Osmanu Rafi

Reciprocal Velocity Obstacle(RVO)

vs

Optimal Reciprocal Collision Avoidance(ORCA)

Supervisor: Boury Fries

Coach: Verspecht Marijn

Graduation Work 2023-2024

Digital Arts and Entertainment

Howest.be

CONTENTS

ABSTRACT & KEY WORDS

This paper presents a comprehensive comparative analysis of two leading collision avoidance algorithms in multi-agent environments: RVO (Reciprocal Velocity Obstacles) and ORCA (Optimal Reciprocal Collision Avoidance). This study is grounded in a systematic evaluation across three distinct scenarios, each chosen to illuminate key performance metrics: computational efficiency, intended path deviation, and estimated time of arrival to the goal location. By employing a range of simulation environments, the paper not only contrasts the operational strengths and limitations of RVO and ORCA but also provides insights into their suitability for different application contexts. The findings contribute to a deeper understanding of the trade-offs inherent in multi-agent navigation algorithms and offer guidance for practitioners in selecting the most appropriate algorithm based on specific operational requirements.

PREFACE

My fascination with collision avoidance algorithms in multi-agent environments was sparked during a previous assignment when I attempted to implement these complex systems but found myself struggling. This initial challenge stayed with me, and a year and a half later, armed with greater knowledge and a renewed sense of determination, I decided to revisit this challenge. My interest lies not only in the intricate mathematics and physics that underpin these algorithms but also in translating these theoretical concepts into clear, accessible code. I recall my early forays into this field, where I often encountered code that was convoluted and intimidating, dampening my enthusiasm to even begin. This project, therefore, was not just a technical pursuit; it was an opportunity to refine my skills as a gameplay programmer, reacquaint myself with mathematical and physical principles, and contribute to making this complex domain more approachable for those who, like me, are fascinated by the theory but daunted by the practical implementation. This paper is a testament to that journey and a resource I wish I had when I first embarked on it.

LIST OF FIGURES

INTRODUCTION

The challenge of collision avoidance in multi-agent environments is a multifaceted problem with diverse solutions, finding applications not only in gaming but also in robotics, traffic simulation, and more. The essence of this research is rooted in algorithmic solutions to this problem, specifically through the comparative analysis of two algorithms based on the principle of Velocity Obstacles (VO).

The VO principle employs a simple yet effective strategy: it involves generating a set of relative velocities that could lead to collisions within a certain timeframe and subsequently selecting a velocity outside this set to avoid collisions. This study delves into two prominent implementations of this principle: RVO (Reciprocal Velocity Obstacles) and ORCA (Optimal Reciprocal Collision Avoidance). Both algorithms have seen practical applications in gaming, sparking my curiosity about their comparative effectiveness in managing agent navigation and collision avoidance in densely populated, dynamic environments.

What prompted this research was the observation of how different groups of agents navigate complex environments without frequent collisions. Such observations raise intriguing questions about the underlying mechanics of these algorithms and their performance in real-world scenarios.

That being said, this study is guided by a central research question: *What are the comparative strengths and weaknesses of the RVO and ORCA algorithms in terms of computational efficiency, adaptability to complex scenarios, and overall performance in facilitating agent navigation in gaming environments?* This question aims to dissect and analyze the core attributes of these algorithms, providing a comparison of their functionalities and effectiveness in the specific context of game development.

In pursuit of this question, the study is driven by a hypothesis formulated through preliminary observations and theoretical considerations. The hypothesis states that *RVO is more computationally efficient in gaming scenarios, whereas ORCA demonstrates greater adaptability to complex gaming environments*. This efficiency is hypothesized to stem from RVO's algorithmic structure, which may allow for faster computations. On the other hand, ORCA is hypothesized to demonstrate superior adaptability in complex gaming environments. This adaptability is attributed to ORCA's advanced collision prediction and avoidance mechanisms, which may offer more nuanced navigation capabilities in dynamically changing scenarios.

The verification of this hypothesis forms the goal of the research, aiming to provide insights and practical comparisons between RVO and ORCA.

LITERATURE STUDY / THEORETICAL FRAMEWORK

1. VELOCITY OBSTACLE(VO)

As mentioned in the introduction, both RVO and ORCA are built upon the same principle, which we call a *velocity obstacle* (VO). To explain this topic, I will refer to entities that avoid collisions as “agents”. A VO is a collection that each agent creates for itself, consisting of relative velocities that would lead to collision within a specific timeframe. An agent has as many VOs as there are other agents it needs to consider, with each VO comprising relative velocities to one specific other agent.

To be able to create a VO, all agents must have knowledge of three things about their neighboring agents: their current velocity, radius and position. Calculating the VO is relatively straightforward, but it’s important to understand a few core concepts, so the process of creating it becomes simpler.

1.1 RELATIVE VELOCITY

First of all, a velocity is a vector quantity representing both the direction and speed of an object, where the direction indicates the object's course and the speed determines the length of the vector line. A relative velocity is the difference between two agents their velocities as observed from one of the agents.

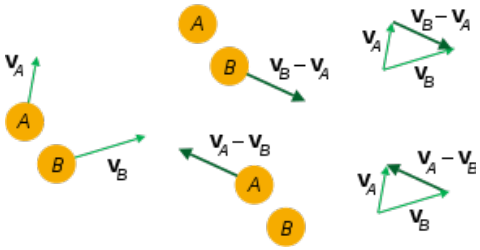


Figure 1: Relative velocity representation

1.2 SPEED EQUATION

Speed in physics is defined as the covered distance over time (Speed = Distance/Time), we will reformulate this formula to help us find the relative velocities that would lead to collision in a certain *timeframe*. It is this timeframe that we have to find in order to predict future trajectories. Time is equal to the distance over speed (Time = Distance/ Speed).

1.3 MINKOWSKI SUM

Imagine you have two shapes, Shape A and Shape B. To find the Minkowski sum of these two shapes, you would take every point in Shape A and add it to every point in Shape B. The resulting set of points forms a new shape, which is the Minkowski sum of A and B.

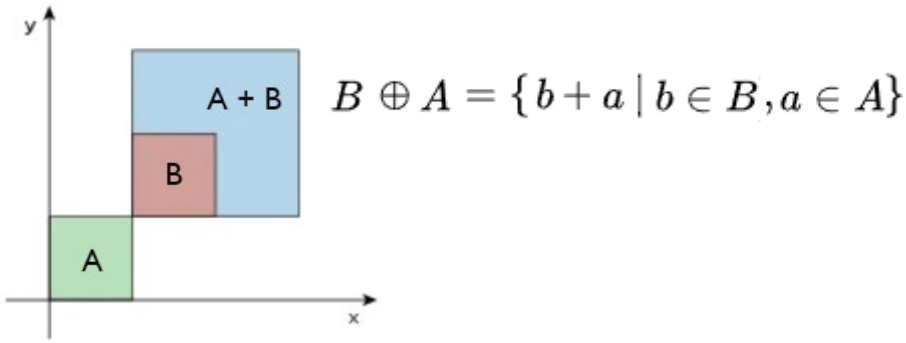


Figure 2: Minkowski sum of two shapes A and B

1.4 MAKING A VELOCITY OBSTACLE

First, we need to determine a velocity to check for potential collisions. The selection method is flexible, but a simple approach is to examine velocities within a specified range. To determine the direction of this velocity, use an angle ranging from -180 to 180 degrees, with a parameterized increment. Convert this angle into a valid velocity vector by multiplying the directional vector (cos(angle), sin(angle)) by the current speed of our agent.

