Comparison of Sampling-based Path Planners for Robocup Small Size League

Conference Paper · December 2020

DOI: 10.1109/LARS/SBR/WRE51543.2020.9307121

3 authors:

Felipe Celso Reis Pinheiro
Instituto Tecnologico de Aeronautica
4 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Marcos Maximo
Instituto Tecnologico de Aeronautica
68 PUBLICATIONS **155** CITATIONS
SEE PROFILE

Takashi Yoneyama
Instituto Tecnologico de Aeronautica
235 PUBLICATIONS **2,564** CITATIONS
SEE PROFILE

Comparison of sampling-based path planners for Robocup Small Size league

Felipe C. R. Pinheiro[1] , Marcos R. O. A. Máximo[1] and Takashi Yoneyama[2]

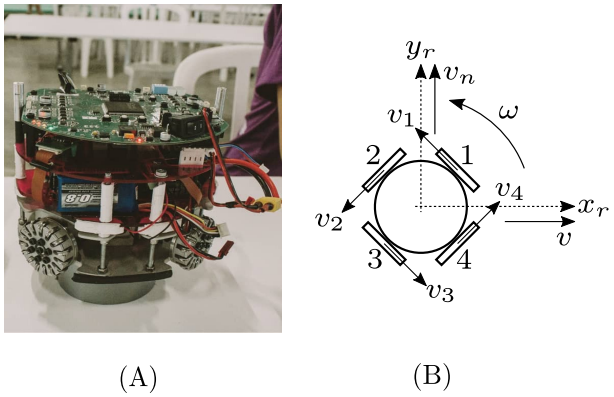
***Abstract*— The path planning problem is an important issue in robotics. In the context of the Small Size Soccer league, a widely used algorithm for path planning is the Rapidly-Exploring Random Tree (RRT), which is a single-query planner. However, properties such as *asymptotic optimality* and *anytime resolution* may not be achieved by some of the algorithms used in the competition. Some of the possible path planning solvers with some of these properties are the Batch Informed Trees (BIT*), Fast Marching Trees (FMT*), and other RRT implementations (such as Extended Execution RRT - ERRT and RRT*). The contribution of this paper is to carry out a benchmark comparative investigation with a selected set of path planners found in the literature, evaluating their performance in the context of Robocup Small Size League Competition. The results show good convergence properties of BIT* when the planning time requirement is larger. ERRT performs well to find a solution, but it falls short concerning the solution quality. RRT* did not have a good convergence to optimal paths, but succeeded to find paths with short planning time. Since FMT* does not present *anytime resolution*, the algorithm did not present a satisfactory convergence with larger planning time. To improve the performance of the planners, it may be interesting to conceive heuristic strategies to allow trivial problems to be solved faster.**

I. Introduction

The ITA (Aeronautics Institute of Technology) robotic soccer team started to partake in the Small Size League (SSL) and this paper describes the developments associated with our team of omni-directional four-wheeled robots.

For this paper, we developed simulations in the environment of the Robocup Small Size Soccer League (F-180) division B involving 6 robots for each team. The robots' horizontal size can not exceed a 180mm diameter circle and the robot's height is limited. The game takes place in a 9 m long and 6 m wide field [14].

The system we use for the competition comprises 6 omni-directional robots (as in Figure 1-(A)) with an external computer connected to a vision system (4 cameras provided by the competition) and with a decision-making algorithm. The computer communicates with robots via a radio link. The robot receives a reference velocity command, and an embedded controller attempts to track the reference velocity. This robot was not in the experiments in our paper, but it is our motivation to investigate the path planning problem in this competition.



(A) (B)

Fig. 1. (A): The actual robot and (B): Top view of the four-wheeled robot with indication of the robot's local coordinate frame. The frame is defined with the front of the robot aligned with the x_r axis.

An important issue in the SSL soccer is the path planning problem, as a good performance in games requires well-positioned robots in strategic positions. For instance, to receive passes and shoot the ball, or to defend against the opponents' attacks. In this context, the ability to move in an omni-directional manner is a key attribute, which makes it easier for the robot to follow paths without some kinematic constraints. The external computer typically runs a global motion planning algorithm. The global planner uses the information provided by the camera to generate a sequence of path waypoints. The path tracker also generates velocity commands to be imposed while following the waypoint sequence.

