Real-Time Single-Agent Motion Planning in a Highly Dynamic Environment

Jackson Kruger
kruge203@umn.edu

May 14, 2019

Abstract

This paper presents a comparison of two approaches to the problem of single-agent real-time motion planning in a highly dynamic environment with a large number of obstacles. The first approach builds upon the notion of a probabilistic roadmap (PRM), while the second utilizes the concept of a rapidly-exploring random tree (RRT). Each approach is implemented inside a game and compared on the basis of computational runtime, as well as a number of metrics evaluating the effectiveness of the plans generated. Experimental results found that the PRM-based approach was faster and more effective than the RRT-based approach, in all metrics. A discussion of the possible reasons for this difference entails.

1 Introduction

Within the fields of robotics and Game AI, planning an agent's path through an environment is a common task. The agent, with some amount of knowledge about its environment, must decide how it's going to move through that environment, typically with the purpose of reaching a goal of a specific location within the environment. In these situations it is critical that the agent choose its path such that it does not risk collision with anything within the environment. If, for example, a robotic car is driving on the road, it is critical that the car avoids all obstacles present in the environment.

Although self-driving cars are only beginning to move beyond the realm of academia, robots are already in widespread use within the world of manufacturing. Much in the same way that cars must, an AI designed to play a real-time game must be able to navigate intelligently through the environment of that game. This make the form of nonplayer characters walking through the player's environment, or perhaps that of an agent taking the role of a player in a game where the core of the game is obstacle avoidance. In both of these examples, the agent faces a variety environmental hazards, commonly referred to as obstacles, that may be either static or dynamic with respect to time. Understanding whether the obstacles will be only static or whether that may also be dynamic is crucial to successful motion planning.

For the example of a self-driving car, suppose the car assumed all obstacles in the environment it's driving through were static. At an intersection, it may observe a crossing car that's not yet directly in front of it, plan to drive through the perceived open space, and begin to execute that plan. The the car does not recompute its plan soon enough, it very well may crash into the car crossing the intersection, as it was a dynamic obstacle, not a static one.

If the obstacles in an environment are all static with respect to time, this environment can be planned over in its most natural physical representation - in either two dimensions or three dimensions. So for a two-dimensional environment with static obstacles, the agent must only plan a series of two-dimensional locations within the environment. However, if the obstacles are known to be dynamic, changing environmental position over time, time itself must be added as a dimension for the path planning. With the increase in dimensionality of the problem, the complexity (and thus the necessary computation) of the problem also rises. This leads to the concept of a *space-time* environment for planning, where dynamic obstacle velocities are captured by the spacetime representations of those obstacles. For example, a circular obstacle moving at a constant velocity through a two-dimensional environment becomes a three-dimensional slanted cylinder.

Once the environment is characterized, the actual planning must be performed. There are a variety of techniques for doing this - this paper examines two in particular: a probabilistic roadmap (PRM)-based weighted A* search, and a rapidly exploring random tree (RRT)-based approach. After a brief description of the exact problem at hand, the body of work related to this problem will be discussed in Section 2. The approach implemented for this paper will be detailed in Section 3. Experiments conducted, their results, and analysis of those results will be presented in Sections 4 and 5. Final conclusions will be drawn and future work will be discussed in Section 6.

1.1 Problem Description

This paper will deal with the specific problem of single-agent motion planning and obstacle avoidance within the highly dynamic environment of a video game. The environment the agent must move through will be two-dimensional, and will only have dynamic obstacles. The dynamic obstacles come in the forms of various enemies that spawn near the top of the environment, then proceed to launch patterns of circular bullets around the environment, while moving towards the bottom or sides of the screen. All dynamic obstacles in this environment move at a constant velocity once they appear. If the agent touches any of the enemies or their bullets, it loses the game. Its goal is to survive for as long possible without getting hit by any obstacles. While avoiding obstacles, it will also have a sequence of randomly generated goal locations to reach. The pattern the enemies appear in is always the same for consistency, however, for the agent the environment and its obstacles are considered unknown until they appear on screen.

2 Related Work

A very wide variety of approaches to motion planning in dynamic environments have been proposed, most within the context of robotics. Each approach can be evaluated on the basis of several factors, the most relevant of which are the speed of the algorithm, the approach's guarantees against collisions (which is closely related to completeness and speed), and the optimality of the solutions generated. For the problem at hand, the first priority of our agent is simply avoiding the dynamic obstacles in the environment; the goals for the agent are secondary to survival. Thus, speed and safety are the primary concerns of the for this agent, while optimality in reaching the goal is secondary.