Next, calculate the relative velocity for the velocity we are examining relative to the current velocity of the agent we want to avoid. Use this relative velocity in our modified speed equation to determine the time to collision (TTC) between two agents, calculated as Time = Distance / Speed. (Speed = Speed from the velocity) Since we know the positions of our agents, we can easily calculate the distance between them using the distance formula.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figure 3: Distance formula

With the TTC determined, calculate the trajectory of the agent by adding the agent's current position to the product of the TTC and the velocity we are checking (Pos + TTC * vel). This calculation predicts the future position of our agent if it were to travel at that specific velocity. Perform a similar calculation for the agent we wish to avoid, using the same TTC but with that agent's current velocity to determine its future position.

Finally, use the Minkowski sum of the radii of the two agents, centered at the future position of the agent we want to avoid, to check for intersection. The rationale for using the Minkowski sum is to simplify our agent to a point, which makes intersection checks easier. We use the point representing the position as a reference to check if the position lies within the boundaries of that new radius, centered at the future position of the agent we are trying to avoid. If the predicted future position of our agent lies within that boundary, we add the relative velocity (between the velocity we checked and the current velocity of the agent we are avoiding) to our velocity obstacle. Performing this process for the entire range will yield the velocity obstacle for that specific timestep.

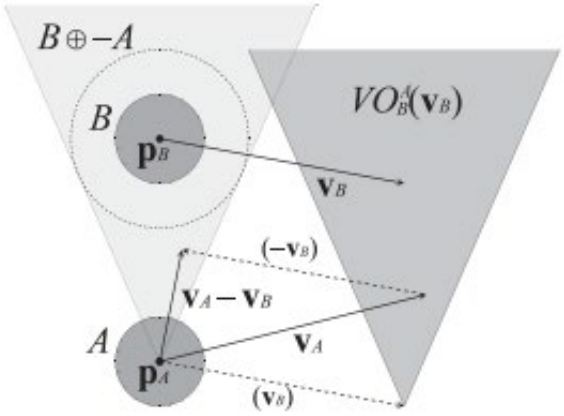


Figure 4: Geometric representation of a velocity obstacle between two agents A and B

Using this principle alone is sufficient to avoid collisions, but it has some notable drawbacks. The main one is the fact that agents often oscillate when trying to avoid collisions. This has a logical explanation: imagine if you calculate the Velocity Obstacle and choose a velocity outside this VO. In the next frame, your agent will likely want to direct its velocity back towards its intended direction. This action effectively puts the velocity back into the VO, causing the process to repeat until the agent has passed the obstacle it is trying to avoid. This problem becomes more prominent as the density of the crowd of agents increases.

2. RECIPROCAL VELOCITY OBSTACLE(RVO)

While VO might not be the most optimal for collision avoidance in multi agent environment, it's concept is very valuable for further usage. The solution for the oscillation problem is something that the reciprocal velocity obstacle(RVO) concept handles. RVO doesn't differentiate all that much from the original VO, It's a simple extension of VO in its basic application.

"*The basic idea is simple: instead of choosing a new velocity for each agent that is outside the other agent's velocity obstacle, we choose a new velocity that is the average of its current velocity and a velocity that lies outside the other agent's velocity obstacle.*" [van den Berg, Lin, & Manocha, "Reciprocal Velocity Obstacles for real-time multi-agent navigation", 2008, p. 2]

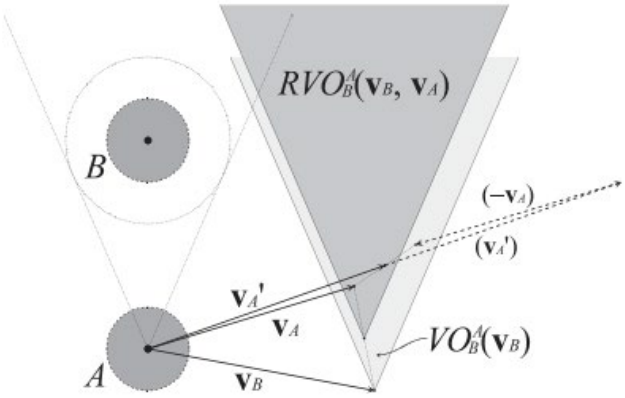


Figure 5: Geometric representation of RVO between two agents A and B

By averaging the preferred velocity with the collision-avoiding velocity, RVO effectively reduces the tendency of an agent to oscillate back and forth between collision avoidance and moving towards its goal. This is because the chosen velocity is a compromise between moving towards the goal and avoiding collisions in a manner that anticipates similar behavior from other agents. They are less likely to make drastic changes in their velocities, thus reducing oscillation. Each agent calculates its new velocity based on the assumption that other agents are also actively adjusting their velocities to avoid a collision. This shared responsibility for collision avoidance shows in the averaging process because the agent is leaving part of the avoidance to its neighbors.

This is an effective way of collision avoidance in multi-agent environments, but it still poses a problem that originates from the Velocity Obstacle (VO). This problem is known as the reciprocal dance, where both agents can't decide on which side to pass each other. This occurs because they would keep making the same decision on how to avoid a collision, potentially leading to a deadlock or, more likely, a collision. RVO doesn't resolve this issue but is still a valid choice for collision avoidance, considering that this type of scenario is not very common. It mostly occurs in head-on collisions, and usually, agents are able to find a different strategy than their neighbor before colliding.

3. OPTIMAL RECIPROCAL COLLISION AVOIDANCE(ORCA)

Finally, we will take a look at how ORCA really works. As mentioned, ORCA also utilizes the Velocity Obstacle method. However, there is a significant difference in how the velocity outside the VO is chosen. When working with ORCA, our focus is not on the velocities we cannot take, but rather on those we are permitted to take. Among these permissible velocities, we find the one that is closest to the optimal velocity. The criteria for determining the optimal velocity will be discussed later in this chapter.

3.1 ORCA LINE

To determine which velocities are permitted, we construct what is called an ORCA line. This line divides the velocity space into two sections: a safe side and a potentially hazardous side. The safe side represents a half-plane where all velocities are guaranteed to be collision-free. Conversely, the other side represents velocities that could lead to a collision.

The construction of this line involves several steps. First, you need to calculate your relative velocity to the agent you wish to avoid. Using this velocity, check if a collision is likely within a specific timeframe (this is similar to checking for an intersection as you would with a simple VO). If a potential collision is confirmed, you must then find the velocity to the closest point on the boundary of your VO, starting from your relative velocity (let's denote this velocity as '**u**'). This represents the minimum velocity required to avoid a collision. Once you have this velocity, the only remaining step to construct the ORCA line is to add half of **u**, the minimal avoidance velocity, to your current velocity. The line perpendicular to this resultant velocity is known as the ORCA line, with the direction of **u** indicating the safe side of the line. It's important to note that we only add half of **u**, the minimal velocity necessary to avoid a collision, leaving a portion of the avoidance responsibility to our neighboring agents.

