Journal of Intelligent & Robotic Systems (2021) 101: 58
https://doi.org/10.1007/s10846-021-01326-0

DVG+A* and RRT Path-Planners: A Comparison in a Highly Dynamic Environment

Leonardo da Silva Costa[1] · **Flavio Tonidandel**[2]

Received: 22 September 2020 / Accepted: 20 January 2021 / Published online: 2 March 2021

Abstract

This work provides a deeper comparison between two path planning algorithms, the Dynamic Visibility Graph A Star (DVG+A*) and Rapidly–exploring Random Trees (RRT), when applied in a high dimension and dynamic environment, which is the RoboCup Small Size League. The algorithms were compared under two different perspectives. In the first analysis, the algorithms were evaluated according to its computational time, path length and path safety in a static environment. Afterwards, they were evaluated regarding the accumulated computational time, number of recalculated paths, total navigation time and number of collisions in a dynamic environment. The static environment results have shown that the DVG+A* has a better overall performance than RRT, except for the path safety, however, some ideas on how to improve this were discussed. In the dynamic environment the algorithms performed similarly and with a high number of collisions during the experiments. Thus, showing the importance of using an obstacle avoidance algorithm combined with the path planner. In conclusion, the results obtained showed that both algorithms aren't suitable for highly dynamic and cluttered environments, however, due how sparse the obstacles are in the SSL, they can still be used with some care. Regarding static environments, the DVG+A* has shown the best results.

Keywords Path planning · A star · Dynamic visibility graph · Rapidly-exploring random trees · RoboCup small-size league

1 Introduction

Path planning algorithms are a common topic of interest in the robotics scenario, which is understandable, due to the fact that generating a global optimal path is something considered a NP-Hard problem [19]. Regarding this complexity, the Small Size League (SSL) category provides a great environment to develop and compare different path planning and obstacle avoidance techniques. The SSL is a category of the RoboCup Competition which aims to develop multi-agent intelligent systems. In order to reach this goal, the robots must play a soccer match while attending to a set of rules similar to human soccer. Therefore, the teams must develop many types of skills for the robots, e.g., motion control, path planning and obstacle avoidance, object tracking (mainly for the ball), multi-agent coordination and others.

The reason for the SSL being a challenging environment, especially for path planning, is due to the fact that it is the category that has one of the most dynamic interactions between robots of the whole RoboCup competition, where the robots can reach a velocity up to more than 2.5 m/s and the ball can reach a maximum velocity of 6.5 m/s. Therefore, there has always been a lot of research in the path planning area for this league [7, 20, 23, 34].

The robots used in the competition are omnidirectional robots, as illustrated in Fig. 1 for instance. Due to the mechanical construction of its wheels, these robots are capable of moving independently in all of its axis (X, Y, and θ). This feature allows some simplifications in the path planning task, which, makes it a little easier when compared to planning a path for a non-holonomic robot, such as a car-like robot.

This work was supported by University Center of FEI.

✉ Leonardo da Silva Costa
unieleocosta@fei.edu.br

Flavio Tonidandel
flaviot@fei.edu.br

[1] Department of Control and Automation Engineering, University Center of FEI, São Bernardo do Campo, SP, Brazil

[2] Department of Computer Science, University Center of FEI, São Bernardo do Campo, SP, Brazil

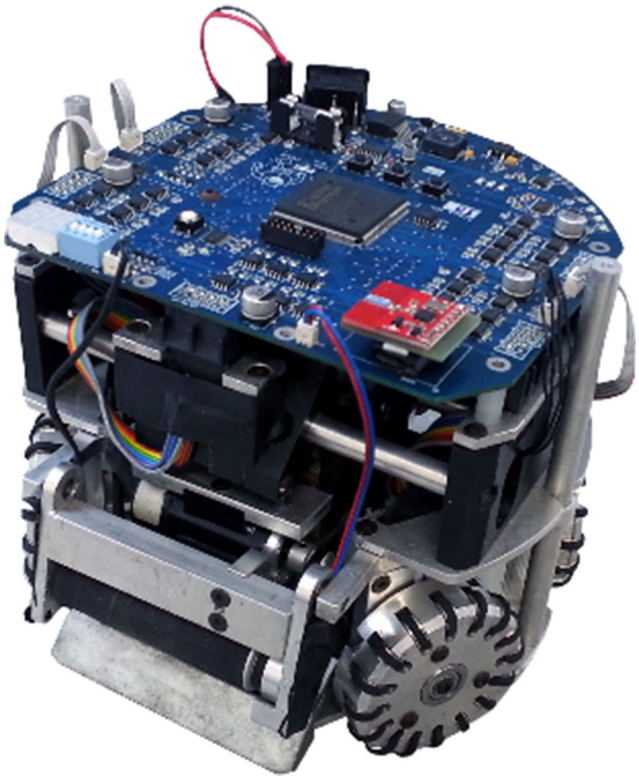

Fig. 1 Robot used in the SSL

In the most competitive division of the league there are 8 robots in each team, playing in a 12×9 m field. Beyond that, due to the fast paced game play, the target point of a robot may change even before the robot has reached it. Furthermore, as seen on [22], there are rules which aim to penalize the collisions between robots, thus, making even more necessary the use of a good path planning algorithm. Given those characteristics of the league, a good path planning algorithm has to offer the best trade-off between computational time, path length, path safety, navigation time and number of collisions.

In a more general description, the challenges involved in the SSL environment for a path planning algorithm are: a dynamic environment with relatively sparse obstacles, where, most of them have a practically unpredictable behavior, there are also several constraints, e.g., minimum computational time and path length, least amount of collisions possible, or even none collisions at all, a reasonable path smoothness, to allow a fast and smooth movement of the robots, and, some kind of coordination between the paths of the same team, so that the robots don't interfere with the current play. All of these characteristics make the path planning task very complex, due to not having a precise prediction of the position of the opponent robots in a time window greater than one second, and the possibility that their trajectory may change in a short period of time.

This paper is an extended version of [24], which compares both Dynamic Visibility Graph A Star (DVG+A*) and Rapidly-exploring Random Trees (RRT) [15] algorithms in the SSL environment. However, its main analysis doesn't get into many details when considering a dynamic environment. Therefore, this paper shall provide an updated analysis of the performance of both algorithms, but, it will also analyze them regarding its obstacle avoidance capabilities, which is a key feature on a environment as dynamic as the SSL. At last, more details on the use of the DVG are provided.

2 Related Work

The use of path planning algorithms is a common topic of research in the league, almost every year there is at least one publication on ways to improve the performance of an already known algorithm or a completely new approach on solving this kind of problem. By analyzing the Team Description Papers (TDP) and Extended TDP (ETDP) of some teams in the league, [20, 21] shows that the RRT is a common object of discussion and many teams have been using custom path planning algorithms. Even outside the SSL environment, RRT based approaches seem to provide great results when applied to high dimensional or dynamic environments [28–30]. Therefore, the RRT algorithm was chosen to be compared against the DVG+A*.

In [34] it is possible to see that the team uses an RRT variant called Bidirectional Multi-Bridge ERRT. In summary, this algorithm is an improvement of the RRT-Connect [14]. This algorithm was a product of a whole PhD thesis, more details can be seen on [8].

There's also an algorithm described in [23] as an alternative to RRT. It recursively tries to connect the goal and start point. When a collision is detected it inserts an intermediate point to avoid it. This algorithm works similarly as the RRT with the exception that it possess a strong goal bias, which guides the algorithm exploration, and consequently, lowers the computational time. Although the RRT algorithm can also be biased towards the goal state, this must be done with care [26]. According to [15], by inserting a strong goal bias in the RRT algorithm, it may impact the algorithm completeness, specially in local minimums, similarly to what happens in potential field based algorithms.

In the ETDP of the MRL team [21] there's an example of a completely customized path planning algorithm. It consists of using several curves, such as polynomial, sines, cosines and straight lines, to create a path. The algorithm checks if the path has any collisions with the objects and then an optimization procedure is run to determine the best path.

Outside of the SSL environment there's also some studies investigating the performance of A* and RRT based algorithms. In [5] there's a comparison using A* and the RRT-Connect, which is a variation of the original RRT. In this study, the author analyzed the path length of the algorithms and it showed that the path length from the A* was shorter than the one generated by RRT-Connect. The RRT-Connect is basically the original algorithm with a tree growing from the start and goal points.

Another study that compares the path length and computational time from A* and RRT is [25]. In this study the author compares the original A*, θ^*, AA*, RRT and dynamic domain bi directional RRT in many different situations, e.g., variable blocked cells, a maze and others. Although the A* was able to generate the shortest path in most of the situations, it was the one that had the highest computational time.

Another example of performance comparison between A* and RRT algorithms can be seen in [6], this study shows that the A* is capable of outperforming RRT in most of the scenarios proposed by the author, both in computational time and path length. Thus, showing that A* provides a better performance in comparison to RRT when in a low dimension environment, which in this case, is composed by 10 cells of 1 m^2. When comparing this study against [25], it can be seen that the major difference is the increased environment dimension, i.e., the grid size in the latter varies between 100×100, 500×500 and 1000×1000 cells, therefore, by having an environment with higher dimension, the A* algorithm takes more time to explore the environment and find a path.

The work seen on [19] does a very thorough comparison among two A* and RRT based algorithms, the Memory Efficient A* (MEA*) and Rapidly-exploring Random Trees Star-Adjustable Bounds (RRT*-AB). The main improvement of MEA* over A* is the reduced memory consumption by only adding the lowest cost neighbors to the open list. As for the RRT*-AB, its convergence rate is considerably improved by only sampling a region of the total search space. Also, just like RRT Star (RRT*) it is capable of achieving optimal paths as the number of iterations approach infinity. A more detailed description of these algorithms can be seen on [19].

Although most of the previous references are related to the SSL league, there are many other examples in the state of the art for path planning algorithms in dynamic environments, specially for robotic manipulators [30], unmanned aerial vehicles (UAV) [16] and others [2, 27].

The work seen on [30] proposes a new variation of the RRT algorithm, which is called Smoothly RRT (S-RRT). In this work, the author used a robotic manipulator and the tests evaluated the path planning algorithms both in static and dynamic environments. The proposed algorithm improves both convergence speed, and smoothness of the original RRT. The first is achieved by changing the extend step of the original algorithm, the author proposes that the algorithm must extend the tree towards the target position until it finds a collision, only then, the algorithm will randomly extend the tree. Also, in the random extension, there is a probability to extend specifically the node closest to the target position. As noted by the author, it is utterly important to keep at least some part of the randomness of the algorithm in order to maintain its completeness. At last, the second improvement was achieved by using three post processing steps in the path found. The first is to remove unnecessary nodes in the path, whilst not causing any collisions, then, it is evaluated if there aren't any sharp angles in the path (angles smaller than α_{min}), if there are, an additional node is added to the path in a way that the respective angle gets bigger. Finally, the last step is to smooth the angles in the path by using a K-order B spline. The results obtained by the author show that the S-RRT is capable of outperforming RRT and one of its variants both in computational time and path smoothness. Moreover, in the dynamic tests, the S-RRT was capable of successfully avoiding the dynamic obstacle. However, it is important to keep in mind that the proposed dynamic test is much more simple and predictable, when compared to the SSL environment. In the experiment made there were only one static and one dynamic obstacle, and the latter didn't interfere drastically in the path.