The path planning problem is usually solved by two major approaches: graph-search and sampling-based algorithms. Graph-search approaches (e.g. A* and Djisktra algorithms) can optimally solve the graph search problem. These algorithms rely on state-space discretization to find the minimum cost path. However, the computing cost to find good quality path solutions may drastically increase as the dimension of the state space increases.

Sampling-based algorithms scale better for high dimension problems than graph-search approaches. This happens since such algorithms avoid the entire state space discretization problem by sampling states from it. Some sampling-based approaches build graphs out of state-space samples [8], [10], [12] and solve the underlying graph search problem.

As the path search problem becomes probabilistic, the sampling-based algorithms have different properties than

[1]Felipe Pinheiro and Marcos Máximo are with the Autonomous Computational Systems Lab (LAB-SCA), Computer Science Division, Aeronautics Institute of Technology, São José dos Campos, SP, Brazil. fcrpfelipe96@gmail.com, mmaximo@ita.com

[2]Takashi Yoneyama is with the Electronics Division, Aeronauticstakashi@ita.br

978-0-7381-1153-7/20/$31.00 ©2020 IEEE

graph-search algorithms. They can be considered *probabilistic complete* if the probability to find a path, if one exists, becomes one when the amount of state-space samples goes to infinity. They can also be *anytime* with *incremental search*. Anytime algorithms can return a valid path solution even if the optimal path has not been found at the moment of the query. An incremental search attempts to improve the path solution at each iteration. Moreover, they can be considered *asymptotically optimal* when the path solution found converges to the optimal solution as the number of samples goes to infinity.

The SSL teams have solved the path planning problem using a variety of techniques. Some teams use sampling-based algorithms, e.g. Rapidly Exploring Random Trees (RRT) [17] and Extended RRT (ERRT) [4], which suits the problem but are not asymptotically optimal algorithms. Custom solutions are also used, e.g. in [16]. Although having good results in the SSL Robocup competition, the afore mentioned custom solution do not have, to the best of our knowledge, any documented mathematical proof of its completeness or optimality. Moreover, [5] compares a Visibility Graph (VG) based path planner with RRT performance in SSL environments, obtaining good results for the first one. However, [5] did not compare the planner with other famous planners used in the competition such as ERRT, and with more recent sampling-based planners as done in this paper.

Other planners in the literature were developed to enhance the sampling-based algorithm properties. The contribution of this paper is to analyze the performance of the algorithms ERRT, RRT, RRT*, Fast Marching Trees (FMT) [10] and Batch Informed Trees (BIT*) [8] for the SSL path planning problem.

The outline of this paper follows: Section II summarizes the theoretical background, Section III details the simulations, Section IV shows the obtained results and discuss them, Section V presents the major conclusions and Section VI shows our acknowledgements.

II. Background

A. Path planning problem

In a given path planning problem, consider χ the state space of available geometric states. The space χ may contain obstacles, which occupy the set of states χ_{obst}. Therefore, the free space is $\chi_{\text{free}} = \chi \setminus \chi_{\text{obst}}$. The path planning problem consists into finding a feasible path as a continuous function of time $\sigma(t) : [0,1] \rightarrow \chi_{\text{free}}$ connecting a given initial state $\mathbf{x}_{\text{init}}$ to a goal state $\mathbf{x}_{\text{goal}} \in \chi_{\text{goal}}$. It is possible to cast this problem into an optimal path planning form. Let Σ be the set of all paths connecting pairs of different states of χ. The function $c : \Sigma \rightarrow \mathbb{R}$ be the length function defined for the state space χ. The optimal path (σ^*) that connects $\mathbf{x}_{\text{init}}$ and $\mathbf{x}_{\text{goal}}$ is defined as

$$\begin{aligned} \sigma^* = \arg\ \min\{c(\sigma) | \sigma(0) = \mathbf{x}_{\text{init}}, \\ \sigma(1) = \mathbf{x}_{\text{goal}}, \forall t \in [0,1], \sigma(t) \in \chi_{\text{free}}\}. \end{aligned} \tag{1}$$

B. Random Geometric Graphs

Considering the path planning problem of moving from a start $\mathbf{x}_{init} \in \chi_{\text{free}}$ to $\chi_{goal} \subset \chi_{free}$, let a set of random states $X_{samples}$, such that $X_{samples} \subset \chi$. An implicit graph can be constructed from $X_{samples}$, in such a way that its edges can be determined heuristically. This type of graph, also known as Random Geometric Graph (RGG) [13], is widely used in sampling-based path planning algorithms. An RGG can be categorized by the way that its edges are built:

- *k-nearest graph* (k-NG) if each of its nodes have edges connecting it to the nearest k nodes.
- *r-disc graph* (r-DG) if each of its nodes have edges connecting it to all other nodes that are up to a distance of $r > 0$.