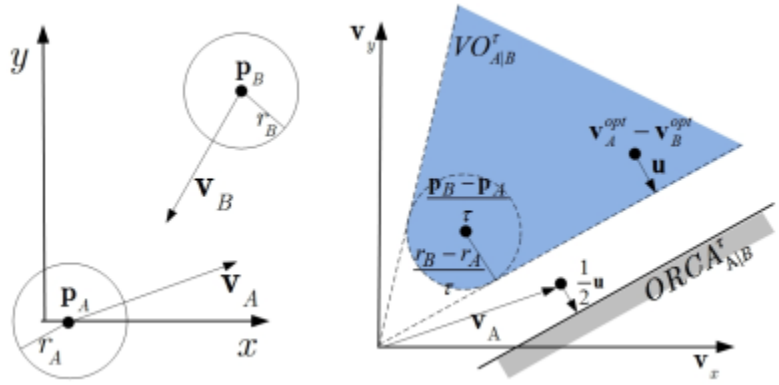


Figure 6: Geometric representation of ORCA between two agents A and B

Now that we know the permitted velocities, we still need to find a velocity that is as close as possible to our optimal velocity. There are a few options when it comes to choosing our optimal velocity, but a credible approach is to set it equal to our current velocity. This ensures we deviate as little as possible from our originally intended path. Finding a velocity closest to our optimal becomes increasingly complex as the density of our crowd increases. We will have to find a velocity that satisfies all constraints, meaning the velocity should be on the safe side of all ORCA lines created for agents of potential collisions, while at the same time finding the one closest to our optimal in that intersection of ORCA lines.

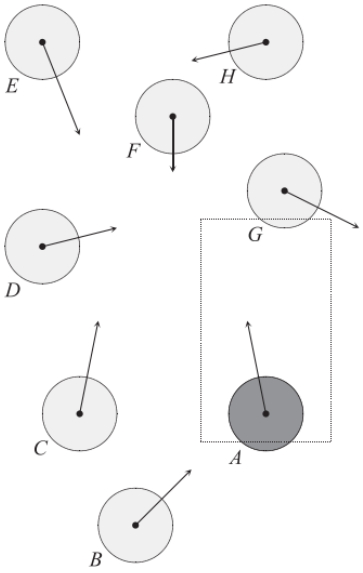


Figure 7: Scenario of multi agent environment from perspective of agent A

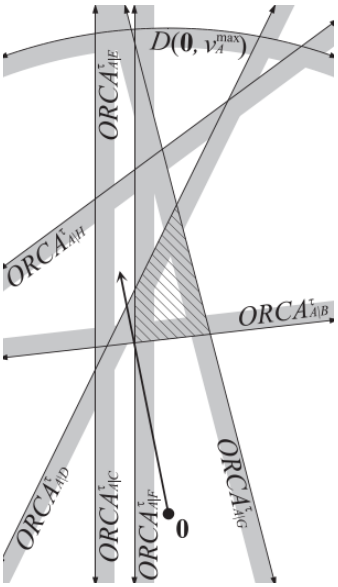


Figure 8: Geometric representation of created ORCA lines for agent A highlighting an intersection zone that satifies all constraints

3.2 LINEAR PROGRAMMING(LP)

The best method for finding this velocity is through the use of a linear program (LP). Given that we have a series of linear constraints, we can establish a valid LP to determine the velocity closest to the optimal while satisfying our constraints. However, before setting up this LP, it's essential to understand what a linear program is. Delving into all the details of LPs is beyond the scope of this paper, so I will focus on highlighting the most important concepts that need to be understood for our use case.

3.2.1 OBJECTIVE FUNCTION & VARIABLES

The goal of a linear program is to either minimize or maximize a specific target, known as the objective function. This function is constructed from the sum of variables, each multiplied by its respective coefficient, and is linear in nature. It might look something like this:

A * x1 + B * x2

The coefficients A and B represent the objectives of the function, which will be maximized or minimized by the variables x1 and x2. These variables, in turn, represent the values necessary to achieve this maximum or minimum result.

In our case, our objective is to determine the optimal velocity. Since we are working with 2D velocities, this means that the A coefficient corresponds to our optimal velocity in the x-direction (vx), and the B coefficient corresponds to our optimal velocity in the y-direction (vy).

3.2.2 LINEAIR CONSTRAINTS

When calculating the ORCA lines, we mentioned that the line divides the velocity space into two parts: a safe side and a dangerous side. We can only choose velocities from the safe side, which is represented as a half-plane. This is, in fact, a linear constraint. If we represent our variables x1 and x2 on a 2D plane, with x1 corresponding to the x-axis and x2 to the y-axis, and then add the linear constraints, it would look something like this:

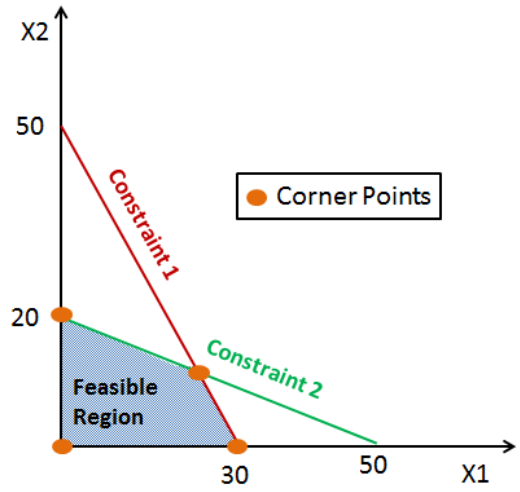


Figure 9: Linear constraints on a 2D plane

As previously discussed, our ORCA lines intersect, creating a region of valid choices for our velocity. Ultimately, you want to incorporate all the constraints imposed by the ORCA lines to ensure that the chosen velocity satisfies all of them

3.2.3 SIMPLEX METHOD

The LP is now ready to be solved, and this can optionally be done using the simplex method. The simplex method is an algorithm used to solve linear programming problems and can be conveniently explained geometrically.

The method starts at an initial vertex of the feasible region, which is the area satisfying all constraints of the LP problem. Usually, this initial point is a solution where all variables are set to zero.

In LP, the feasible region is often a polygon (in two dimensions), formed by the constraints. The corners or vertices of this region are potential solutions. The goal of LP is to maximize or minimize a linear function. The simplex method moves along the edges of the feasible region to find the optimal value for this function.

It iteratively visits vertices of the feasible region. At each vertex, it evaluates whether moving to an adjacent vertex improves the value of the objective function. If a better adjacent vertex is found, the method 'pivots' to this new vertex, updating the variable values to reflect the new point. This process continues until no adjacent vertex offers a better solution. At that point, the current vertex represents the optimal solution to the LP problem.

Once the LP is solved, we should retrieve the variables x1 and x2 to define our new optimal velocity, with x1 representing the new x-direction and speed, and x2 representing the new y-direction and speed.

Understanding the workings of your program is important, but in most cases, you will use a library to handle the simplex method. However, setting it up is always your responsibility, so knowing the meaning of each step is crucial.

RESEARCH

1. EXPERIMENT SETUP

1.1 UNREAL ENGINE 5.03

To bring these algorithms to life, the chosen environment is Unreal Engine. The primary reason for this choice is the desire to expand knowledge in using C++ within a large engine. Given prior experience, the engine is familiar and provides the necessary tools to set up scenarios for collision avoidance quite conveniently. The blueprinting aspect of Unreal Engine is also very useful for quick tests and meeting higher-level requirements.

1.2 UCHARACTER

The agents that have been referred to throughout this paper will be represented by a custom class derived from *UCharacter*. This is a class in Unreal Engine that has predefined functionality for movement, known as '*CharacterMovement*'. This feature is convenient in our workflow for retrieving and setting the velocity necessary for avoidance, and it also allows us to enforce a maximum speed that all agents must adhere to. To set up a destination for these agents, you can use a simple node in the blueprint called 'AI Move To'. Combined with a navigation mesh, this will guide your agent to the designated position. This method has been used throughout the entire experiment to set up all the scenarios.

