Algorithm in comparison with AGA. In this way, both the time and the number of steps of ORCA are less than AGA for a range of values from five to fourteen robots. However, for the number of steps, AGA comes close to ORCA.

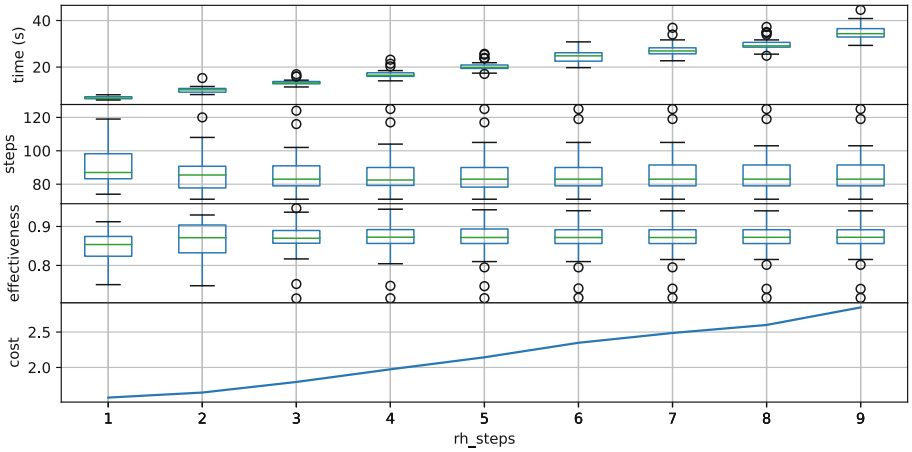


Fig. 5. Time in seconds, number of steps and effectiveness vs number of in QP-based RHC algorithm.

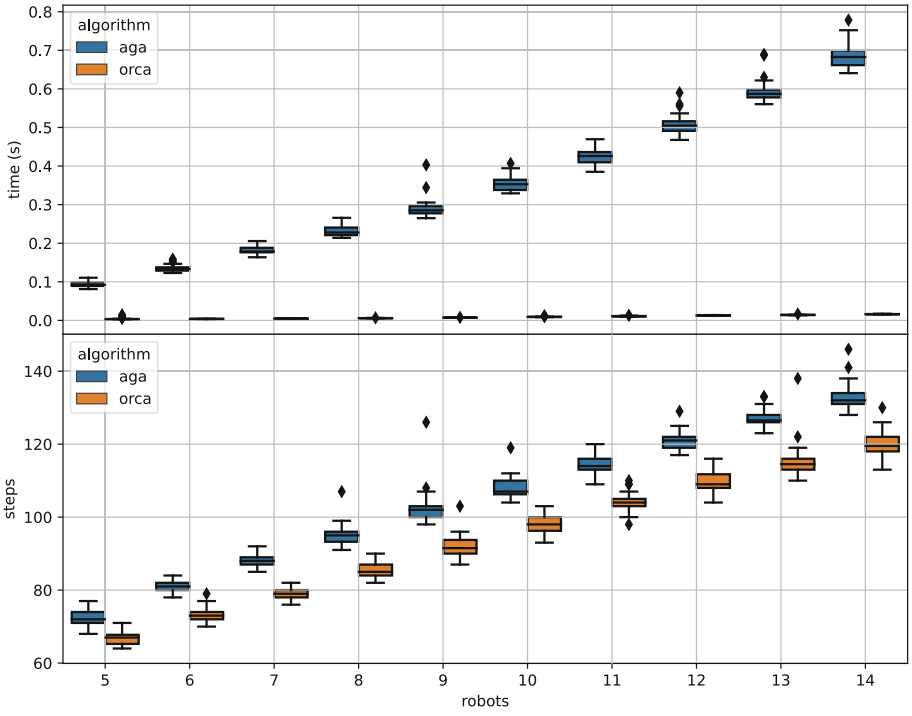


Fig. 6. ORCA and AGA performance in time and number of steps vs number of robots.

4.4 Implementation

The experiments were executed in a computer with a Intel ® Core ™ i7-10750H CPU @ 2.60 GHz processor with 6 CPU Cores, 12 Logical processors, 15.5 GB RAM, 64-bit system type running on Ubuntu 20.04.4 Operating System. The following repository has our algorithm implementation, performance, and experimental setup for reproducibility purposes: https://github.com/pepeleduin/Collision-Avoidance-Algorithms-in-2D-using-Voronoi-Diagrams.

5 Conclusions

We conclude that the Analytical Geometric Algorithm successfully addressed and solved the execution time limitations and the number of steps to converge towards the solution. On the other hand, the Receding Horizon Control algorithm based on Quadratic Programming requires more execution time. We did not always get optimal results working with heuristics, but we obtained good-enough solutions that are close to the performance of the Optimal Reciprocal Collision Avoidance algorithm. However, the algorithm produces promising results with an accuracy of around 95%.

5.1 Future Work

We consider using techniques such as High-Performance Computing to experiment with more significant problems for future work. The nature of the agent system and the environment can be modified to analyze the behavior of the algorithms. In this way, concurrency could be applied to decentralize the system and expand the dimension of the environment. In addition, the simulation visualization can be improved to better appreciate the behavior of the agents. Finally, other solvers could find better optimization of the solution of the problem.

References

1. Zaccone, R., Martelli, A.: A collision avoidance algorithm for ship guidance applications. J. Marine Eng. Technol. **19**(sup1), 62–75 (2020)
2. Hamid, U., Saito, Y., Zamzuri, H., Rahman, M., Raksincharoensak, P.: A review on threat assessment, path planning and path tracking strategies for collision avoidance systems of autonomous vehicles. Int. J. Veh. Autonomous Syst. **14**(2), 134–169 (2018)
3. Huang, S., Teo, R., Tan, K.: Collision avoidance of multi unmanned aerial vehicles: A review. Ann. Rev. Control **48**, 147–164 (2019)
4. Dahl, J., de Campos, G., Olsson, C., Fredriksson, J.: Collision avoidance: A literature review on threat-assessment techniques. IEEE Trans. Intell. Veh. **4**(1), 101–113 (2018)
5. Zhou, D., Wang, Z., Bandyopadhyay, S., Schwager, M.: Fast, on- line collision avoidance for dynamic vehicles using buffered voronoi cells. IEEE Robotics Autom. Lett. **2**(2), 1047–1054 (2017)

6. Xuan P., Lesser, V.: Multi-agent policies: from centralized ones to decentralized ones. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1098–1105 (2002)
7. Camacho, E., Alba, C.: Model predictive control. Springer Science & Business Media (2013)
8. Mattingley, J., Wang, Y., Boyd, S.: Receding horizon control. IEEE Control Syst. Mag. **31**(3), 52–65 (2011)
9. Diamond, S., Boyd, S.: CVXPY: a Python-embedded modeling language for convex optimization. J. Mach. Learn. Res. **17**(83), 1–5 (2016)
10. Carbone, C., Ciniglio, U., Corraro, F., Luongo, S.: A novel 3d geometric algorithm for aircraft autonomous collision avoidance. In: Proceedings of the 45th IEEE Conference on Decision and Control, pp. 1580–1585. IEEE (2006)
11. Van den Berg, J., Lin, M., Manocha, D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In: 2008 IEEE International Conference on Robotics and Automation, pp. 1928–1935. IEEE (2008)