Two examples of the RGGs principles are illustrated in Figure 2.

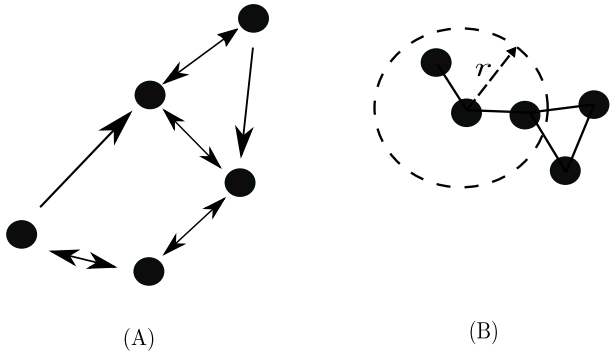


Fig. 2. (A) is a 2-NG and (B) is an r-DG.

For each node $\mathbf{x}$ of an RGG there is associated a *cost to go*. This cost is equivalent to the length of the shortest path connecting $\mathbf{x}_{\text{init}}$ and $\mathbf{x}$. The algorithms RRT*, FMT, and BIT* make use of the RGG theory.

C. RRT*

The classical RRT incrementally builds a tree of feasible paths. The tree nodes are states from the state space χ_{free}. This algorithm gets a feasible path as a solution. However, the path obtained is suboptimal with probability one [11]. The RRT* enhances the RRT algorithm by introducing a *rewire* and *chooseParent* function. Algorithm 1 depicts the RRT*. The RRT* expands a tree τ, whose first node is the initial state $\mathbf{x}_{\text{init}}$, by sampling other positions in χ_{free}. The function *sampleSpace* gets the sample $\mathbf{x}_{\text{random}}$. The function *nearestNeighbor* finds the nearest neighbor $\mathbf{x}_{\text{nearest}} \in \tau$ of $\mathbf{x}_{\text{random}}$. After that, the function *selectInput* chooses a control input to drive the system from $\mathbf{x}_{\text{nearest}}$ to $\mathbf{x}_{\text{random}}$. This control input is propagated (using the state transition equation $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$) by a time interval of Δt. The propagation generates the new node $\mathbf{x}_{\text{new}}$ to be added to the three. All the steps described so far are derived from the classic RRT. As opposed to RRT, that straightly adds the new node to τ as defined by the function *newState*, RRT* initially looks for nodes inside a ball of radius r around $\mathbf{x}_{\text{new}}$. The algorithm

Algorithm 1: RRT*

procedure $generateRRT(\mathbf{x}_{\text{init}}, K, \Delta t, \chi_{\text{free}}, \chi_{\text{goal}})$
  $\tau.insertNode(\mathbf{x}_{\text{init}})$
  for $k = 1$ to K
    $\mathbf{x}_{\text{random}} \leftarrow sampleSpace(\chi_{\text{free}}, \chi_{\text{goal}})$
    $\mathbf{x}_{\text{nearest}} \leftarrow nearestNeighbor(\mathbf{x}_{\text{random}}, \tau)$
    $\mathbf{u} \leftarrow selectInput(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{random}})$
    $\mathbf{x}_{\text{new}} \leftarrow newState(\mathbf{x}_{\text{nearest}}, \mathbf{u}, \Delta t)$
    $\mathbf{x}_{\text{min}} \leftarrow chooseParent(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})$
    $\tau.insertNode(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{min}})$
    $\tau.rewire(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{min}})$
  end for
  return τ
end procedure

attempts to find nodes in this set such that if added as a father of $\mathbf{x}_{\text{new}}$ produce shorter paths (starting from $\mathbf{x}_{\text{init}}$ and leading to $\mathbf{x}_{\text{new}}$), as depicted in Figure 3-(C). The node which produces the shortest path ($\mathbf{x}_{\text{min}}$) is added as father of $\mathbf{x}_{\text{new}}$ in τ (Figure 3-(D)). Besides, the algorithm has a function *rewire* to check if rewiring $\mathbf{x}_{\text{new}}$ as a father of each one of its neighbors produce a shorter path. If yes, $\mathbf{x}_{\text{new}}$ is added as the father of the corresponding neighbor that generates the shortest path (Figure 3-(E)).