1.3 PRE-EXPERIMENT CONSIDERATIONS

As mentioned in the abstract, the experiment aims to compare two algorithms based on three performance metrics: computational efficiency, intended path deviation and time of arrival at the goal location.

The implemented algorithms are basic representations of the concepts. This approach ensured that the implementation of both RVO and ORCA remained within scope. Further optimization is definitely possible and would likely lead to more efficient, collision-free navigation.

2. SCENARIO 1: 1 TO 1 HEAD ON COLLISION

2.1 SCENARIO SETUP

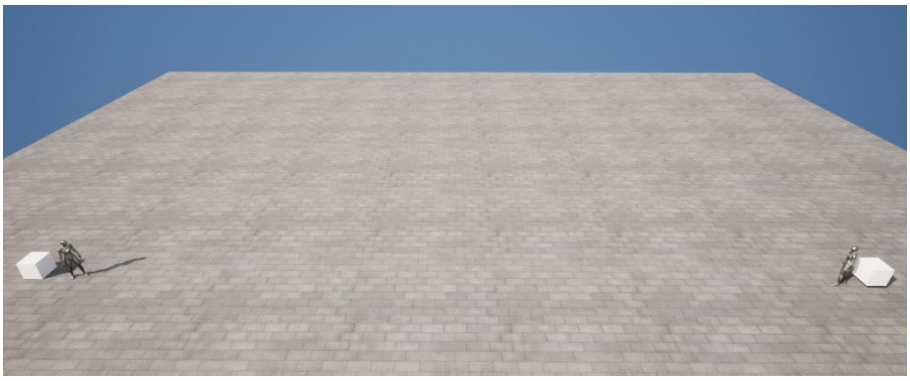


Figure 10: 1 to 1 head on collision setup

In this scenario, two agents are positioned facing each other, with each agent's goal being the position directly behind the other, leading to a potential head-on collision. The importance of this setup is to evaluate how well both algorithms perform in a simple, low-density scenario and to isolate the earliest possible differences in their avoidance.

2.2 RESULTS

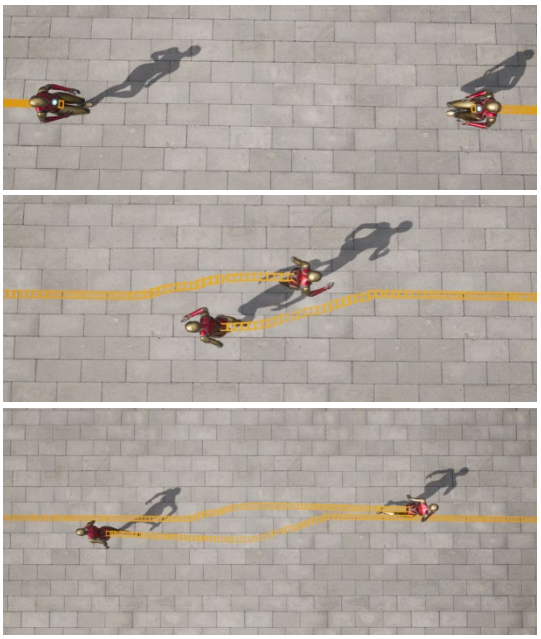


Figure 12: 1 to 1 collision avoidance *RVO*

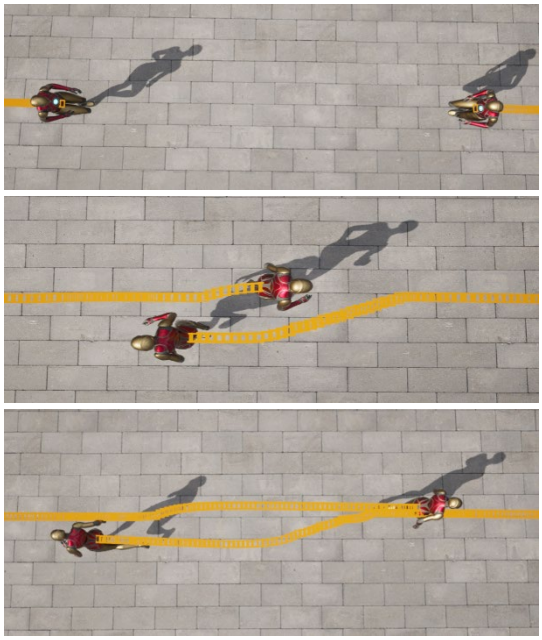


Figure 11: 1 to 1 collision avoidance *ORCA*

In a simple scenario like this, there is almost no noticeable difference between them, they perform equally well in avoiding collisions.

2.3 COMPUTATIONAL EFFICIENCY

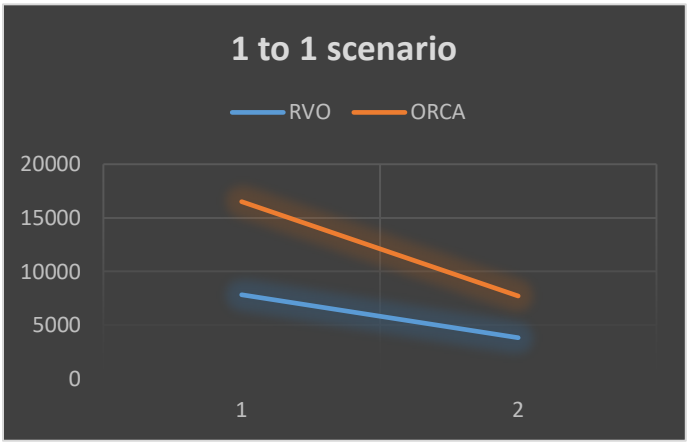
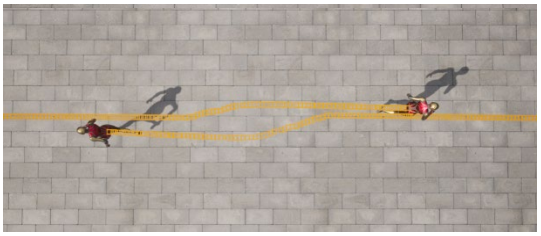


Figure 13: 1 to 1 scenario computation speed graph

This average is taken from running both algorithms a hundred times for each agent, with the y-axis representing the time it took to compute (expressed in nanoseconds) and the x-axis representing the number of agents. We observe that ORCA consistently requires more time to compute than the other algorithm, which aligns with our expectations from the implementation of both algorithms. This is because they both utilize VO, and RVO differs less from the original VO, whereas ORCA incorporates a linear program in addition to calculations for the linear constraints. In a low-density environment, however, this difference in computation speed is not as critical for performance, but it could potentially become a significant issue in really high-density, complex environments

2.4 INTENDED PATH DEVIATION

RVO



ORCA

Figure 14: 1 to 1 path deviation RVO vs ORCA

As discussed in our theoretical framework, both agents avoid collisions by partially delegating the responsibility of avoidance to the other agent. This is why they both exhibit a similar deviation from their paths. The deviation is minimal in both cases. It is evident that in low-density environments, both algorithms perform comparably in terms of path deviation

2.5 TIME OF ARRIVAL AT THE GOAL LOCATION

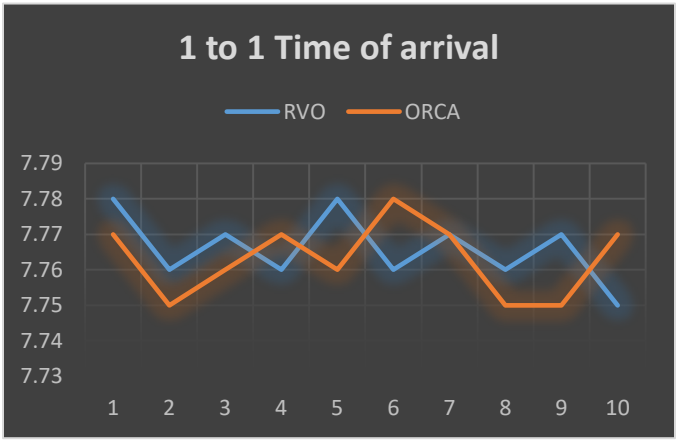


Figure 15: 1 to 1 time of arrival

The time of arrival at the goal location for both algorithms is nearly identical. This suggests that for simple collision avoidance scenarios, such as a one-to-one head-on collision, both RVO and ORCA effectively avoid collisions while guiding the agents to their goal locations within a similar timeframe. For reference the y-axis represents the time of arrival in seconds, while the x-axis represents the amount of iterations.