The work seen on [27] also seems quite interesting, first, the author brings into attention that the task of finding a path for a non-holonomic robot in a dynamic environment can be interpreted as a 4D problem $(x, y, \theta, time)$, therefore, making it extremely complex to find a path. Even though non-holonomic robots are not the case in the SSL, this approach can be valid for future works. To address this problem, the author proposes the use of adaptive dimensionality techniques (AD). The objetive of this technique is that, in the cases where one, or more, dimensions are not important, they can be ignored in the process of finding a solution. In this case, since there are some regions where the robot must move in a straight, the θ dimension can be ignored, and when there is practically no possibility that a collision might occur, then the time dimension can be ignored. The author compared this AD approach against a 4D A*, and the results show that the AD was much more efficient than 4D A*. But, in order to use the AD, the author makes the assumption that the trajectories of the obstacles are previously known, and that all obstacles move with constant speed, both of these are practically the opposite of what happens in the SSL. However, the need to consider the time dimension might be an important parameter in future path planning algorithms for the SSL environment. Furthermore, due to the holonomic properties of the SSL robots, the planning problem is

reduced to a 3D problem, involving only the x, y and time dimensions.

At last, [2] proposes a whole new algorithm for dynamic cluttered environments using Voronoi Diagrams (VD) and Computational Geometry Techniques (CGT). In this work, the author uses the position, velocity, acceleration and direction of the obstacles to determine if there is a risk of collision. The studied environment is composed of static and dynamic obstacles, which, the latter has a completely unpredictable behavior, i.e., its position, velocity, acceleration and direction are randomly generated, also, both obstacles, and robot, follow non-holonomic kinematics just like a car-robot. Regarding the collision avoidance, the author proposes many different rules that evaluate if there is a high, or a low, risk of collision, from which side of the path the collision might happen and the distance that the obstacle is from the controlled robot. Along with these rules, it is also defined which action should be taken in order to avoid the collision. Note that, if in any case, the obstacle gets closer than a critical distance, a local reactive collision avoidance algorithm is used. The steps of the path planning algorithm are the following: it calculates the roadmap using VD, then, A* is used to find a global path and cubic splines are used to smooth the result. After this, in the presence of a new obstacle, the algorithm checks which rule is applied in that condition, then, it creates a rectangular region that contains the robot and the obstacle, this region delimits the section of the path that needs to be replanned. These steps are repeated until the target position is reached. Although there aren't any results regarding the number of collisions that might have occurred, if any occurred, the results shown demonstrate that this approach is capable of successfully avoiding dynamic obstacles. The author compared the algorithm against both RRT and A*, and in the comparison, the proposed algorithm performed much better regarding the considered metrics. A few important points in this work are: the use of VD, which could be an interesting improvement over the DVG used in the present work, due to the fact that a VD tends to be as distant as possible from the obstacles, which is the opposite of what happens in the DVG. The second point is the use of pre determined rules that guide which action the algorithm should take to adjust its current path in order to avoid collisions. This is certainly more useful than simply replanning the whole path, which is the action taken by A* and RRT.

As seen previously, the literature has some good examples of comparisons and improvements using the A* and RRT families in dynamic, cluttered and/or unpredictable environments. However, there aren't many studies related to applying or comparing these algorithms on the SSL, or some environment with similar characteristics, such as, highly unpredictable and holonomic obstacles. The researches in the SSL community regarding the path planning problem are mostly focused on alternative algorithms or improvements to the already existing ones, there ain't many comparisons regarding what is a better choice for the SSL environment or what may be challenges of applying them in this environment. Therefore, the work presented in this paper will provide such comparison, and, a special attention will be given when it comes to using the studied algorithms in a environment with the dynamic characteristics of the SSL. The experiments made aimed to test the algorithms on a dynamic environment, where practically all of the obstacles are moving with the same velocity as the controlled robot and no information about them is used. This way, not only the computational performance parameters, such as, computational time and path length, are evaluated, but also the navigation capability of the algorithms in a constantly changing environment.

3 Dynamic Visibility Graph A Star – DVG+A*

When analyzing some recent research on the A* based path planners [6, 19], it is noticeable that in most cases the algorithm is used with a grid based state space to represent the environment. Although this approach can be applied in many domains, the grid maps require a trade-off between grid resolution and memory consumption, consequently, a high resolution grid also causes the search algorithm to run slower, due to the higher complexity. In the SSL environment it is important that the path generated yield a good precision, while also maintaining a low computational time. Therefore, a grid based representation might not be the best choice in this scenario.

There are some grid based A* variations which focus on reducing the memory consumption, such as the MEA* [19]. However, according to the author, due to using a grid map, the MEA* can only outperform sampling based algorithms, such as the RRT, in low dimension search spaces.

By doing some simple math, it can be seen that, for a 12×9 m field, with 50 mm of resolution for the grid, which is the size of the smallest object (the ball), the grid map would end up with 43200 cells, or, a 240×180 map. Therefore, using grid maps in the SSL is not a good choice for the A* algorithm.

Alternatively, [13] provides a study on applying a visibility graph as a map of the environment. However, the results show that the computational time to build the visibility graph is the major drawback of the algorithm. In some of the tests made by the author, the computational time reached the order of a few minutes. The reason for this is mostly due to the process of connecting the vertexes of the objects that are considered in the graph, even though many of the connected obstacles may not be used when finding a feasible path. With that in mind, the DVG [12]

was chosen as the environment representation for the A* algorithm. The choice was based on the fact that the DVG provides a better computational time due to the pruning of unnecessary obstacles. More details about the DVG are discussed in Section 3.1.

Note that there are some techniques which focus on improving the performance of grid maps, i.e., to reduce its memory consumption, hence, reducing the computational time to explore the map. These techniques are know as multi resolution grids, or, multi scale grids. This technique has already been explored within the league and can be seen on [17]. In this case, the author proposes that a grid map is divided into many different layers, which, each layer has an increased resolution, therefore, the areas close to the obstacles shall use layers with a greater resolution (smaller cells), and free space regions shall use layers with a lower resolution (larger cells), this way, the total memory consumption of the grid is reduced, whilst maintaining the precision needed to safely avoid obstacles. Outside of the SSL there's also some recent research on this topic with some experiments on 2D and 3D environments [9]. In this work, the author uses a multi resolution map combined with multiple weighted A* searches. This is a quite more complex implementation, but, it could lead to good results in the SSL, specially due to it having a lot of free space in the environment. Therefore, this is certainly a topic of research to be considered for future works.

3.1 Dynamic Visibility Graph – DVG

According to [12], the DVG has few different characteristics when compared to a common visibility graph [31]. Instead of considering every obstacle in the environment, it will only consider those that are in an active region, which is the main improvement achieved by the DVG. The main idea behind the active region is that the shortest path between points **A** and **B** is the straight line that connects both of them. However, when there are obstacles involved, the shortest path shall be the one that is tangent to the obstacles which intercept the line $\overline{\boldsymbol{AB}}$. That is the general concept involved in the DVG, for more details see [12]. Although the concept is simple, by pruning unnecessary objects it greatly improves the computational time required to build the graph when compared to a common visibility graph [12].

In summary, the graph generated by DVG represents every vertex the robot can travel to and is in the region that contains the shortest path. On Fig. 2 there is an example of the DVG generated to take the robot 0 to the center of the field, the purple lines represent possible paths the robot can take.

All objects were modeled as a polygon which is defined by a radius r and an angle θ, which are the radius of the circumscribed circle and the angle between each vertex,

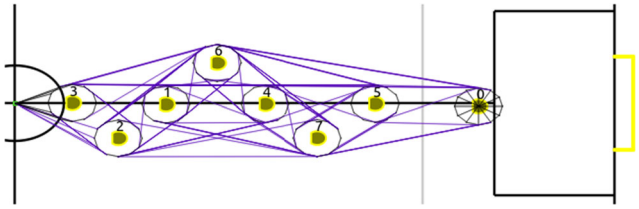

Fig. 2 DVG created to take robot 0 to the center of the field

respectively. The values of r and θ used in this study are:

- $r_{ally} = 240\ mm$
- $r_{opponent} = 240\ mm$
- $r_{ball} = 300\ mm$
- $\theta = 45°$

Due to the previously mentioned characteristics of the robots of the SSL, the robot orientation can be ignored in the path planning step, hence, the dimension of the state space is $\mathbb{R}^2$.

As a simplification, the process of finding the active region is a little different from what can be seen on [12]. In this work, the active region is determined as follows: given the start and goal points, **S** and **G**, respectively, the active region is the rectangle that is parallel to the segment $\overline{\boldsymbol{SG}}$, has a width of $2d_{AR}$ and a length of $\|\overline{\boldsymbol{SG}}\|$. The value of d_{AR} used throughout all tests is 1 m. This simplifies significantly the process of finding the active region, therefore, the only process that is computationally expensive is to connect all the possible vertexes of the obstacles in the active region. Note that this simplification reduces even further the number of considered obstacles, thus, depending on the size of the these obstacles, it might affect the search algorithm completeness, it could define an active region where there were no valid paths inside that region and the only valid path would need to go outside of it, for instance. In the SSL environment this isn't a problem, since all objects have the same size it is practically impossible that no path exists inside the active region.

Another concern pointed out by [8], is that a visibility graph, in general, will have performance issues when the number of obstacles involved grows. This happens due to the high cost of connecting the vertexes of the objects. However, it can be improved by reducing the number of vertexes that each object has, or, by connecting only the tangent lines that connect two objects. The approach taken to solve this problem was to only connect vertexes which the distance between them is greater than the distance between the center of each object, i.e. $d_{vertexes} \geq d_{centers}$. When the objects have the same radius/size, this mathematically results in connecting the tangent lines between them.

In order to verify the impact of this improvement, a simple test was made, it consisted of generating a path using

the DVG+A* in a situation where there were a significant amount of connected vertexes, in this test, 200 samples were collected. The physical setup for this test, i.e., robot placement, as well as the visual result can be seen on Fig. 3. The difference in the number of vertexes connected can be clearly seen when no limit at all is applied (top figure), and, when limiting the length of the connections between objects (bottom figure). In Table 1, the mean, standard deviation (σ), minimum and maximum values for the computational time and path length achieved by the DVG+A* are shown.

By analyzing the results it can be seen that, in general, this change was capable of reducing the computational time by roughly 50% at the expense of increasing the path length by a little more than 6%, which is expected, since the state space was reduced.

Another way to minimize the effect of multiple obstacles in the active region is to group the objects that are close to each other, an example of that can be seen on Fig. 4. The grouping works by detecting which objects are close to each other by a given distance d_{group}, and then, all those within the same group are replaced by another object, which, its center is the geometric center of the group, and the radius is such that it contains all objects plus the r value of the object which is further away from the center of the group.

The effects of this optimization were tested under the worst case scenario for the DVG+A* (see Section 5) and the comparison between the computational time, path length and path safety can be seen on Table 2, just like the others tests, there were 200 samples collected. Also, Fig. 5 shows the difference between grouping and not grouping close obstacles, visually it looks like the path length doesn't change dramatically, this can be confirmed by analyzing Table 2.

Table 2 also shows that there is a slight improvement (approximately 15%) in the path safety, which is expected, due to the increased size of the polygons for larger groups, leading to a greater distance between the robots from the group and the path.