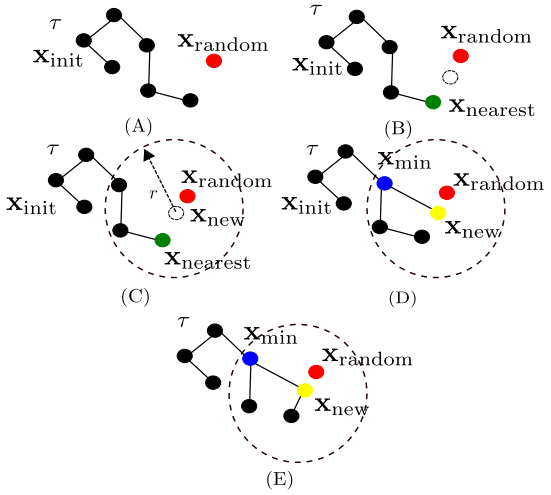


Fig. 3. RRT* steps. (A) shows the sampling of a position $\mathbf{x}_{\text{random}}$ in a 2D space. In (B) the nearest neighbor $\mathbf{x}_{\text{nearest}}$ of $\mathbf{x}_{\text{random}}$ is found and $\mathbf{x}_{\text{new}}$ is generated. (C) and (D) represents the *chooseParent* step. (E) depicts the *rewire* step.

This algorithm has two important parameters, the *range length* and the *goal bias*. The range length corresponds to the distance between parent-child nodes of the tree, used to generate $\mathbf{x}_{\text{new}}$. The goal bias is the probability to sample $\mathbf{x}_{\text{goal}}$ from the state space, i.e. $\mathbf{x}_{\text{random}} = \mathbf{x}_{\text{goal}}$. The reader is referred to [11] for further details.

D. Extended Execution RRT (ERRT)

The *Extended Execution RRT* (ERRT) was presented in [3] as a good path planner for SSL Robocup competitions, since it exploits the search space by reusing previously found path waypoints to expand the search tree. The difference between RRT and this variant is on the way that the function *sampleSpace* is implemented. There is one additional parameter called *waypoint bias*. This bias represents the probability of sampling a waypoint from a previously found solution. In [3] they claim that 1) in real time problems it is often better to execute a bad plan than to stand still while looking for better ones and 2) the quality of the solution does not matter as long as there are no position discontinuities. The first proposition is still disputed since in [3] there is no evidence that bad path quality does not affect very much the performance. Nevertheless, algorithms with anytime resolution can fulfill the requirement of improving the solution quality within short planning times, as it will be seen in Section IV. The reader can find further details in [3].

E. FMT*

The algorithm *Fast Marching Trees* (FMT*) [10] attempts to optimally solve a path planning problem using an r-DG defined in χ. Thus, the solution is to find the shortest path connecting $\mathbf{x}_{init}$ to a node in the goal region χ_{goal}.

This algorithm builds an r-DG using samples from χ_{free}. After evaluating the cost to go to each node, a graph-search algorithm attempts to find the shortest feasible path connecting the initial node and χ_{goal}. The biggest advantage of this method consists in using a lazy collision check strategy. This strategy reduces the number of collision checks, substantially reducing the algorithm execution time. For more details on the algorithm a good source is [10].

F. BIT*

The BIT* algorithm [8] relies on RGG theory and heuristics to improve the planning execution time when compared to RRT* and FMT*. This algorithm uses the concept of *batches*. A batch is a set of n random states of χ_{free} as nodes of an RGG (it could be either k-NG or r-DG). To each batch, the algorithm searches for the shortest path connecting $\mathbf{x}_{\text{init}}$ and a node inside of χ_{goal}. The main idea is to reduce the computation cost of this search by restricting it to states inside elliptical boundaries. The shorter the found path solution is, the shorter is the elliptical search region [8], [9].