3. SCENARIO 2: CIRCLE SETUP

3.1 SCENARIO SETUP

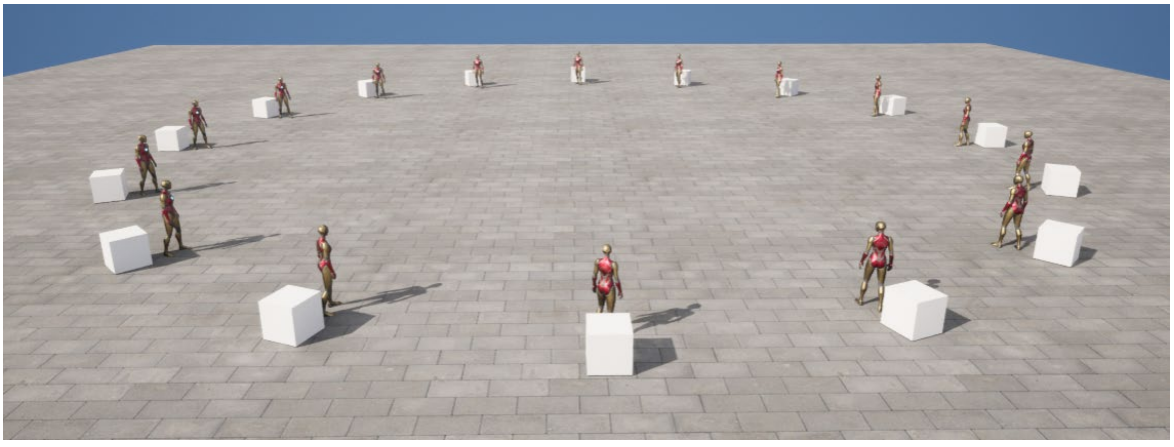


Figure 16: Circle collision setup

Agents are uniformly positioned around a circle, with each agent placed at intervals of 22.5 degrees. This arrangement ensures that one agent is located in every 22.5-degree segment of the circle's circumference, providing even distribution and spacing around the entire circle. The goal location for each agent is indicated by a

cube positioned 180 degrees across the circle, directly opposite the agent. We aim to increase the complexity of the environment by making it denser, giving us more insight on the differences and strategies of both algorithms.

3.2 RESULTS

RVO

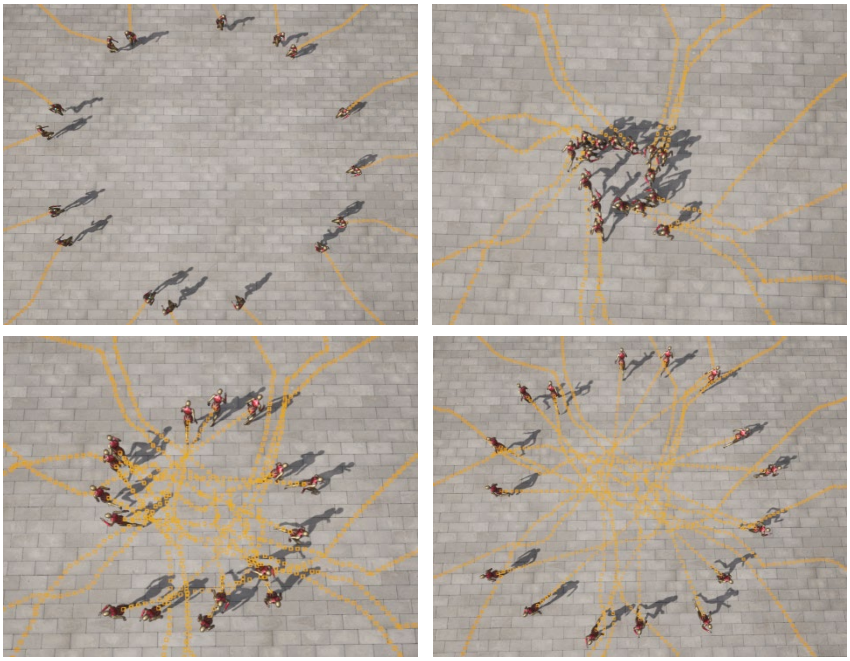


Figure 17: Circle setup *RVO*

ORCA

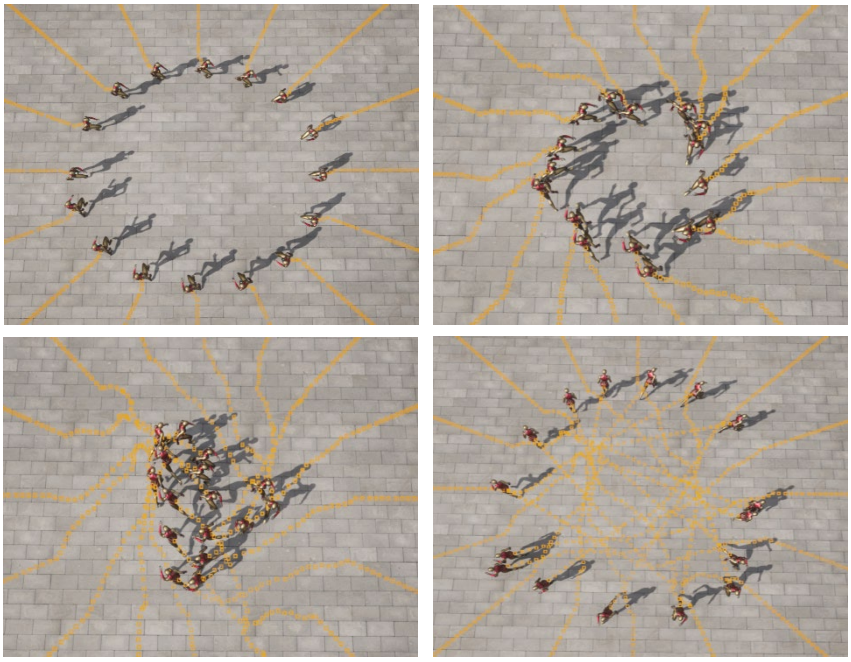


Figure 18: Circle setup *ORCA*

The movement is similar, but ORCA appears to move in a more circular manner than RVO, though the difference is slight.