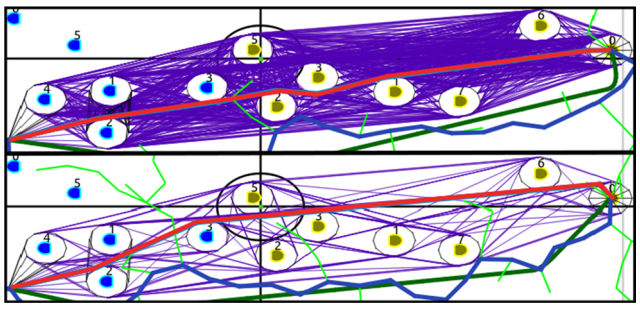

Fig. 3 Difference when limiting the number of connections between objects. The purple lines are the DVG, the red lines represent the A* path, the blue and dark green lines are the pre and post processed RRT paths, respectively. The light green lines are the RRT Tree

Table 1 Difference in computational time and path length

	Time [ms]		Path Length [m]	
	Without Limit	With Limit	Without Limit	With Limit
Mean	6.65	3.14	6.72	7.15
σ	4.47	2.37	0	0
Min	2.92	1.31	6.72	7.15
Max	22.78	12.97	6.72	7.15

Even though this change greatly improved the computational time of the DVG+A*, some further investigation done with the logs from the games of the last year competition from both league divisions, showed that, in general, the robots in a SSL game tend to stay dispersed over the field, i.e., most of the time there aren't groups of robots bigger than two or three robots closer than $d_{group} = 500\ mm$ or even $d_{group} = 1500\ mm$. Considering the results shown in Table 3, it is possible to conclude that groups of robots larger than two or three robots are difficult to happen during a match. Therefore, in the SSL environment the grouping technique wouldn't drastically increase the performance of the DVG+A*, but, it still is a feasible solution to improve the worst case scenario of the algorithm.

Although groups greater than four robots seem to occur, as seen on Table 3, they usually only happen at the very beginning of the match, while teams are still doing some final preparations, thus, the robots are just sitting side by side. Also, the amount of such large groups where only detected in roughly around 3.23% of the total log frames analyzed.

At last, it is important to note that, the grouping isn't applied unconditionally, if the resulting obstacle group contains the start or goal points within its region, the group will not be built and the robots that would compose it, will be treated normally. This condition is important to not affect the completeness of the algorithm, otherwise, there may be some cases where a path exists, but, due to the obstacle grouping it may not be found by the algorithm.

Still regarding the completeness of the algorithm, the object radius ($r_{ally}, r_{opponent}$ or r_{ball}) is greater than the radius of the real object, this increased radius is a safety

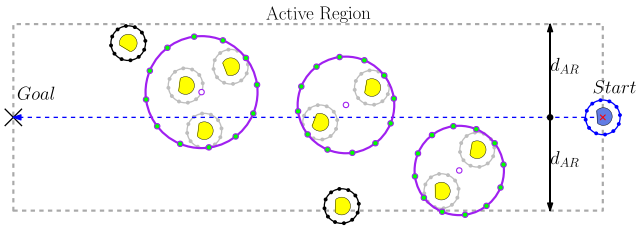

Fig. 4 Obstacle grouping to reduce the number of objects that need to be connected inside the active region

Table 2 Computational time, path length and path safety comparison when using obstacle grouping inside the active region of the DVG

	Time [ms]		Path Length [m]		Path Safety [m]	
	No Grouping	Grouped	No Grouping	Grouped	No Grouping	Grouped
Mean	7.39	1.53	8.27	8.24	8.60	9.53
σ	4.67	0.35	0.00	0.00	0.00	0.00
Min	3.63	0.31	8.27	8.24	8.60	9.53
Max	22.04	2.48	8.27	8.24	8.60	9.53

offset to avoid collisions. Hence, one may note that this could also cause, at least the goal point to be obstructed by another robot. In order to avoid this situation, the following special condition is used when analyzing which vertexes to connect with the goal point: if the line that connects a given vertex and the goal point doesn't intersect, or gets closer than the robot radius, to the center of the obstacle, then, the connection can be made. An example of this happens when two robots are disputing the ball, as illustrated in Fig. 6, in this case, both nodes N_1 and N_2 can be safely connected to the goal position, even though $D_1 = 0$, it doesn't intersect the obstacle. In conclusion, although the representation might not be completely accurate regarding the free space, the algorithm should always have a solution to find.

3.2 A*

The A* [10] is an improvement over the Dijkstra algorithm [4], it has a heuristic function that guides the algorithm to reach the goal faster. And, just like the Dijkstra algorithm, it is capable of finding the shortest path within the explored graph. The algorithm uses an evaluation function, which, can be seen in Eq. 1.

$$f(s) = g(s, s_{prev}) + h(s, s_{goal}) \quad (1)$$

Where, $g(s, s_{prev})$ is the cost to move from the previous state (s_{prev}) to the actual state (s), $h(s, s_{goal})$ is the heuristic function, a common heuristic used is the euclidean distance, and $f(s)$ is the evaluation value of the current state (a state is a position that the robot can go to).

The A* algorithm has three basic steps:

1. Expand the neighbors of the current state and calculate the $f(s)$ for them;
2. Add the neighbors to the open list;
3. Retrieve the state with the lowest $f(s)$ from the open list and add it to the closed list;

When the goal state gets added to the open list, or, the open list is empty, the algorithm finishes. Then, it is only necessary to retrieve the final path from the closed list if one was found. This last step can be greatly improved by using a linked list.

4 Rapidly-exploring Random Trees – RRT

As seen previously, the RRT is a widely used algorithm in the league, mostly due to its simplicity and that it doesn't require a representation of the environment (map), which, A* needs. There are many variations of this algorithm [7, 14, 19, 28], but the one used in this study is the original version.

The RRT algorithm also has three basic steps:

1. Create a random point (P_{rand}) in the search space;
2. Find the closest tree vertex (P_{close}) to P_{rand};
3. Extend the tree from P_{close} to P_{rand} in a given fixed distance, λ, if there are no obstacles in the path.

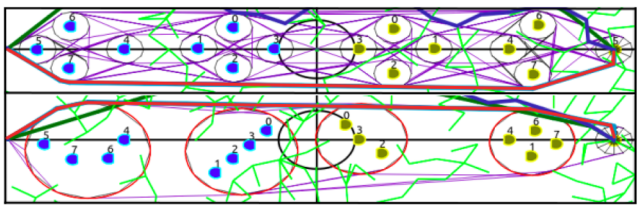

Fig. 5 Difference when grouping close obstacles. In the upper image the obstacles aren't grouped, in the lower image they are grouped. The red path is the A* path, the purple lines are the DVG, the blue and dark green paths are the pre and post processed RRT paths, the light green lines are the RRT Tree

Table 3 Mean, standard deviation and maximum values of the robot group size during a SSL match

d_{group}	Group Size	Log 1	Log 2	Log 3	Log 4
500 *mm*	Mean	1.20	1.27	1.08	1.32
	σ	0.76	0.61	0.36	0.66
	Max	8.00	7.00	5.00	4.00
1500 *mm*	Mean	1.44	1.43	1.22	1.76
	σ	1.16	1.04	0.65	1.84
	Max	14.00	9.00	8.00	15.00

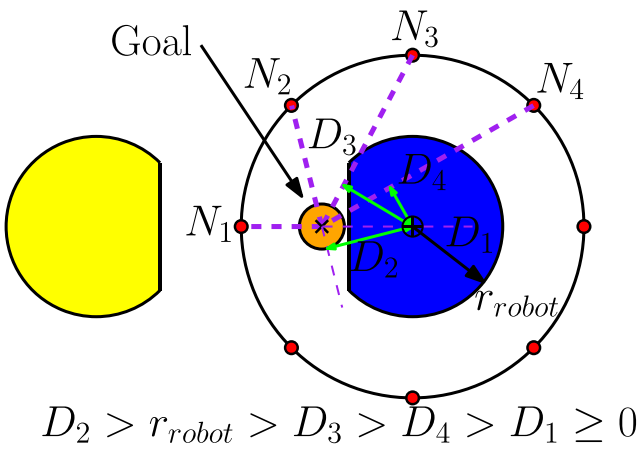

Fig. 6 Special condition example

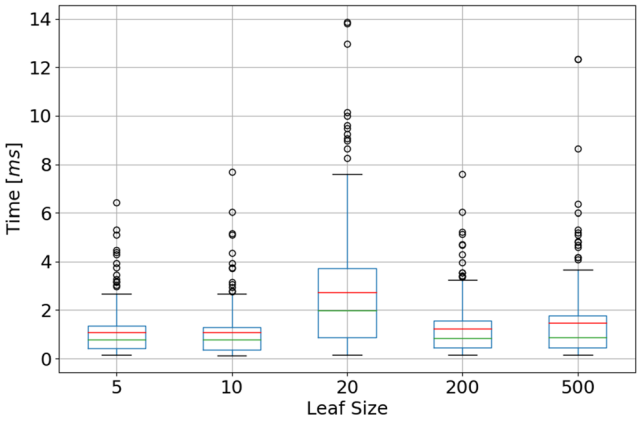

Fig. 8 Computational time variation due to the leaf size changes

These steps repeat until the maximum number of iterations, N, is reached, or, the tree reaches a minimum distance from the goal, d_{goal}. In Fig. 7 there's an example of this process, the red points are the RRT tree vertexes.

After a path is found, a post processing algorithm is applied to make the RRT path more smooth. The algorithm used can be seen on Algorithm 1, and it is quite simple. The algorithm prunes unnecessary points while not causing any collisions. When analyzing any figure with the pre and post processed paths the effect of this algorithm can be seen on Fig. 3. This step substantially reduces the raggedness of the path, however, there are more elaborate techniques to accomplish this task [32]. Also, there are some cases where the presented algorithm might leave right angles in the path, which can be hard for the robot to follow. In general, due to the holonomic properties of the robot, this algorithm is enough to generate a feasible path.

Algorithm 1 Path post processing.