The algorithm BIT* iteratively increases the number of samples χ_{free} inside the boundary. This creates denser RGGs, reducing the expected average distance between the nodes. The found solutions are iteratively optimized during the execution, as shown in Figure 4.

III. Simulation Experiment

The Open Motion Planning Library (OMPL) [15] RRT, RRT*, BIT* and FMT* implementations were used adapting only the parameters for each planner. The ERRT algorithm was implemented using the base RRT provided by OMPL. All the simulations were performed in a computer with an Intel Core i7 3.8 GHz processor, 8 GB RAM memory and Ubuntu 18.04 Operating System, using C++ language.

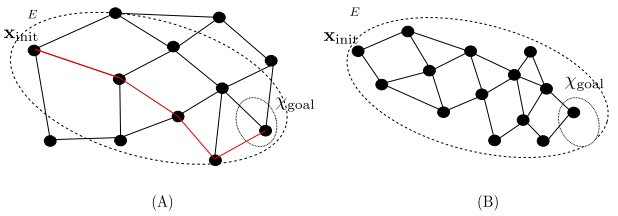


Fig. 4. Iteration of a *batch* of states. In (A) the shortest path found (in red) with this batch will define a new ellipse E. The region within E will be the search space for the next iteration. The nodes outside of the ellipse shall not be considered for the search. In (B) this iteration the batch was built using only states within the ellipse.

A simulation environment was created to emulate the path planning problem for the SSL league. In this simulation the agent robot starts on a random position $\mathbf{x}_{\text{init}}$ inside the initial region. The static obstacles (9 robots, excluding the 2 goalies) are randomly placed inside the obstacle region. Finally, the goal $\mathbf{x}_{\text{goal}}$ is randomly placed inside a goal region. The role of the robot is to find feasible paths in this environment. The figures of merit consisted of the *success probability* of finding a feasible path and the *average path length* of the found path, given a fixed maximum allowed computational time. A visual representation of the adopted arrangement is shown in Figure 5. In the simulations, the obstacles were circles of 90mm.

The success probability is estimated using a Monte Carlo simulation with 10000 iterations for each given configuration.

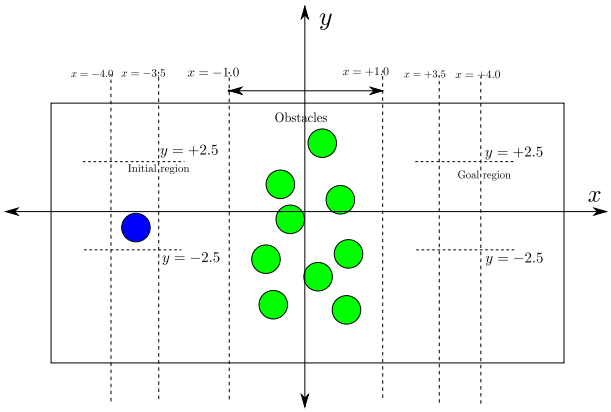


Fig. 5. Field representation of initial region, obstacles and goal.

IV. Results and Discussion

The code that represents the study done in this section can be found in [6].

A. Parameter tuning

FMT* and BIT* build graphs to solve the path planning problem. The amount of *RGG samples* used to build the graphs is an algorithm's parameter. The more samples the algorithm uses, the larger is the algorithm's computational cost. However, the number os samples can potentially improve the quality of the solution.

This parameter was tuned considering the simulation depicted in Figure 5, fixing the maximum planning time allowed. To estimate the success probability for a given amount of samples, it was considered that other algorithms used in the competition (such as decision making and control methods) have a negligible computational cost comparing to path planning. For the estimation of the maximum allowed planning time, the global camera's sample time [1] was assumed evenly divided for the planning time across the team's robots. In our team, as the goal keeper does not perform any path planning task and considering an additional thread to sequentially run the path planning solvers, the camera sample time is divided by five robots. Therefore, the maximum planning time would be

$$t_{max} = \frac{1}{5(\text{FPS})}, \tag{2}$$

where FPS is the camera's Frames Per Second. The Figure 6 depicts the estimated probabilities.