3.3 COMPUTATIONAL EFFICIENCY

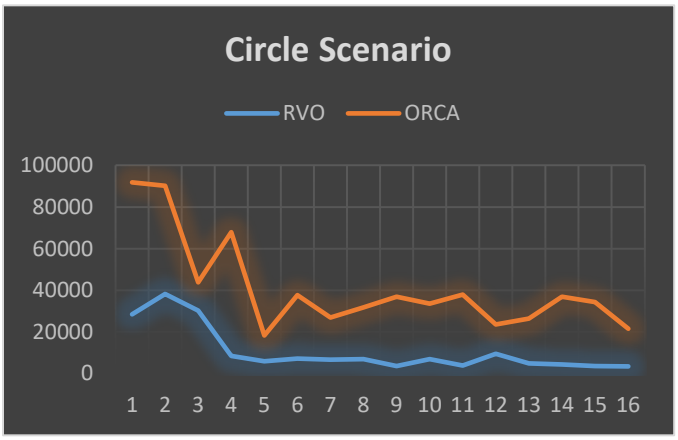


Figure 19: Circle scenario computation speed graph

ORCA consistently requires more time to compute than RVO. In this scenario, generating a considerable number of ORCA lines adds constraints to the linear program, potentially slowing down the computing process. However, it is observed that in this scenario, both ORCA and RVO experience a similar drop in FPS, with ORCA experiencing an additional drop of 3-4 fps compared to RVO. This means that both algorithms are becoming quite resource-intensive, but ORCA is slightly more so.

3.4 INTENDED PATH DEVIATION

RVO

ORCA

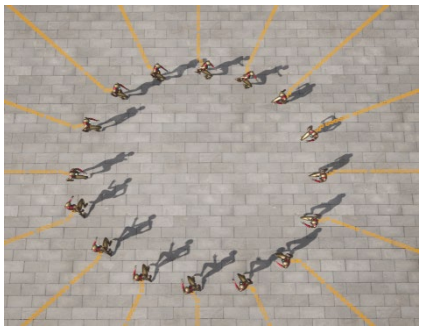


Figure 20: Path deviation initial difference circle scenario

Both algorithms generally adhere to their intended paths, but there are some noticeable differences. RVO begins to deviate from the intended path earlier to avoid collisions, while ORCA remains on the intended path for quite some time before starting to avoid collisions. This difference is mainly due to the cutoff time set for considering collision avoidance. With ORCA, this time can be set shorter and still effectively avoid collisions. However, with RVO, if the time is set too short, the agents are unable to avoid collisions and consequently collide in the center of the circle.

RVO **ORCA**

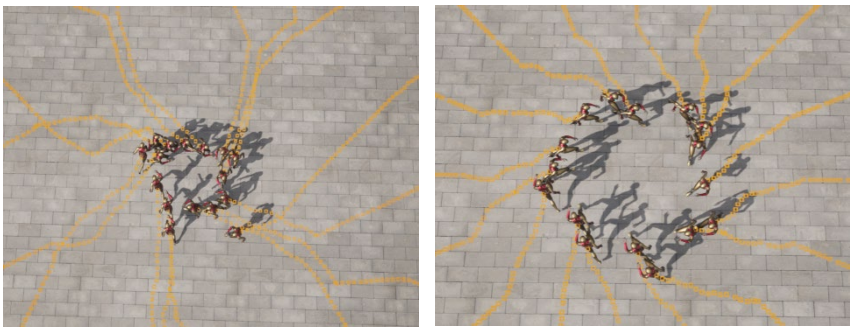


Figure 21: Path deviation difference collision phase circle setup

During the actual collision avoidance phase, their movements and deviations from the intended path are similar. However, ORCA exhibits a more coherent circular movement compared to RVO.

3.5 TIME OF ARRIVAL AT THE GOAL LOCATION

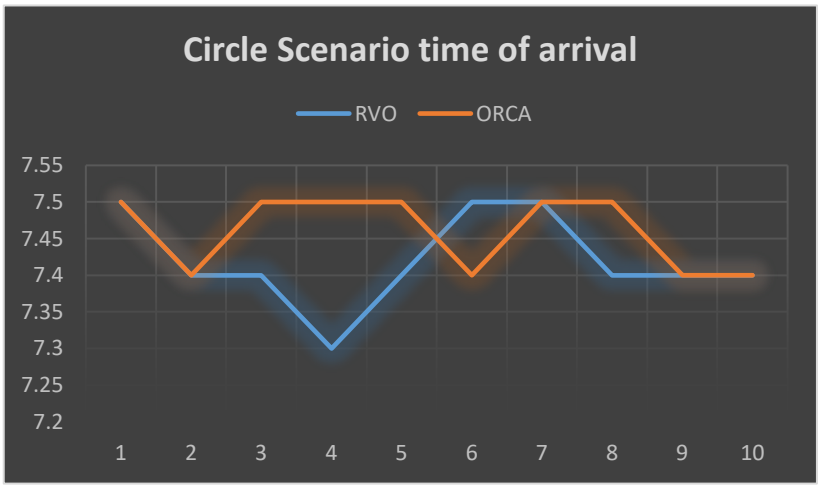


Figure 22: Circle scenario time of arrival

Even in a more populated and complex scenario like this, both algorithms demonstrate almost equal capability in guiding the agents to their goal locations. This is highlighted by their nearly identical times of arrival. Furthermore, this serves as a good indicator of their similar efficiency in avoiding collisions within semi-dense crowds.

4. SCENARIO 3: CROSS INTERSECTION SETUP

4.1 SCENARIO SETUP

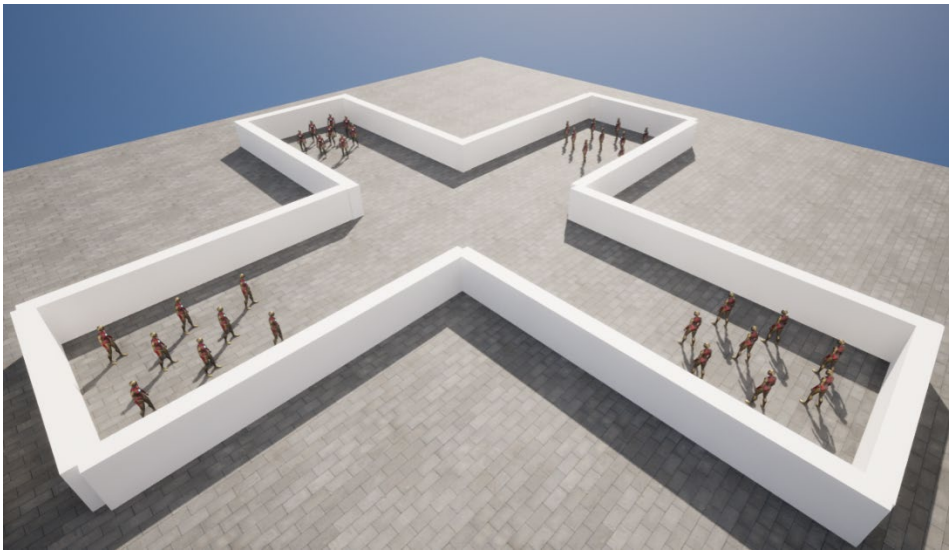


Figure 23: Cross intersection collision setup

This is the final scenario we are testing, now with double the number of agents, all confined within a closed space. This setup prevents them from straying far to avoid collisions. The goal destination for each group of agents is the wall on the opposite side from where they are facing. Our aim is to simulate a denser and more realistic scenario to further inspect and compare the two algorithms when faced with increased complexity.