Input : The path generated by RRT
Output: The post processed path
Data: $i \leftarrow 0$;
Data: $n \leftarrow$ `size(`*rrtPath*`)` -1;
ppPath.clear();
ppPath.append(*rrtPath*[*n*]);
while $n > 0$ **do**
  while `collision(`*rrtPath[i], rrtPath[n]*`)` **do**
    $i++$;
  end
  ppPath.prepend(rrtPath[i]);
  $n \leftarrow i$;
  $i \leftarrow 0$;
end

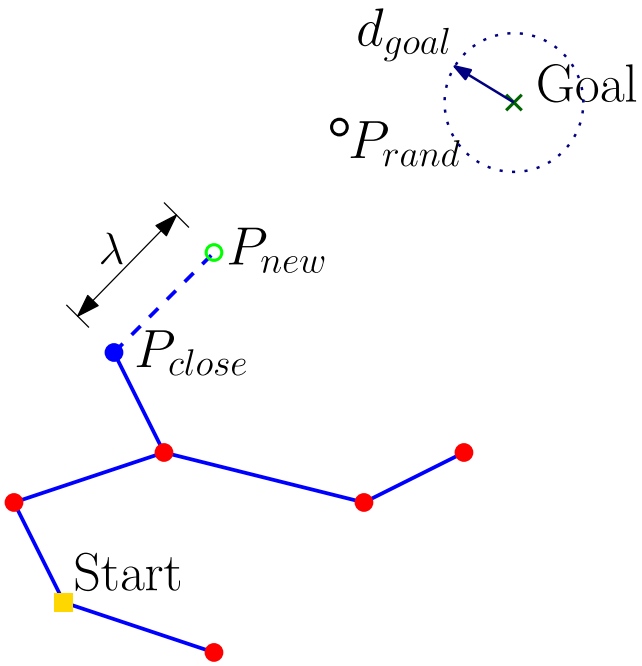

Fig. 7 Example of an RRT step

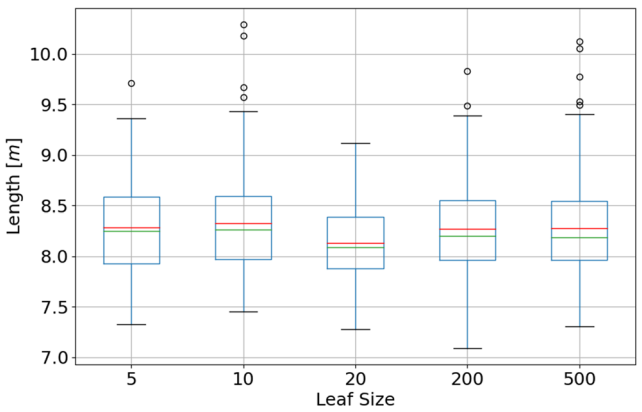

Fig. 9 Path length variation due to the leaf size changes

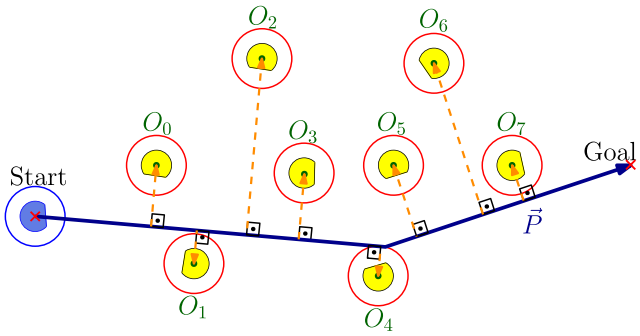

Fig. 10 Example of how the path safety is calculated

By using the RRT algorithm, a famous computing problem must be addressed, which is the nearest neighbor search. This procedure happens in the RRT's process of finding the vertex P_{close}. The KD-Tree is a data structure that provides a great solution to this problem. It can be seen on [7] that the use of a KD-Tree greatly decreases RRT computational time when compared to a non KD-Tree approach. In this work, the nanoflann library [3] was used to implement the KD-Tree. The reason for it is the simplicity to use and that only one parameter needs tunning, which is the leaf size. The leaf size represents how many leafs are grouped in each KD-Tree branch, this parameter influences the time needed to build the tree index and the search time, i.e., the greater the leaf size, the longer a search may be, but, the time to build the index is lower, and when the leaf size is lower, the opposite happens [3].

To better adapt the leaf size to this application, some tests were made with five intermediate values for it, which are 5, 10, 20, 200 and 500. The results regarding computational time and path length can be seen on Figs. 8 and 9, Section 6 explains in details the components of a box plot.

By analyzing Figs. 8 and 9 it can be seen that the path length and computational time doesn't change considerably with the leaf size, except for the leaf size of 20. In this case, the computational time obtained seems to be some kind of anomaly. According to [3], the performance of the KD-Tree depends heavily on the application. Therefore, by analyzing some benchmarks available in [3], it can be seen that this kind of variation is possible within a 95% confidence interval, hence, this could be the normal behavior of the KD-Tree for this application.

Consequently, the leaf size of 5 was chosen due to having the lowest maximum computational time, and the constants for RRT are:

- $\lambda = 360\ mm$;
- $d_{goal} = 360\ mm$;
- $N = 2500$;

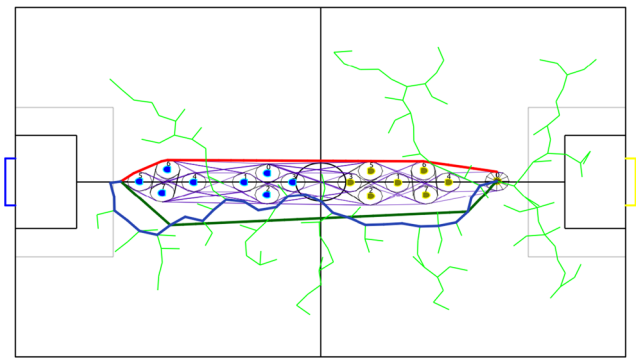

Fig. 11 Worst case scenario for the DVG+A* algorithm. The purple lines are the DVG, the red lines represent the A* path, the blue and dark green lines are the pre and post processed RRT paths, respectively

5 Methodology

The algorithms were programmed using C++ in Qt framework. The code was compiled in release mode to improve its performance. The hardware used to make all the simulations is an Intel i7–9750H@4.5GHz on Ubuntu 20.04 LTS 64 bits.

Regarding the experiments, both algorithms were tested under two different perspectives with multiple test setups, where, 200 samples were collected for each test. In the second perspective, one sample is counted once the robot leaves the initial position and arrives at its destiny. As for the first perspective, each calculated path count as one sample.

The first perspective will test the algorithms in a static environment. The parameters used in the comparison are: computational time, path length and path safety. The path safety, in this case, is defined as: given a path **P**, the path safety is the sum of the perpendicular distances between the considered objects, O_n, $n = \{0 \dots j\}$ and the path **P**. Figure 10 exemplifies this, the orange lines represent the distance from the opponents to the blue path **P**.

The setups proposed in the first perspective aims to explore what are the worst case scenarios for both algorithms, as well as showing how they perform in some common situations that may happen during an SSL game. With that in mind, four scenarios were developed and they are described below.

Figure 11 exemplifies the first test scenario, which is the worst case scenario for the DVG+A* algorithm. As previously mentioned, the process of generating the DVG is the most critical part when generating a path, thus, by placing all possible obstacles in the active region without grouping them, it is expected that this configuration results in the highest computational time for the DVG+A*.

Figure 12 exemplifies the second scenario, i.e., worst case scenario for the RRT algorithm. This setup can be considered as such due to having a narrow passage before reaching the goal point, and, having a considerable distance

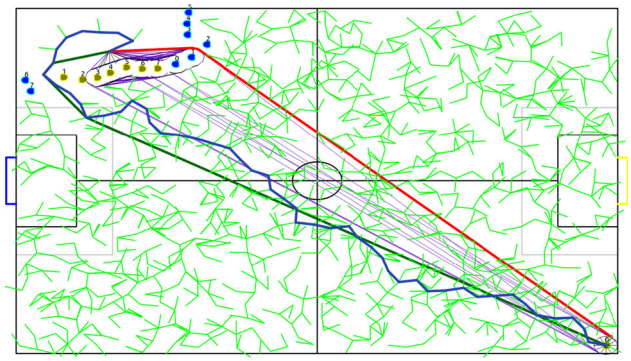

Fig. 12 Worst case scenario for the RRT algorithm. The light green lines are the RRT tree, the dark blue and dark green lines are the pre and post-processed RRT paths, respectively, the red line is the DVG+A* path and the purple lines are the DVG

between the start and goal points. These characteristics make it harder for the RRT to find a path due to the fact that it doesn't have any kind of heuristic or bias to guide the exploration. Therefore, it is expected that the algorithm explores practically the whole environment before finding a path, thus, making the computational time in this case to be higher.

The third scenario proposes a mix between the worst case scenarios for both algorithms, i.e., it has a considerable amount of obstacles in the path and the distance between the start and goal points is not short. This can be seen on Fig. 13.

The last scenario exemplifies a common in-game situation, which is positioning during a game stoppage. In this case, the rules define that the robots need to keep a distance of at least $0.5m$ from the ball (illustrated by the yellow circle). This can be seen on Fig. 14.

On the second perspective, the algorithms were evaluated according to their ability to navigate the robot through a dynamic environment, i.e., the obstacles that were stationary are now moving through the path. The parameters that were measured are:

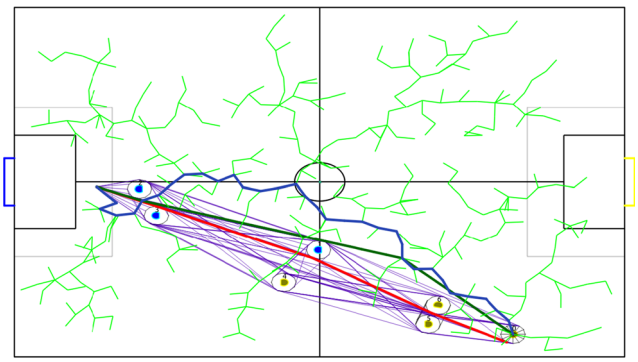

Fig. 13 Mixed scenario for the RRT and DVG+A* algorithms. The light green lines are the RRT tree, the dark blue and dark green lines are the pre and post-processed RRT paths, respectively, the red line is the DVG+A* path and the purple lines are the DVG

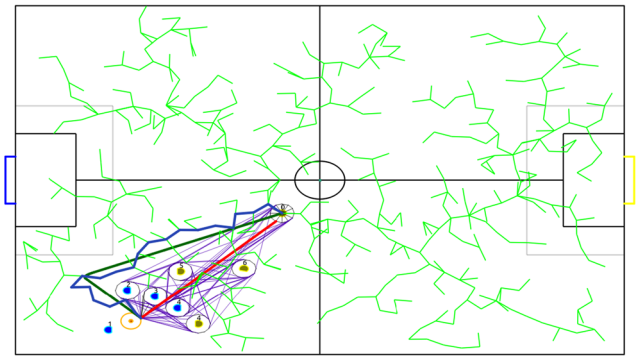

Fig. 14 Positioning during a game stoppage. The light green lines are the RRT tree, the dark blue and dark green lines are the pre and post-processed RRT paths, respectively, the red line is the DVG+A* path and the purple lines are the DVG

- Accumulated Computational Time: is the sum of the cost of all paths that had to be generated until reaching the goal position, measured in milliseconds.
- Paths Recalculated: is the total number of times a previous path was invalid and had to be replanned.
- Navigation Time: is the total amount of time between leaving the start position and reaching the goal position, measured in seconds.
- Number of Collisions: is the number of times the robot collided with any obstacle.

Regarding the number of paths recalculated, a path is considered invalid whenever an obstacle crosses it, when that happens the whole path is discarded and replanned. To define when an obstacle crosses the path, there is a threshold distance of 90 mm, which is the radius of the robot. It is important to note that this analysis doesn't take into account any kind of dynamic parameter, e.g., it doesn't detect if there is a robot moving in the direction of the path and might cross it in the future. Therefore, even though a new path would only need a minor change in the invalid path, the whole algorithm needs to run again. That happens for both DVG+A* and RRT.

There were two different scenarios built to test the algorithms in the dynamic perspective, the difference between them is basically how many obstacles are in the path between the start and goal points and how much overlap there is between the obstacles trajectories and the start to goal line. Figures 15 and 16 shows the most and less crowded environments, respectively. The start and goal points for the robot are defined by the red horizontal line.