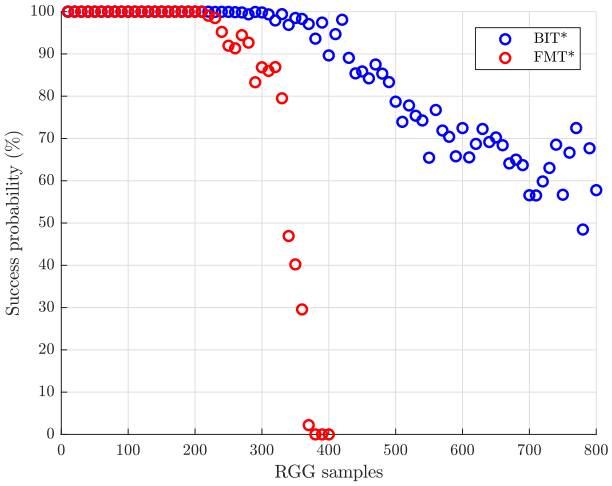


Fig. 6. Success probability vs # of RGG samples

For this parameter, 100 samples was adopted as the number of samples for both algorithms, since the number lies on the interval with a 100 percent success rate with a reasonable clearance.

One can also change the RRT range length. It was chosen in the same order of magnitude of the obstacles' size. As per the SSL rules, the maximum admissible size for obstacles is 18mm. Therefore, the range length of 15mm was chosen [14].

The RRT goal bias was estimated using the same Monte Carlo simulation used to evaluate the success probability. For this, the same maximum planning time presented in (2) was used. The threshold of seventy percent was adopted as an acceptable value for the problem after testing a wide range of values. For the ERRT waypoint bias, the best value was 20 percent.

B. Path planners benchmark

The results of the benchmark tests for the referred algorithms are presented in this subsection. For each random

configuration, (as in Figure 5) of obstacles in the middle field, ten thousand iterations were carried out, varying the maximum allowed planning time. As this benchmark does not require re-plannings, ERRT was not included in this comparison since for a single path planning iteration it behaves exactly as the RRT.

The first benchmark was comparing computing cost performance within the problem. Figure 7 shows the success probability estimation.

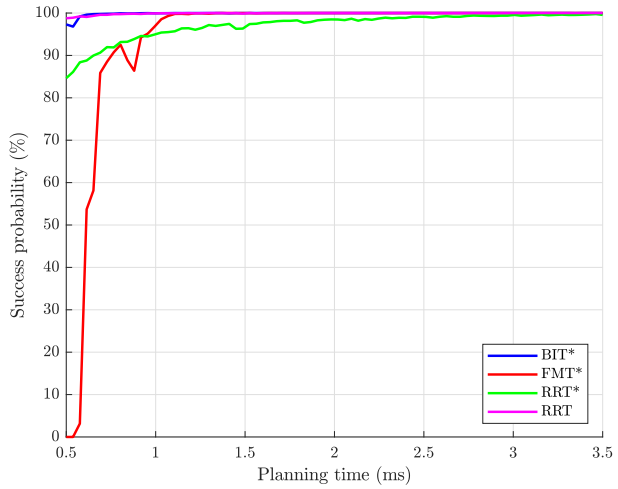


Fig. 7. Success probability vs maximum allowed path planning time

For this first comparison, BIT* outperforms FMT* with shorter planning times. Many reasons can be stressed for this, but one important feature BIT* has is the informed search. As the search space is reduced at each new solution, samples outside of the search space do not improve it. Therefore, these samples must not be considered for the path search, if the objective is to search for a better solution. Also, the data structures BIT* were optimized for practical applications [8].

The RRT* scheme outperforms FMT* with a shorter planning time due to its greediness. The larger the goal bias, the more greedy is RRT*. On the other hand, by doing so, RRT* eventually loses path quality. Moreover, the planning time range is not large enough to see the asymptotic convergence of RRT*. It is also widely known in the literature the poor performance of RRT* when it comes to convergence time [2], which is also confirmed in Figure 8.

The FMT* algorithm has poor performance during a short time. After that, there is an abrupt improvement (from one millisecond on in the tested cases). This happens since the algorithm attempts to solve a graph search for a constant amount of nodes. There is a minimum average computational cost to solve this random graph. As the algorithm does not have anytime resolution, the path quality is expected to not vary much after the initial computational cost.

The second comparison reasons about the average path length created by each algorithm for a given maximum planning time. Figure 8 shows the benchmark. The algorithms have a good performance initially since the solutions found must be shorter so that the algorithm succeeds in the time given. However, when the planning time goes up to one millisecond, BIT* and RRT* performs worse than with shorter planning time. This happened because worse solutions can now be successfully found within time.