4.2 RESULTS

RVO

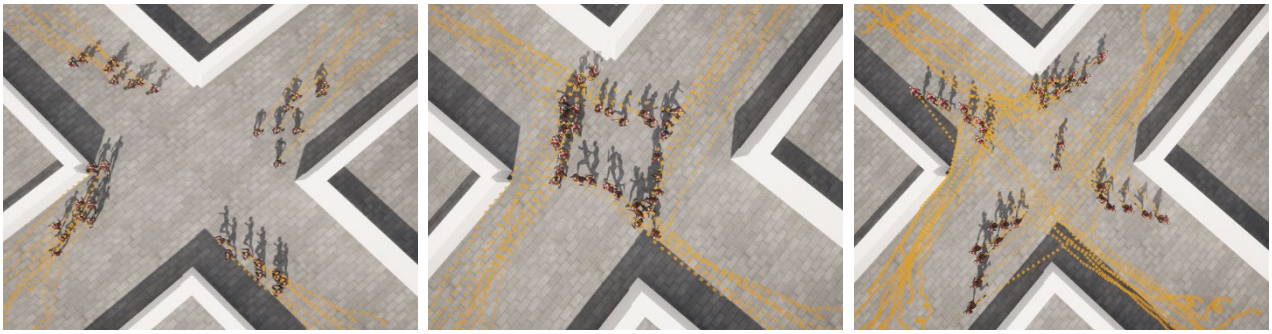


Figure 24: Cross intersection setup *RVO*

ORCA

Figure 25: Cross intersection setup *ORCA*

With both algorithms, the agents seem to assume a left position in expectation of a collision so that they can all pass on that side. Although neither algorithm is completely collision-free in this confined space, they do help the agents navigate through each other without causing a deadlock, significant loss of pace for any of the agents, or any oscillation.

4.3 COMPUTATIONAL EFFICIENCY

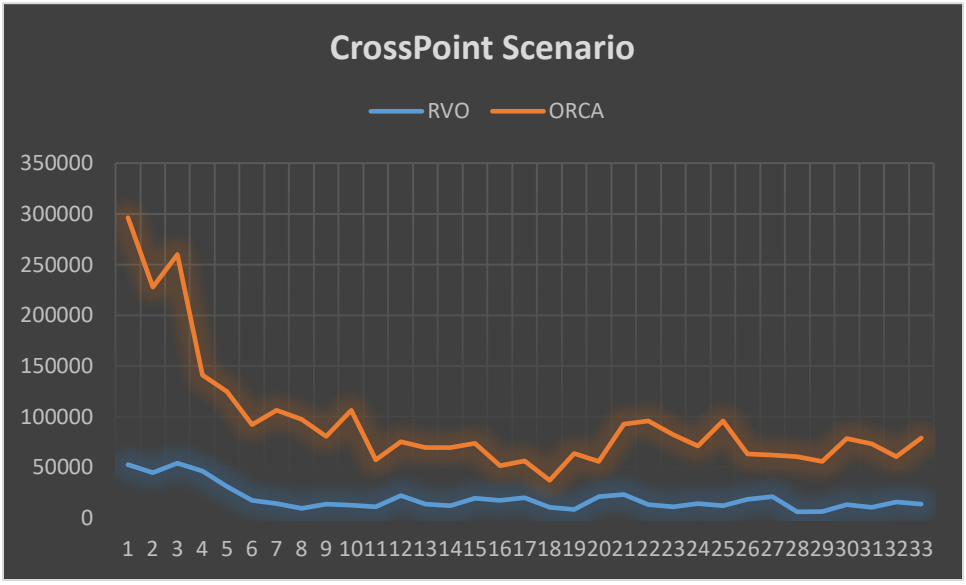


Figure 26: Cross section scenario computation speed graph

As expected, ORCA consistently takes longer to compute than RVO. There is also a big jump in computation speed in general for both algorithms; this is due to the doubled number of agents present. This is also noticeable in the FPS, as it drops even lower than before, with RVO having a slightly higher FPS. This indicates that both algorithms most likely need more optimization, starting from the velocity obstacle that they both use, since the FPS is low in general.

4.4 INTENDED PATH DEVIATION

RVO **ORCA**

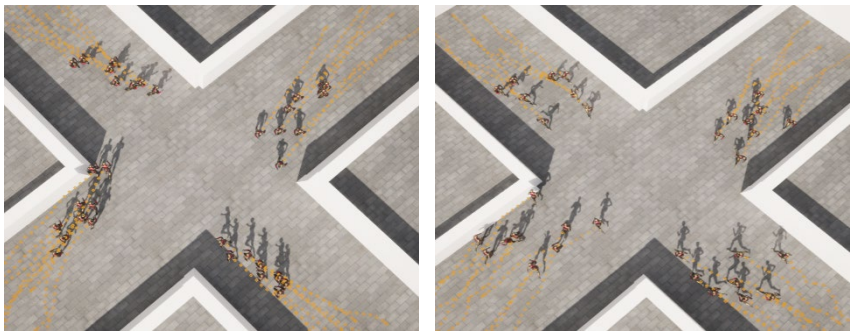


Figure 27: : Path deviation difference initial cross intersection setup

During the approach of a collision, they both assume a left-leaning position; however, ORCA stays more neutral and does not lean as significantly.

RVO **ORCA**

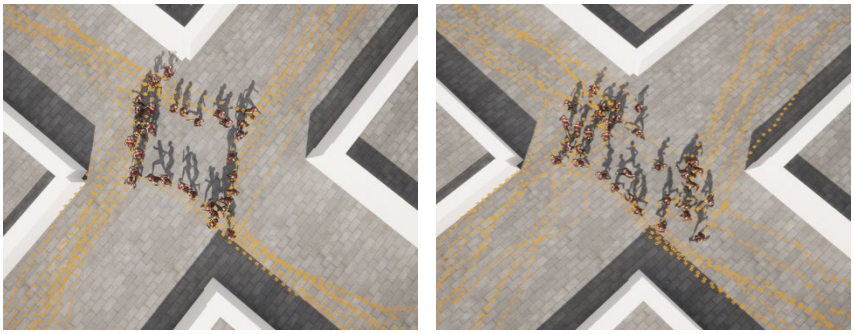


Figure 28: Path deviation difference collision phase cross intersection setup

We can see that their deviation from their intended path is similar; however, their methods of avoiding collision are somewhat different. RVO prompts the agent to assume a left position early on, enabling them to proceed straight ahead more effectively during collision avoidance. In contrast, ORCA causes the agent to curve more, as it addresses collision avoidance slightly later.

4.5 TIME OF ARRIVAL AT THE GOAL LOCATION

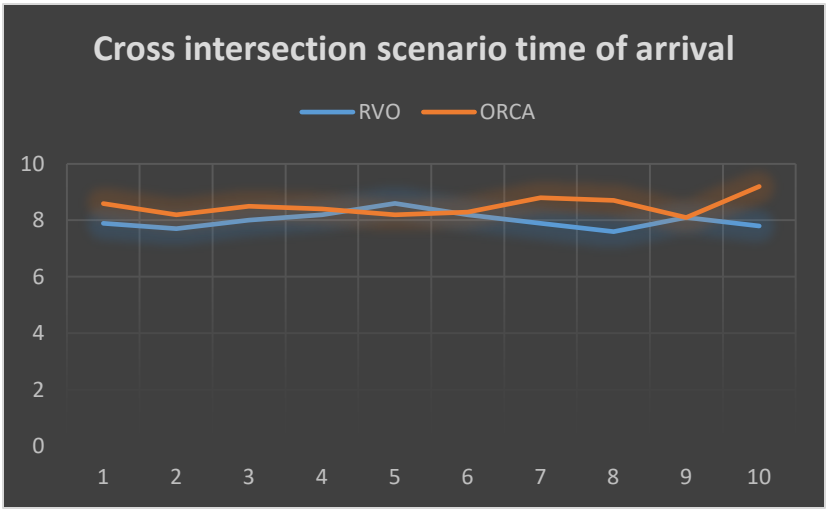


Figure 29: Cross intersection scenario time of arrival

Very similar in both simple, low-density scenarios and more complex, high-density ones, they both facilitate an equal capacity in navigating groups of agents to their destination. This is evident considering they have had similar times of arrival in every scenario.