6 Results

In this section, the results of the previously defined tests are presented. Firstly, the tables containing the statistical results

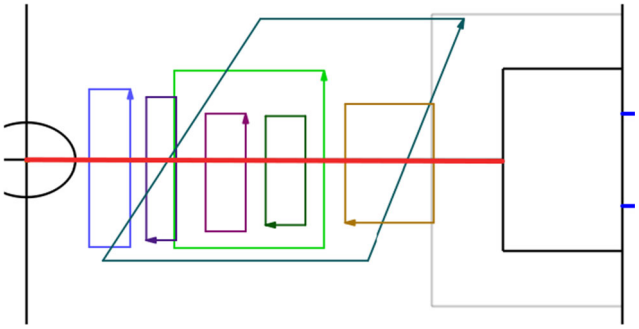

Fig. 15 Dynamic test with 7 moving obstacles. Each color represents the path of a dynamic obstacle. The red line represents the start and goal points of the algorithm

will be presented and discussed. After that, each parameters will also be analyzed graphically using box plots, which, the lower bound of the box represents the 25 th percentile (q_1), the upper bound of the box represents the 75 th percentile (q_3), the whiskers represents the most extreme samples that aren't considered outliers, e.g., a sample s is considered as such if $s \geq q_3 \pm 1.5(q_3 - q_1)$. The median and mean values are represented by the green and red horizontal lines, respectively, and the outliers are the black circles.

6.1 Static Environment Perspective

The first test to be analyzed is the DVG+A* worst case scenario. The results can be seen on Table 4, and, just as expected, the average computational time for the DVG+A* was almost four times higher than RRT, this clearly shows that an environment with lots of obstacles causes the computational of time algorithm to increase considerably. Regarding the path length, the results are also according to the expected, although RRT was able to find some paths shorter than DVG+A*, on average they performed pretty similarly. It is also noticeable that the standard deviation for the path length and safety are zero, this happens due to both

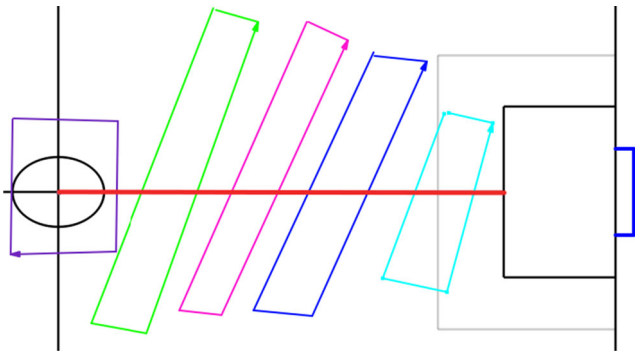

Fig. 16 Dynamic test with 5 moving obstacles. Each color represents the path of a dynamic obstacle. The red line represents the start and goal points of the algorithm

Table 4 Results for the worst case scenario of the DVG+A* algorithm

	Time [ms]		Path Length [m]		Path Safety [m]	
	DVG+A*	RRT	DVG+A*	RRT	DVG+A*	RRT
Mean	6.50	1.52	7.58	7.93	7.84	10.96
σ	4.55	1.66	0.00	0.43	0.00	3.10
Min	3.41	0.13	7.58	6.93	7.84	2.41
Max	26.07	12.85	7.58	9.14	7.84	16.37

DVG and A* being deterministic algorithms, thus, given the same inputs, the results must be the same, therefore, for the static tests the DVG+A* always calculates the same path.

One interesting observation to make is that, because the DVG limits the configuration space for the A* search, the shortest path within the graph may not, and in general isn't, the global shortest path. This effect also happens when using grid maps and [18] provides an important analysis of it.

Table 5 shows the results for the RRT worst case scenario. By analyzing the data it is clear that the algorithm did struggle with the proposed configuration, its maximum computational time was approximately 41.7% higher than the maximum computational time of the DVG+A* in Table 4. Moreover, the path length also increased significantly, it wasn't able to find any paths shorter than DVG+A*, and, the average difference grew from 4.06% in Table 4 to 12.7%. This shows that, when in a more constrained environment, the RRT algorithm suffers a significant performance decrease. Regarding the path safety, the algorithm successfully kept a greater distance from the objects when compared to the DVG+A*.

Throughout some preliminary testing it was observed that the total computational cost was lower if the RRT algorithm was allowed to run longer instead of restarting it. Because of that, during the RRT worst case scenario, the maximum number of iterations (N) was increased, from 2500 to 5000, to guarantee that the algorithm would be capable of finding the path. The maximum number of iterations during the test was 4513, and it took on average 1300 iterations to find a path.

The results for the mixed scenario can be seen on Table 6. It is interesting to see that apart from the maximum values

Table 5 Results for the worst case scenario of the RRT algorithm

	Time [ms]		Path Length [m]		Path Safety [m]	
	DVG+A*	RRT	DVG+A*	RRT	DVG+A*	RRT
Mean	1.38	8.86	12.94	14.58	4.42	6.40
σ	0.71	6.10	0.00	0.23	0.00	0.41
Min	0.37	1.00	12.94	13.98	4.42	5.14
Max	4.53	36.95	12.94	15.79	4.42	8.37

Table 6 Results for the mixed case scenario

	Time [ms]		Path Length [m]		Path Safety [m]	
	DVG+A*	RRT	DVG+A*	RRT	DVG+A*	RRT
Mean	0.94	1.13	9.02	9.41	1.65	4.29
σ	0.70	1.29	0.00	0.50	0.00	1.78
Min	0.45	0.16	9.02	8.36	1.65	1.03
Max	4.75	8.76	9.02	11.13	1.65	9.13

and the path safety, the algorithms performed very similarly. The average computational time of RRT was only 0.2 *ms* higher than DVG+A*, and the path length was 4.6% longer. The major differences were in the maximum computational time, which was 84.4% higher for RRT, and the path safety, which, RRT maintained more than the double of the distance that DVG+A* kept.

At last, Table 7 shows the results for the stoppage scenario. In this setup, the RRT algorithm struggled more than on the mixed scenario. The average computational time for RRT was 97.4% higher than DVG+A*, also, the maximum value more than tripled. The path length also was considerably higher, on average it was 20% longer and the maximum values almost doubled. One possible reason for these results is that the goal location is almost at the edge of the field, therefore, generating random points at that region might not happen very often, thus, the exploration takes longer to reach that area.

6.1.1 Computational Time Analysis

By analyzing Fig. 17 it is possible to see that with the exception of its worst case scenario, the DVG+A* algorithm was faster than RRT in all the other scenarios. Although RRT was capable of achieving the lowest minimum computational time, DVG+A* is more consistent, this can be identified by the height of the boxes in the plot, e.g., a taller box indicates that the distribution of the samples is more dispersed, i.e., higher standard deviation.

Another observation to make is that, when comparing the algorithms in each of its respective worst case scenario, the DVG+A* also performed better than RRT, even though

Table 7 Results for the stoppage case scenario

	Time [ms]		Path Length [m]		Path Safety [m]	
	DVG+A*	RRT	DVG+A*	RRT	DVG+A*	RRT
Mean	1.92	3.79	3.65	4.38	3.14	3.88
σ	0.92	2.94	0.00	0.74	0.00	1.04
Min	0.44	0.24	3.65	3.05	3.14	1.37
Max	5.73	19.69	3.65	6.32	3.14	5.33

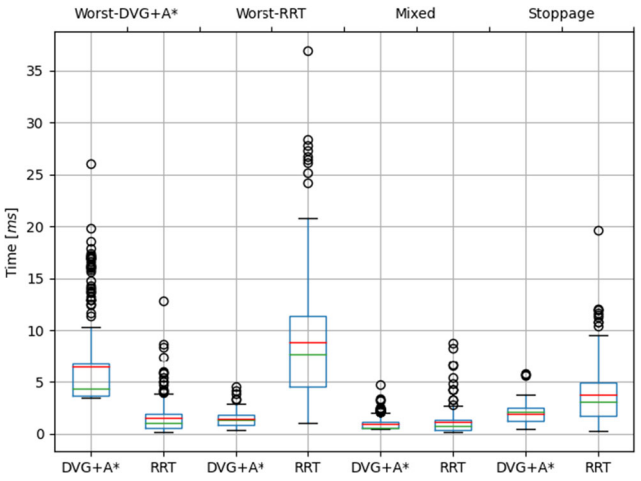

Fig. 17 Box plot of the computational time for both algorithms in all scenarios

its minimum time was around 3.4 *ms*, approximately 75% of the samples are lower than 7.5 *ms*. Conversely, RRT minimum computational time was around 1 *ms*, but, it had approximately 50% of the samples higher than 7.5 *ms*.

Further discussions on the computational time can be seen on Section 7.

6.1.2 Path Length Analysis

Figure 18 shows the box plot for the path lengths of the algorithms throughout the test scenarios. Since there is no variation in the path length for DVG+A*, there is practically no box representing its samples, however, this plot still is useful to give some insights into the path length samples distribution for RRT. It can be seen on the plot that, although RRT is indeed capable of finding shorter paths than DVG+A*, this only happens for less than 25% of the samples in general. As already mentioned, the only reason

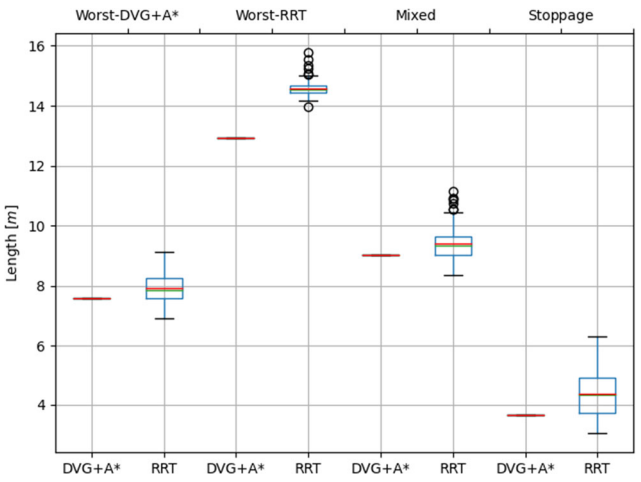

Fig. 18 Box plot of the path length for both algorithms in all scenarios

that RRT is capable of finding shorter paths than DVG+A*, is due to the reduced configuration space, of the DVG.

6.1.3 Path Safety Analysis

Figure 19 shows the box plot of the path safety for the algorithms in all test scenarios. Similarly to what happened in the path length analysis, the distribution shape for the path safety samples of the DVG+A* are the same as the path length.

Due to the stochastic behavior of the RRT algorithm, as well as how the DVG+A* generates a path, there were no doubts that the RRT paths would have a higher path safety, however, it is interesting to see that some RRT samples have a lower path safety than DVG+A*. The most critical cases are in the DVG+A* worst scenario, where the RRT algorithm had a considerable amount of samples lower than 8 m, this can be seen by how much the lower whisker is extended, meaning that even samples as low as 4 m would still not be considered outliers. The second critical case is in the stoppage scenario, where it can be seen that around 25% of the RRT samples have a path safety lower than DVG+A*. More details about this behavior can be seen on Section 7.

6.2 Dynamic Environment Perspective

In this section, the results obtained during the dynamic scenario tests shall be presented. The goal of these tests was to investigate how the algorithms would perform in a environment with moving obstacles and a stationary goal position. There were two different scenarios, the first had about seven obstacles, and the second had five obstacles.

Table 8 presents the accumulated computational time of the algorithms during the tests. The results show that the DVG+A* had a lower computational time. However, it must also be taken into account that the DVG+A* had to recalculate its path fewer times than RRT, because of that, even though it had the lowest total computational time, the individual computational time was more than two times higher than RRT in the first scenario. In the second test scenario, the RRT average individual time was, approximately, 32% higher than DVG+A*.