However, as time goes up, BIT* showed a better average convergence rate to the optimum than the other algorithms. This conclusion can also be reached when comparing the variance (Figure 9) of the path length found in the Monte Carlo simulation. The variance of the path length found by BIT* reduces as the planning time increases.

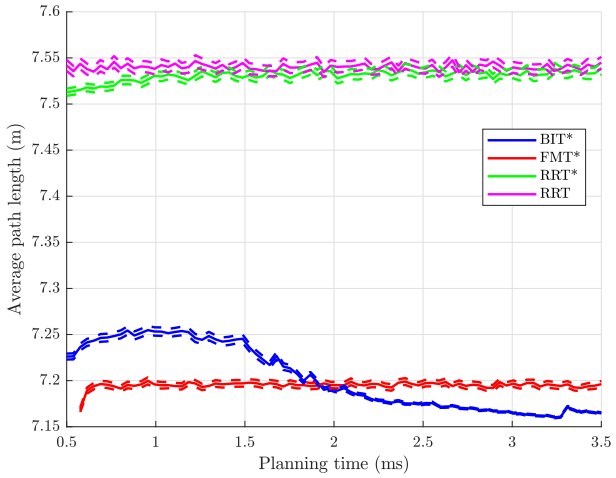


Fig. 8. Average path length vs maximum allowed planning time. Dashed lines represent upper and lower extremes of the 95% confidence interval.

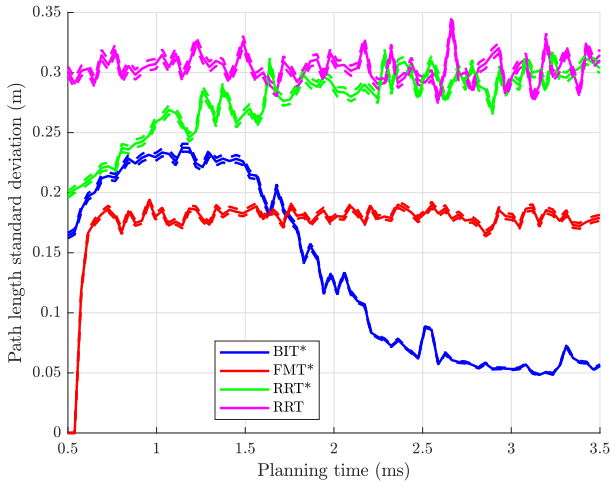


Fig. 9. Path length standard deviation vs samples per batch. Dashed lines represent upper and lower extremes of the 95% confidence interval.

C. Tracking performance evaluation

Finally, the path re-planning performance was compared in a tracking problem. An ideal tracking system was adopted for this comparison, controlling the position of the agent at each step, following the planned path. The ideal tracker was chosen to have a speed of 3 meters per second. The figure of merit is, for this benchmark, the distance from

the agent to the goal. In this comparative experiment, one random configuration was simulated 50 times. The result in Figure 10 shows the average distance to goal versus the re-planning iteration. The path planning algorithm had the same time limit to find a solution (given by (2)). The best results reached by BIT* and FMT* reflects their good convergence rate shown in the previous results.

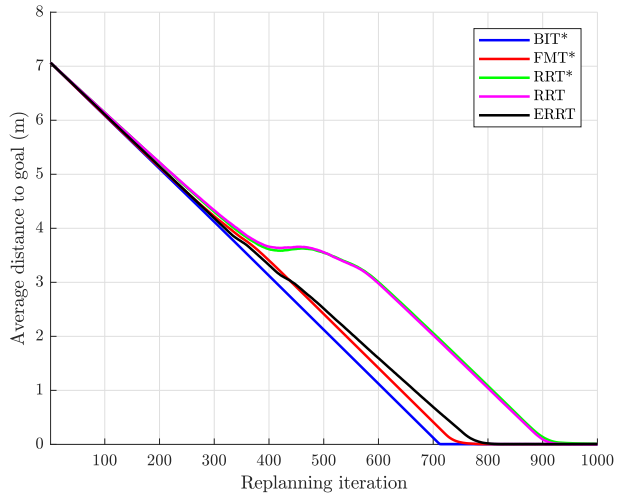


Fig. 10. Current distance to goal vs re-planning iteration.