DISCUSSION

There were aspects of the results that aligned with expectations, as well as some that did not. From the perspective of collision avoidance strategies, both algorithms implemented demonstrated their applicability across all experiments. One notable aspect was the reciprocity in avoidance, as outlined in the theoretical framework. Agents do not completely avoid collisions; rather, they partially avoid them, expecting neighboring agents to do likewise.

However, I was surprised to observe that both algorithms exhibited similar efficiency in avoidance. Given ORCA's complexity, I had anticipated that this complexity would translate into a more optimal solution. It appears that RVO is on par with ORCA, although this might be attributable to an unoptimized implementation of ORCA.

The movement pattern in ORCA's collision avoidance is rationalized by its line constraints. These constraints prevent agents from choosing unsafe paths, leading them to consistently opt for safe directions. This results in the ORCA lines adjusting repeatedly in response to agents moving towards their destination, ultimately creating the illusion of agents forming a circle in most cases.

It is also important to note a primary drawback of RVO, the reciprocal dance, which was expected to be uncommon and indeed proved to be so in our experiments. This phenomenon was not noticeably observed, but it might have contributed to some collisions. The oscillation, which was expected to disappear with the implementation of RVO, stayed true to its promise, as such behavior was no longer displayed.

CONCLUSION

Overall, given the results we have obtained, it seems fair to conclude that RVO is often sufficient as a valid collision avoidance approach. ORCA not only demonstrates slower computation times but also presents a complexity in implementation. Given the similarities in the efficiency of collision avoidance, RVO appears to be the safer choice. However, in large crowd simulations, ORCA might have an advantage if it is fully optimized. Since my aim was to compare the algorithms without prior bias, I had to rely on a basic implementation, focusing my efforts on gaining a thorough understanding of the topic.

This project showcases an implementation grounded in a solid understanding of the subject, serving as a credible resource for academics and others interested in collision avoidance algorithms. The tests and results provide insight into what can be expected from basic implementations of RVO and ORCA in a major engine like Unreal. I also want to note that this research can serve as a reference for what to expect in terms of movement and collision-free guarantees from these algorithms.

In conclusion, my results align partially with my initial hypothesis. RVO, as demonstrated by the results, is indeed computationally more efficient. However ORCA did not show a noticeable superiority in adaptability to complex scenarios. Since both algorithms did not significantly differ in terms of arrival time, this indicates that they have comparable abilities in avoiding collisions without causing deadlocks or significant losses in pace.

FUTURE WORK

As previously mentioned, this project utilizes a rather basic implementation of the algorithm. Further optimization, not only adhering to existing standards but also exploring innovative solutions to the problem as a whole, is something I believe will be pursued in the near future. This could enhance the optimality and ease of implementation of these algorithms. Developing a method to guarantee collision-free navigation in less controlled environments is a significant challenge, one that could potentially lead to the discovery of new and innovative collision avoidance strategies.

CRITICAL REFLECTION

Of course, I would have liked to implement fully optimized versions of these algorithms, but overall, I'm quite satisfied with the outcome of this project. My previous attempts to engage with these algorithms were hindered by their daunting implementation and the highly technical nature of the available information on math and physics related to this topic. Overcoming such barriers is immensely gratifying and reaffirms my capabilities as a developer. The process of conducting this experiment and composing this paper has been academically enriching. It taught me that complex subjects can be made accessible when approached step by step.

I leave this paper with a renewed interest in math and physics. I have learned and revisited numerous topics and now eagerly anticipate my future endeavors as a programmer in the gaming industry, or more broadly, as a creator.

REFERENCES

[1] J. A. Douthwaite, S. Zhao, and L. S. Mihaylova, “Velocity Obstacle Approaches for Multi-Agent Collision Avoidance,” Un. Sys., vol. 07, no. 01, pp. 55–64, Jan. 2019, doi: 10.1142/S2301385019400065.

[2] “Velocity obstacle,” Wikipedia. Jan. 21, 2023. Accessed: Oct. 25, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Velocity_obstacle&oldid=1134896483

[3] The Art of Linear Programming, (Jul. 04, 2023). Accessed: Oct. 25, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=E72DWgKP_1Y

[4] J. Van Den Berg, Ming Lin, and D. Manocha, “Reciprocal Velocity Obstacles for real-time multi-agent navigation,” in 2008 IEEE International Conference on Robotics and Automation, Pasadena, CA, USA: IEEE, May 2008, pp. 1928–1935. doi: 10.1109/ROBOT.2008.4543489.

[5] Reciprocal Velocity Obstacles for Real-time Multi-agent Navigation, (Nov. 12, 2009). Accessed: Jan. 14, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=1Fn3Mz6f5xA

[6] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal n-Body Collision Avoidance,” in Robotics Research, vol. 70, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds., in Springer Tracts in Advanced Robotics, vol. 70. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–19. doi: 10.1007/978-3-642-19457-3_1.

[7] S. Bonagiri, “Paper Review: Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation,” Medium. Accessed: Oct. 25, 2023. [Online]. Available: https://medium.com/@suraj2596/paper-review-reciprocal-velocity-obstacles-for-real-time-multi-agent-navigation-aaf6adbedefd

[8] Optimal Reciprocal Collision Avoidance, (Mar. 03, 2015). Accessed: Jan. 14, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=zmTgj4SKftc

[9] Local Navigation with ORCA, (May 10, 2019). Accessed: Jan. 14, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=0pSkpUtLH74

[10] H5 P3 Velocity Obstacles: Collision Cone and Velocity Obstacles, (Jan. 23, 2021). Accessed: Jan. 14, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=CeGQ6sLpZD0

[11] “Forced-Based Anticipatory Collision Avoidance in Crowd Simulations.” Accessed: Jan. 14, 2024. [Online]. Available: https://www.gdcvault.com/play/1021986/Forced-Based-Anticipatory-Collision-Avoidance

[12] “Decentralized Navigation of Multiple Robots Based on ORCA and Model Predictive Control.” Accessed: Jan. 14, 2024. [Online]. Available: https://www.sysu-hcp.net/projects/robotics/30.html

[13] The COIN-OR Foundation, “coin-or/Clp: COIN-OR Linear Programming Solver.” Accessed: Jan. 14, 2024. [Online]. Available: https://github.com/coin-or/Clp

[14] S. Guy et al., “ClearPath: Highly parallel collision avoidance for multi-agent simulation,” Aug. 2009, pp. 177–187. doi: 10.1145/1599470.1599494.

ACKNOWLEDGEMENTS

I would like to use this section to express my gratitude to my family, who provided me with invaluable mental and financial support throughout this journey. Additionally, I extend my heartfelt thanks to all my friends from the DAE program, with whom I shared countless hardships and celebrated numerous victories.

APPENDICES

Experiment Results: https://youtu.be/Br5ssHow3UE

GitHub link RVOvsORCA project source code: https://github.com/RafiOsmanu/ORCAvsRVO/tree/main