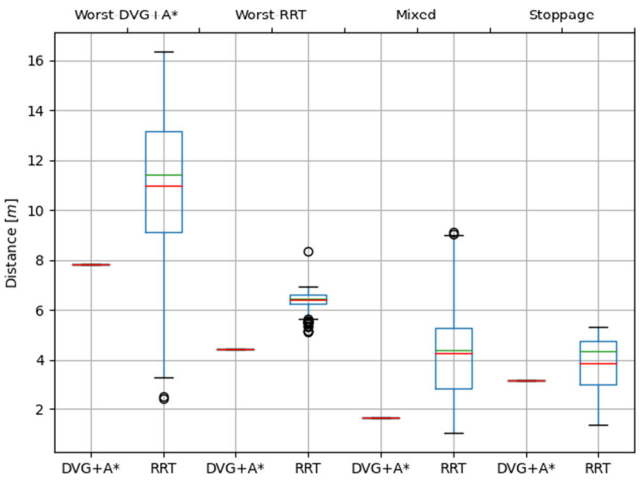

Fig. 19 Box plot of the path safety for both algorithms in all scenarios

Table 8 Accumulated computational time for the dynamic scenario tests

Time [ms]	Scenario 1		Scenario 2	
	DVG+A*	RRT	DVG+A*	RRT
Mean	6.80	7.12	1.49	2.36
σ	3.03	2.98	0.83	1.63
Min	1.17	1.26	0.12	0.04
Max	14.99	17.28	4.75	8.67

Table 9 presents the number of times each algorithm recalculated a path. As previously mentioned, the results clearly show that the DVG+A* algorithm had to recalculate its paths fewer times than RRT, with the exception for the second scenario, which its maximum value is higher than RRT. It is also interesting to see that, in this scenario there were at least one case where both algorithms didn't had to recalculate the original path.

The results regarding how much time it took the robot to reach the goal position can be seen on Table 10. In this parameter the algorithms performed very similarly. The only noticeable difference is in the second scenario, where the maximum time for RRT was more than two seconds higher than DVG+A*. During a real SSL match, such delay could possibly result in a play gone wrong or something like that. In order to verify if this was a random outlier, a graphical analysis of this parameter is definitely required. This is one of the most important dynamic parameters due to the nature of an SSL game, e.g., if the robots from team A takes, in general, 2 seconds longer than team B to reach its goal point, then, team A would most likely be dominated by team B during the whole match, possibly leading to team A being defeated.

Table 9 Number of times it was necessary to recalculate a path during navigation

Recalculated Paths	Scenario 1		Scenario 2	
	DVG+A*	RRT	DVG+A*	RRT
Mean	13.09	34.08	5.79	6.93
σ	5.05	9.60	3.15	3.32
Min	5.00	10.00	0.00	0.00
Max	48.00	61.00	25.00	19.00

Table 10 Time needed to navigate from the start to the goal position

Time Elapsed [s]	Scenario 1		Scenario 2	
	DVG+A*	RRT	DVG+A*	RRT
Mean	5.98	5.89	5.30	5.40
σ	0.59	0.71	0.35	0.61
Min	5.00	4.50	4.65	4.65
Max	8.15	8.40	6.30	8.55

Regarding the number of collisions, both algorithms did not perform well. Table 11 shows the results for this parameter. It can be seen that DVG+A*, in general, had a similar performance than RRT, its maximum and mean values were either very close or slightly higher than RRT.

As previously mentioned, the game rules [22] penalizes practically every collision that happens between the robots, thus, the goal of a path planning algorithm is to successfully drive the robot around the field without any collisions, but this is not a simple task. More details about this will be discussed on Section 7.

6.2.1 Graphical Analysis

As previously mentioned, the graphical analysis of the parameters is also a very useful tool, for it allows to get a better understanding of the distribution shape of the analyzed parameter. Although the tables may show what are the minimum and maximum values, it does not contain the information regarding if these data points are considered outliers or not, for example.

With that in mind, Fig. 20 shows the box plot of the accumulated computational time. The plot shows that in the first scenario, although RRT had the highest accumulated time, most of these high value samples are considered outliers. In general, the samples distribution in the first scenario are similar, the DVG+A* algorithm has a slightly lower accumulated time than RRT for 75% of its samples.

In the second scenario, the DVG+A* performed significantly better than RRT, it can be seen that around 25% of the RRT samples had a higher accumulated time than almost all of the DVG+A* samples that are not considered outliers,

Table 11 Number of collisions during navigation

Collisions	Scenario 1		Scenario 2	
	DVG+A*	RRT	DVG+A*	RRT
Mean	2.10	2.12	0.89	0.74
σ	1.51	1.40	1.09	0.84
Min	0.00	0.00	0.00	0.00
Max	7.00	6.00	5.00	3.00

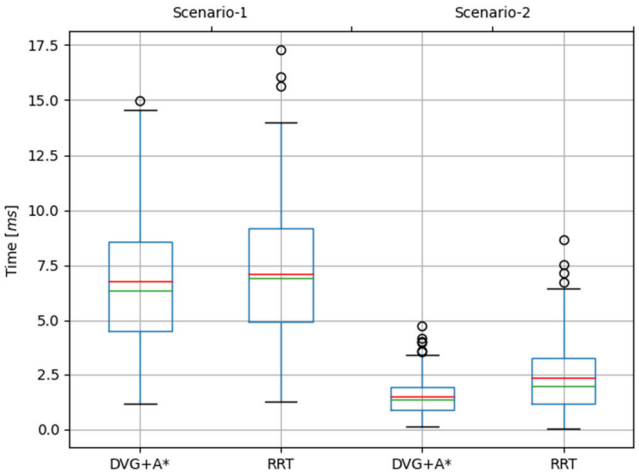

Fig. 20 Box plot of the accumulated computational time in the dynamic scenario tests

also, 75% of the DVG+A* samples have a lower time than 50% of the RRT samples. This is a reasonable result, since the second scenario had fewer obstacles and these obstacles were more dispersed.

The plot for the number of recalculated paths will not be presented, due to it not adding too much information beyond what was already possible to see in Table 9.

Moving on with the analysis, Fig. 21 shows the total navigation time. As seen on Table 8, in the first scenario the algorithms performed very similarly. However, in the second scenario the situation changes. Even if the outliers are discarded, it can be seen that the RRT algorithm has 25% of the samples which the navigation time is between 5.75 s and 7 s, conversely, for the DVG+A*, its top 25% of the samples are between 5.5 s and 6.2 s. Not only the amplitude of this interval is lower, but the maximum value also is.

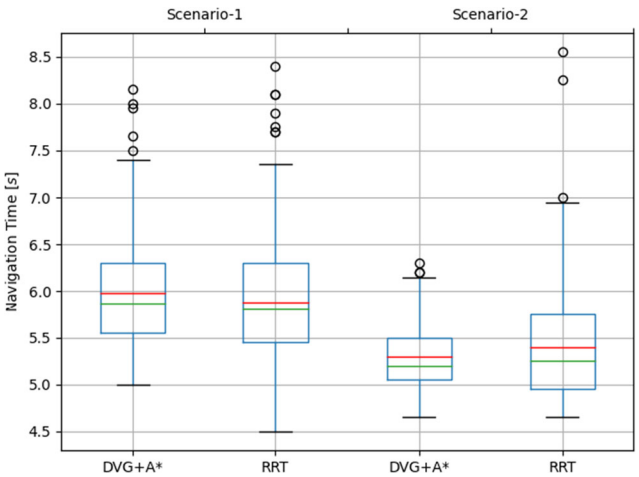

Fig. 21 Box plot of the total navigation time in the dynamic scenario tests

At last, there are Figs. 22 and 23 which show the histogram that depicts the frequency of collisions that occurred when moving from the start to the goal point, e.g., Fig. 22 show that there were approximately 12 and 8 cases where the robot collided with 5 obstacles when moving from the start to the goal point for the RRT and DVG+A* algorithms, respectively. To account for consecutive collisions there was a time window of 0.5 s, that is, another collision would only be counted after 0.5 s from the previous. This time window was added mostly to avoid accounting for a collision with the same obstacle more than once. Note that, for example, a path with seven collisions doesn't mean that these obstacles were grouped, it only means that the robot collided with seven obstacles outside the specified time window. Also, the figures show the total count of collisions in the test for each algorithm, i.e., red and green lines.

The first scenario is definitely the worst, the plots show than it had more than double of total collisions for both algorithms when compared to the second scenario, beyond that, both algorithms achieved practically the same number of total collisions, which is around 410 collisions. Figure 22 also shows that it is more common for the algorithms to collide with one or two robots, around 52% and 60% of the DVG+A* and RRT samples, respectively.

Due to the reduced number of objects and more dispersed paths, thus, more sparse obstacles in the environment, the second scenario resulted in less collisions when compared to the first. To be more precise, it can be seen on Fig. 23 that for a little less than 50% of the samples for both algorithms, there were no collisions at all.

By analyzing Figs. 22 and 23 it can be seen that DVG+A* has less collisions with one or two obstacles. However, it also has the collisions with most obstacles in the path. Moreover, there are a few cases, less than 5, where the robot

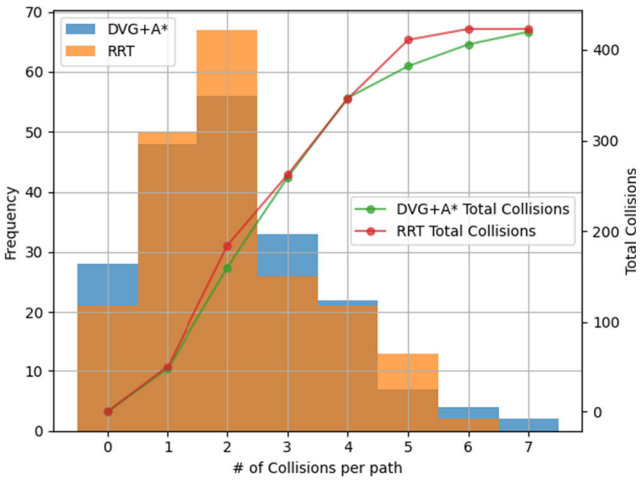

Fig. 22 Histogram of the collisions during navigation for the first scenario

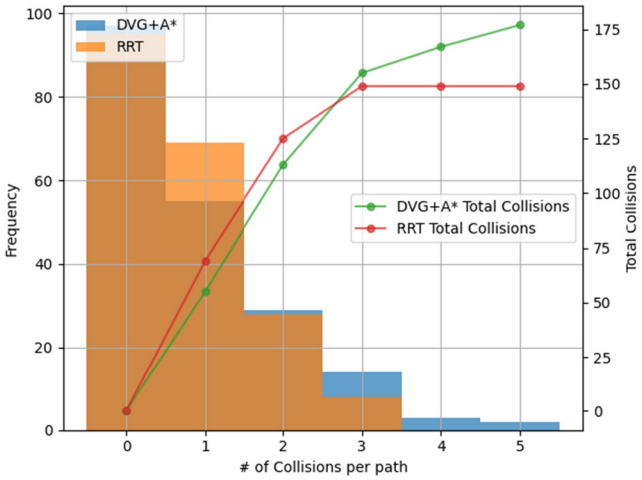

Fig. 23 Histogram of the collisions during navigation for the second scenario

being guided by the DVG+A* collided with all obstacles in the path. If it wasn't for these cases with a lot of collisions in the path, the DVG+A* would've performed equally and even better than RRT.