V. Conclusion

The main contribution of this paper was to do a benchmark between the proposed path planners and evaluate its performance in the Small Size League. The results show good performance for BIT* with larger planning time, consistent performance for FMT* across a wide range of planning time values, and poor performance of RRT, RRT* and ERRT in the simulation environment created.

The BIT* planner has shown its versatility concerning computing cost, given that it was able to solve the problem across a wide planning time range. Moreover, the planner had a reasonable convergence rate with larger planning times.

The FMT* had a versatile performance with the path's quality, whereas having an average performance with the computing cost. After the minimum required computational cost to build the RGG, the performance does not change much since the algorithm is not of anytime resolution.

The RRT and RRT* had a poor performance on path quality and path length variance. This may be a bad choice to be used for path planning in the SSL environment. The larger the path length variance is, the larger is the instability of the path throughout the tracking. Therefore, as the robot system relies on path re-planning to handle the obstacle dynamics, the tracking is expected to have poor performance due to successive waypoint changes. Whereas the variant ERRT had better tracking performance by improving the state sampling.

With the benchmark, it was possible to notice the potential of using other algorithms e.g. FMT* and BIT* in the league, as more stable and versatile path planners. The versatility is justified since both planners have a good path planning performance with reasonable planning times, besides the good planning stability demonstrated with their tracking performance. Moreover, even in dynamic environments such as the SSL, the path solution quality has a major role on the tracking problem performance, as one could observe in Section IV regarding the evaluation of tracking performance. BIT*, with a better path convergence, clearly had a better performance as regards to the time of arrival.

Future works on these algorithms may apply heuristics to improve its performance on trivial problems, i.e. problems with a trivial solution e.g. a straight line path, and evaluate other features such as *sample removal* and *just-in-time sampling* [7].

VI. Acknowledgements

The authors thank the ITAndroids robotics team for the contribution to this paper and ITAndroids' sponsors: Altium, Cenic, Intel, ITAEx, MathWorks, Metinjo, Micropress, Polimold, Rapid, SolidWorks, STMicroelectronics, Wildlife Studios, and Virtual.PYXIS. We would like to thank FAPESP for the Master's Scholarship 2020/04561-0.

References

[1] Allied Vision. *Stingray Data Sheet F-046*, 8 2020.
[2] Oktay Arslan and Panagiotis Tsiotras. The role of vertex consistency in sampling-based algorithms for optimal motion planning. 04 2012.
[3] James Robert Bruce. *Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments*. PhD thesis, Carnegie Mellon University, 2006.
[4] C. S. Cosenza, G. Fernandez, and G. C. K. Couto. Roboime: From the top of latin america to robocup 2019. Technical report, Instituto Militar de Engenharia (IME), 2019.
[5] Leonardo Costa and Flavio Tonidandel. Comparison and analysis of the dvg+a* and rapidly-exploring random trees path-planners for the robocup-small size league. pages 1–6, 10 2019.
[6] C.R.P. Felipe. Ssl sampling based path planning. https://gitlab.com/itandroids/open-projects/ssl-sampling-based-path-planning, 2020.
[7] Jonathan D Gammell, Timothy D Barfoot, and Siddhartha S Srinivasa. Batch informed trees (bit*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research*, 39(5):543–567, Jan 2020.
[8] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Bit*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. *CoRR*, abs/1405.5848, 2014.
[9] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed rrt*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic. *CoRR*, abs/1404.2334, 2014.
[10] Lucas Janson and Marco Pavone. Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions - extended version. *CoRR*, abs/1306.3532, 2013.
[11] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *CoRR*, abs/1105.1186, 2011.
[12] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.
[13] Mathew D. Penrose. Connectivity of soft random geometric graphs. *Ann. Appl. Probab.*, 26(2):986–1028, 04 2016.
[14] Robocup. Rules of the robocup small size league. Technical report, Robocup, 2020.
[15] I. A. Sucan, M. Moll, and L. E. Kavraki. The open motion planning library. *IEEE Robotics Automation Magazine*, 19(4):72–82, 2012.
[16] A. Wendler and T. Heineken. Extended team description paper - er force. Technical report, FAU, 2020.
[17] K. Zhu, C. Hooper, J. Keszler, and K. Gonzalez. Team description paper - rfc cambridge. Technical report, Harvard University, 2020.