7 Discussion

By analyzing the results, it is clear that the algorithms aren't capable of dealing with a highly dynamic environment. The prediction of the opponent's position could improve this situation. However, there are two main characteristics of the SSL which makes it a complex task for prediction: the high dynamics of the game and the physical capabilities of the robot itself. The former is related to the high speed of the robots, the ball, passes that may occur and special kicking mechanisms, which can kick the ball in an non-linear trajectory, these properties contribute to a very fast change in the current objective of the opponent, making it harder to predict. The latter is definitely the most important, and, is related to the fact that a robot may change its direction very abruptly while moving at a reasonably high speed, this is completely different from an environment with a high traffic of cars or humans, in these cases there is much less noise associated, and the objects have a much more predictable behavior, e.g., cars/humans won't change its direction suddenly while at a high speed.

These two factors combined create an environment with a high uncertainty about the future position of an opponent robot and even the ball, which also influences the path. Therefore, even though some probabilistic techniques, like the Extended Kalman Filter [33], could be used to track and predict the opponent's position, they probably wouldn't be capable of predicting the position of the robots throughout the whole planning and execution of the path, or, to do

so with an acceptable error, besides making the planning task more complex and time consuming. A feasible way to address this problem would be to predict the position of only the opponents that are close to the allied robot, and in a small time window, ignoring future collisions that may happen beyond this time window, this way, the agent would have to constantly recalculate its path.

Another way to solve this problem would be to infer, using some kind of learning technique, possible decisions of the opponent given the current, and future, situation of the game, hence, using this information to avoid colliding with them. The latter is definitely a complex solution to implement and is out of the scope of this work. It is important to keep in mind that, all of this planning has to happen within a short period of time, in the worst case scenario a well built path could take at most a few hundred milliseconds.

Regarding the static perspective, the DVG+A* had an overall better performance than RRT. The results obtained showed that it usually has a lower computational time than RRT, with a few exceptions where the latter is capable of finding paths almost instantaneously. Moreover, when comparing the algorithms in its respective worst case scenario, RRT performed worse than DVG+A*, both in computational time and path length. For the path length parameter, the results for DVG+A* are more consistent, due to it being a deterministic algorithm. Even though RRT is clearly capable of finding shorter paths than DVG+A*, that is not what happens in general.

Given how much time it took for both algorithms to find a path in its respective worst case scenario, a few more tests were made, to see how the algorithms would perform with a strict time constraint in the static perspective. More specifically, the algorithms were run in both worst case scenarios within a time constraint of 5 ms, the parameters of these tests are the same as before, i.e., 200 samples were collected in each scenario. Also note that these scenarios were chosen due to the fact that they are the most time critical ones, in the other scenarios the difference between the performance of the algorithms wouldn't be too much expressive.

A few reasonable metrics to use in this type of test are the rate of failure (RoF) for each algorithm, which is the percentage of how many times the algorithm wasn't capable of finding a path that reaches the target position and the average distance that the final path was from the target position T_{dist}.

The results can be seen on Table 12. They show that the algorithm with less RoF is the RRT, which is expected. The RRT will always generate a path, even if it doesn't lead to the target position, hence, the importance of the T_{dist} parameter. Although RRT had the lowest RoF, in the case that a path didn't reach the target position, it would be more than double the distance from the target when compared to the DVG+A* paths.

Table 12 Computational time performance within time constraint

	Worst-DVG+A*		Worst-RRT	
	DVG+A*	RRT	DVG+A*	RRT
RoF [%]	29.50%	0.00%	0.00%	25.50%
T_{dist} $[m]$	1.40	0.01	0.00	3.05

Given the results shown in Fig. 17, one may find the RoF lower than expected for the algorithms, however, the values in that graph represent the amount of time it took to find an entire path. For example, the RRT sample higher than 35 ms probably comes from a case where the algorithm failed several times in a row, therefore, that single sample could generate at least seven or eight samples in the current test when assuming a maximum time limit of 5 ms. Therefore, by analyzing Fig. 17, it cannot be assumed that the RRT algorithm should have a total RoF around 75%, which is roughly the amount of samples above 5 ms, for instance. At least not if the same amount of samples is collected for both tests.

Now considering the path safety, it is certainly not expected that the RRT could have a lower path safety than the DVG+A*, due to the fact that by using a visibility graph the A* path will be tangent to all obstacles that it needs to avoid (when using a grid map the same thing happens). This RRT behavior has two possible reasons: the post-processing of the original path, and the free space among groups of robots. Since the smoothing algorithm tries to prune as much points it can from the original path, in some cases it can make the RRT path get closer to the obstacles, this behavior can be seen on Fig. 13, it can be clearly seen on the image that the original RRT path (dark blue) has a greater distance from the obstacles, however, after the smoothing algorithm (dark green) the path is quite similar when compared with the DVG+A* path.

Regarding the second reason, an example of that can be seen on Fig. 24, what happened in this case is that the RRT algorithm was capable of finding a path between the obstacles, which causes the path safety to decrease. This can be seen on the highlighted rectangle, the yellow lines are the contribution of these robots to the path safety for DVG+A*, and the cyan lines are the contribution to the path safety for RRT. In this case, the path safety of RRT would be a approximately 0.5 m less than DVG+A*, but, this difference could increase if the algorithm had found a path that goes inside one, or more, groups of robots. For example, Fig. 19 shows that there are some situations where the path safety of DVG+A* can be around 4 m greater than RRT, these are most likely the cases where the algorithm

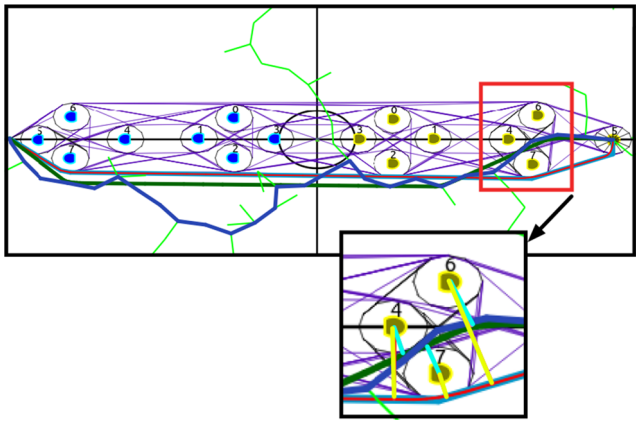

Fig. 24 Example of an RRT path less safe than DVG+A*. The light green lines are the RRT Tree, the purple lines are the DVG, the red lines are the DVG+A* path, the dark blue and dark green lines are the pre and post processed RRT paths, respectively. The yellow lines are the contribution of the respective robot to the path safety of DVG+A*, and the cyan lines are the contribution of the respective robot to the path safety of RRT

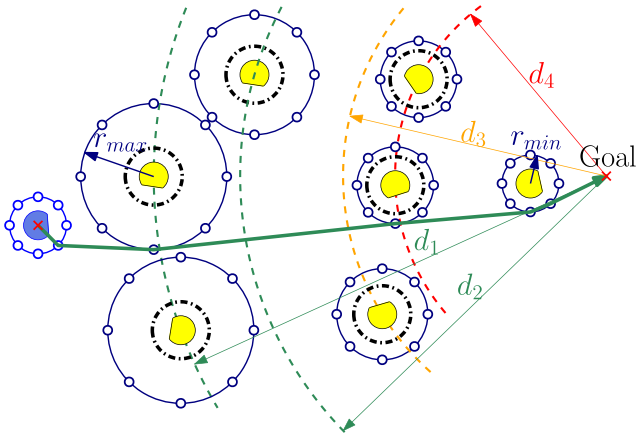

Fig. 25 Variable object radius example

had found a path that goes inside all blocks of obstacles in the scenario. The results show that this is the most probable reason for this behavior, since, in the scenarios where there was the possibility of a path between large groups of robots, i.e., worst case for DVG+A* and game stoppage scenarios (Figs. 11 and 14), the RRT had cases of a low path safety.

Still, these tests showed that the path safety parameter of the DVG+A* is usually much lower when compared to RRT. It is also worth to mention that, although in general DVG+A* performed worse than RRT in this parameter, this could be improved by increasing the radius r of the objects in the DVG. Another improvement that could be made is in the modeling of the obstacles in the graph. Instead of having a constant radius, to make it change according to some parameters, like, distance from the goal, obstacle velocity and others. With a change like that, it should be possible to make the algorithm maintain a safer distance from the obstacles, while not increasing the path length drastically. There are certainly some cases where the algorithm would benefit if it wasn't constrained to a fixed object radius to generate the graph. In Fig. 25 there is an example of how that would work by having the object radius to change according to the distance from the goal position. In this case, objects closer than d_4 don't get their radius changed, this is done to not obstruct the goal. Also, the object radius only increases up to a maximum $r_{\max}$, to prevent having objects being too big. The dark dash dot circles represent the standard object radius (r_{ally} or $r_{opponent}$). Similarly to the grouping technique, there would be the need of a condition where the radius could not be increased so much that it would obstruct the start or goal points, as this would interfere on the algorithm completeness.

Regarding the dynamic scenario, the results are much more similar between the algorithms. With respect to the individual computational time, each algorithm performed better in different scenarios, however, in general, the accumulated computational time was lower for DVG+A*. The time that it took the robot to drive from the initial to the goal position was similar in the first scenario, but, in the second scenario DVG+A* had a lower time. What is interesting about this result is that, even though DVG+A* made the robot collide more times than RRT in the second scenario (30 collisions), it still managed to drive the robot faster.

Note that, there are attempts in the SSL community to have a path planning algorithm to generate a path according to dynamic variables of the environment, e.g., in [20], the proposed path planning algorithm doesn't invalidate a path if the collision happens in a time instant greater than 3 s in the future, due to the fact that when this instant comes, the obstacle will probably already have moved to another position. However, as the author says, the paths must usually be generated in a 10 ms window, to allow the algorithm to recalculate the path in case some obstacle gets too close.

Another helpful insight that the number of collisions gives is that, considering solely the distance of the objects from the path, doesn't precisely indicate the path safety, e.g., in practically all the static experiments, RRT paths had a higher path safety than DVG+A*, however, in the dynamic experiments both algorithms had a similar number of collisions.

In another words, the path safety parameter, defined initially for the static experiments, doesn't measure at all how safe a path actually is when there are fast, and unpredictable, moving obstacles. In this case, another metric should be implemented to try to measure the actual safety. However, defining such metric in a way that it could be useful and valid for dynamic environments, would need some particular research, given the already mentioned

complexity of this environment. For example, the rules defined in [2] could be a great starting point for solving this problem, however, the development and adaptation of such rules for the SSL can be the core of a completely new algorithm, hence, it could be something to do in a future work.

The number of collisions during the dynamic tests brings into the discussion the need of not only a path planning algorithm, but also an obstacle avoidance algorithm. According to [1], in order to a robot to navigate through the environment, there are two tasks involved, which can be separated in high-level and low-level tasks, these tasks are executed by the path planning and obstacle avoidance algorithms, respectively. The path planning algorithm is responsible for generating a high-level path that guides the robot through the environment while considering many parameters, such as, path length, energy consumption and others. The obstacle avoidance algorithm is the one responsible for avoiding obstacles that may have gotten into the path after it was originally generated. Also, this is a low-level task due to it having to handle variables that are more dynamic than what the path planning algorithm usually handles. In this work, an obstacle avoidance algorithm would be the one that makes the robot dodge another incoming robot that might be at a high speed.

However, it is important to keep in mind that these path planning algorithms (RRT and DVG+A*) do avoid collisions at the planning step, their generated path will never cause a collision if the obstacles don't move after the planning has been made. But, when the path execution begins, and the obstacles start moving from their original position, the algorithms just can't properly avoid these obstacles, specially if they are already close to the robot. In summary, this means that the algorithms can't avoid the collisions long before they happen, this could only be done, to some extent, if some kind of prediction of the obstacles position were available.

Throughout the dynamic experiments, it was observed that the collisions usually happened when a object was moving perpendicular to the robot path, in that case, the algorithms have no way of trying to dodge the obstacle. This is a situation were a obstacle avoidance algorithm would work well.

In conclusion, even though in the proposed tests the robot collided a lot with the obstacles, the scenarios built for the dynamic tests are much more cluttered than what usually happens in a SSL match. As it can be seen on Table 3, during most of the time the robots are practically alone or with a second robot within a radius of 500 to 1500 mm, that means that the obstacles are quite sparse within the environment, thus, it should be easier to avoid them.

In fact, Table 13 shows the total number of collisions that one robot had during a few games of the last competition.

Table 13 Total number of collisions during the last competition

in-Game Oponnent	Total Collisions
RoboIME	154
RoboCIn	287

Note that the games last for around ten minutes each, thus, the in-game results are quite satisfying, considering that there are thousands of paths calculated during the whole match. This means that the number of collisions is much smaller when put into the same proportion as the number of samples for the tests performed. Also, even though it might seem that there were a lot of penalties due to the number of collisions in these matches, the game referee considers the velocity of the collision and if robot A collided with robot B or the other way around. Therefore, since this kind of sensibility wasn't implemented when detecting the number of collisions in the tests and in the script that analyzed these games, the results regarding number of collisions are higher than what would be detected in a real match.

8 Conclusions

This work has shown the application of the DVG+A* algorithm in the SSL environment, as well as, pointing out some of its weaknesses and proposing a few changes that increased the DVG computational time performance, e.g., the obstacle grouping technique.

Several experiments were made in order to compare both algorithms. These experiments evaluated the algorithms regarding its performance on both static and dynamic environments. Furthermore, the results in the static environment showed that the DVG+A* is capable of outperforming RRT when comparing the computational time and path length. The only parameter where RRT outperforms DVG+A* is the path safety. However, there are ways to improve this on the DVG+A*.

In the dynamic experiments, both algorithms performed similarly in all analysed parameters. However, these experiments showed that the algorithms are not capable of avoiding collisions properly, thus, showing the importance of also having an obstacle avoidance algorithm. In fact, all experiments would have to be redone, and the results reevaluated, when such algorithm gets implemented. This change would certainly impact the performance of both algorithms, therefore, the conclusions could change significantly.

In conclusion, there's two important points, the first, considering static, or sparse dynamic environments, the DVG+A* has a slightly better performance than RRT. It has a low computational time, is capable of finding short paths

consistently and the path safety can be improved without sacrificing the completeness of the algorithm. The second, considering cluttered dynamic environments, in this case, both algorithms aren't suitable for avoiding collisions, to improve this, either having predictions about the obstacles positions, or implementing a dedicated obstacle avoidance algorithm, would be necessary.

In a future work, it would be interesting to see the effect of having a dedicated obstacle avoidance algorithm working together with the path planner. The work seen on [11] provides a thorough survey on obstacle avoidance algorithms, which could be a great starting point for future works on this subject in the SSL. And last but not least, investigating proper metrics to measure the safety of paths when in cluttered dynamic environments is certainly something that could produce great results.

Author Contributions Leonardo da Silva Costa implemented the algorithms, designed and performed the experiments, analyzed the data and wrote the manuscript. Flavio Tonidandel provided critical feedback, supervised and shaped the project, research, analysis and manuscript.

Funding This work was supported by University Center of FEI and it is based on a scientific project on mobile robots under funding number PBIC133/17.

Availability of Data and Material The authors confirm that the data supporting the findings of this study are available within this article and in the following repository: https://bit.ly/34IM5qe. All images shown in the article were made by the authors and can also be found at the repository.

Declarations

Conflict of Interests The authors declare that they do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

References

1. Ari, M.: Elements of Robotics SpringerOpen. Cham, Switzerland (2018)
2. Ayawli, B.B.K., Mei, X., Shen, M., Appiah, A.Y., Kyeremeh, F.: Mobile robot path planning in dynamic environment using voronoi diagram and computation geometry technique. IEEE Access **7**, 86026–86040 (2019)
3. Blanco-Claraco, J.L.: C++11 header-only library for building kd-trees. https://github.com/jlblancoc/nanoflann, Accessed 12 Aug 2020 (2020)
4. Bondy, J.A., Murty, U.S.R., et al: Graph theory with applications, vol. 290. Macmillan London (1976)
5. Brandt, D.: Comparison of A* and RRT-connect motion planning techniques for self-reconfiguration planning. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 892–897. IEEE (2006)
6. Braun, J., Brito, T., Lima, J., Costa, P., Nakano, A.: A comparison of A* and RRT* algorithms with dynamic and real time constraint scenarios for mobile robots, pp. 398–405, https://doi.org/10.5220/0008118803980405 (2019)
7. Bruce, J., Veloso, M.M.: Real-time randomized path planning for robot navigation. In: Robot soccer world cup, pp 288–295. Springer (2002)
8. Bruce, J.R.: Real-time motion planning and safe navigation in dynamic multi-robot environments. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science (2006)
9. Du, W., Islam, F., Likhachev, M.: Multi-resolution A*. arXiv:200406684 (2020)
10. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
11. Hoy, M., Matveev, A.S., Savkin, A.V.: Algorithms for collision-free navigation of mobile robots in complex cluttered environments: A survey. Robotica **33**(3), 463–497 (2015)
12. Huang, H.P., Chung, S.Y.: Dynamic visibility graph for path planning. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566), vol. 3, pp. 2813–2818. IEEE (2004)
13. Kaludjer, H., Brezak, M., Petrović, I.: A visibility graph based method for path planning in dynamic environments. In: 2011 Proceedings of the 34th International Convention MIPRO, pp. 717–721. IEEE (2011)
14. Kuffner, J.J., LaValle, S.M.: RRT–Connect: An efficient approach to single-query path planning. In: Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, vol. 2, pp. 995–1001. IEEE (2000)
15. LaValle, S.M.: Rapidly-Exploring Random Trees: A New Tool For Path Planning. Citeseer (1998)
16. Lin, Y., Saripalli, S.: Sampling-based path planning for uav collision avoidance. IEEE Trans. Intell. Transp. Syst. **18**(11), 3179–3192 (2017)
17. MacDougall, M., Ellis, G., Hashemi, A., Jackson, E., Lip, O.C., Ivanov, N., Petrie, J., Tonks-Turcotte, K., Sousa, C., Lai, K., Xu, B., Li, Q., Goto, E., Deutsch, D., Lee, A.B.M.: 2018 Team Description Paper: UBC Thunderbots (2018)
18. Nash, A., Koenig, S.: Any-angle path planning. AI Mag. **34**(4), 85–107 (2013)
19. Noreen, I., Khan, A., Asghar, K., Habib, Z.: A path-planning performance comparison of RRT*-AB with MEA* in a 2-dimensional environment. Symmetry **11**(7), 945 (2019)
20. Ommer, N., Ryll, A., Geiger, M.: TIGERs Mannheim (2019)
21. Poudeh, A.G., Mohammadi, H.B., Hosseinikia, A., Esmaeelpourfard, S., Adhami-Mirhossein, A.: MRL extended team description 2016. In: Proceedings of the 19th International RoboCup Symposium (2016)
22. RoboCup, S.SL.: Rules of the small size league. https://ssl.robocup.org/rules/, Accessed 19 Jul 2019 (2011)
23. Rodríguez S., Rojas, E., Pérez, K., López, J., Quintero, C., Calderón, J.: Fast path planning algorithm for the RoboCup Small Size League. In: Robot Soccer World Cup, pp. 407–418. Springer (2014)
24. da Silva Costa, L., Tonidandel, F.: Comparison and analysis of the DVG+A* and rapidly-exploring random trees path-planners for the Robocup-Small size league. In: 2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE), pp. 1–6. IEEE (2019)
25. Šišlák, D., Volf, P., Pechoucek, M.: Accelerated A* trajectory planning: Grid-based path planning comparison. In: 19th International Conference on Automated Planning and Scheduling (ICAPS), pp. 19–23. Thessaloniki, Greece, Citeseer (2009)

26. Urmson, C., Simmons, R.: Approaches for heuristically biasing RRT growth. In: Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), vol. 2, pp. 1178–1183. IEEE (2003)
27. Vemula, A., Muelling, K., Oh, J.: Path planning in dynamic environments with adaptive dimensionality. arXiv:160506853 (2016)
28. Wang, J., Chi, W., Li, C., Wang, C., Meng, M.Q.H.: Neural RRT*: Learning-based optimal path planning. IEEE Trans. Autom. Sci. Eng. (2020a)
29. Wang, J., Meng, M.Q.H., Khatib, O.: EB-RRT: Optimal motion planning for mobile robots. IEEE Trans. Autom. Sci. Eng. (2020b)
30. Wei, K., Ren, B.: A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm. Sensors **18**(2), 571 (2018)
31. Welzl, E.: Constructing the visibility graph for n-line segments in O(n^n) time. Inf. Process. Lett. **20**(4), 167–171 (1985)
32. Yang, K., Sukkarieh, S.: An analytical continuous-curvature path-smoothing algorithm. IEEE Trans. Robot. **26**(3), 561–568 (2010)
33. Yang, S., Baum, M.: Extended Kalman filter for extended object tracking. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4386–4390. IEEE (2017)
34. Zickler, S., Bruce, J., Biswas, J., Licitra, M., Veloso, M.: CMDragons 2009 extended team description. In: Proc. 14th International RoboCup Symposium, Singapore (2010)

Leonardo da Silva Costa is a Control and Automation Engineering student at the Centro Universitário FEI (2021), with a technician degree in Mechatronics from the SENAI institution. He is a student at Centro Universitário FEI and has experience with mobile robots and control systems. He is a member of the research group in Robotics called RoboFEI. He is a member of the Technical Committee of the international RoboCup Federation, and a Co-chair of the RoboCup Brasil Small-Size League (2020). His main areas of interest are: Planning, Motion Control, Autonomous Mobile Robots and Data-based modeling.

Prof. Flavio Tonidandel is graduated in Electrical Engineering from the Federal University of Uberlandia (1996), with a master's (1999) and doctor's (2003) degrees in Electrical Engineering from the Polytechnic School of USP. He is a professor of Centro Universitario FEI and has experience in Computing and Electrical Engineering, with emphasis on Artificial Intelligence and Robotics. Since 2003, he has participated and coordinated the research group in Robotics called RoboFEI. He is a member of the Board of Trustees of the international RoboCup Federation, and a member of the Board of Trustees of RoboCup Brasil, the entity of which he was president from 2016 to 2019. He coordinated the OBR - Brazilian Robotics Olympics in 2013 and 2014, and the Special Robotics Commission (CER) of the Brazilian Computer Society between 2010 and 2014; He was Co-General Chair of RoboCup 2014 held in Brazil; Member of the IEEE-RAS Latin American Robotics Committee; Organized and coordinated several symposia, congresses, and scientific robotics competitions. He has papers published in the main areas of interest: Planning, Case-Based Reasoning, Intelligent Home Automation, Intelligent Human-Robot Interaction, and Autonomous Mobile Robots.