Motion planning algorithms can broadly be broken down into two categories: global planning techniques, and local planning techniques. The global algorithms will attempt to find the entire path from the start to the goal before the agent takes any action. The local techniques broadly focus on moving towards the goal while avoiding local obstacles, often not planning a full path.

One of the most common global planning approaches is using the concept of a probablistic roadmap (PRM). Originally introduced in [5], a PRM is a graph generated randomly sampling the configuration space in which the planning will be done. Through this random sampling, the potentially highly dimensional continuous space is discretized, making it possible to run a graph search algorithm such as A* to determine a valid path. However, the original conception of a PRM alone did not account for a dynamic environment.

The concept of a PRM has since been adapted for use in a real-time dynamic environment, as in [10]. This approach abstracts a 2D configuration space into 3D by adding a time dimension, allowing obstacle movement to be accounted for while performing the motion planning. It also does so using anytime D* to search over the PRM, allowing a subobtimal solution to be returned quickly, and refining the solution as time allows.

An alternate method of discretizing a configuration space has been proposed in the form of rapidly exploring random trees (RRTs) [7]. With this approach, a tree is incrementally built over the configuration space by randomly sampling the space and connecting to the existing tree. Once the tree reaches the goal, the tree can be backtracked to find the path from the start to the goal. This original approach made paid no heed to the optimality of the solutions it generated - it simply focused on finding *a* solution. Additionally, the original conception of RRTs had to be completely recalculated if any changes in the environment occur, making it a poor fit for a dynamic environment. Both of these issues are addressed in [2], which proposes an RRT algorithm that is both dynamic, handling environmental changes without full recomputation, and anytime, first providing a highly suboptimal solution quickly, then refining the solution's optimality given more time.

A third approach to motion planning doesn't use the concept of a configuration space at all. Instead, [3] introduces the notion of velocity obstacles. A velocity obstacle is the set of velocities that, when taken by the agent, will lead to a collision with an obstacle, static or dynamic. Thus, once the velocity obstacles are calculated for the obstacles in the dynamic environment, any velocity outside these obstacles can be taken to guarantee collision avoidance. Sets of these 'avoidance maneuvers' can be chained together in a search tree to plan a full path from a start to a goal. This approach focuses on obstacle avoidance first, with path planning as a step that can be added to the algorithm. An extension of the notion of velocity obstacles, commonly referred to as ORCA, provides guarantees on collision avoidance between very large number of agents [11]. It optimizes the VO approach in several ways, and combines it with linear programming techniques to do this. However, it makes the fundamental assumption that any obstacles that are moving are also agents following the same algorithm. It is thus no suited to the task at hand.

Another global planning approach, with a focus on optimality, builds an algorithm around the notion of safe time intervals for the agent [9]. A safe interval is a combination of a state configuration and a time period without any collisions. This definition compressed potentially large periods of time into a single state, with the goal of resulting in a smaller search tree than a more typical space-time discretization of the configuration space. It also allows agents to remain in place as a means of obstacle avoidance, where other approaches may detour around space-time obstacles or even fail to find a solution. However, even though this approach outperforms other optimal motion planning approaches, it cannot scale regarding speed as subobtimal solutions.

A variety of local obstacle-avoidance techniques have also been proposed, using notions like artificial potential fields [6], virtual force fields [1], and kinematically-based methods [4]. However, although these methods all focus on producing solutions faster than is typically possible for the more complex problem of global planning, they make a sacrifice in optimality and even completeness. Local methods can be prone to falling into local minima and thus ultimately colliding with obstacles, and often pay little or no heed to the optimality of the solutions they produce. One method uses the notion of inevitable collision states to provide guarantees against collisions while doing partial motion planning [8].

3 Approach

Two separate approaches were implemented to solve this problem: a PRM-based approach and an RRT-based approach. Each approach was allowed an initialization phase, after which the game was treated as continuously running, independent of the agent's computation. In other words, the environment was treated as being dynamic. Both approaches were implemented as additions to a game the author had previously written using the game engine Unity[1][2]. This allowed the agent to have full, direct knowledge of its environment, without any intermediate perception on the agent's part.

Both approaches make use of a spacetime representation of the environment, with only circular obstacles. When an obstacle appears on screen, its initial position, velocity, and its radius are recorded. The radius of the obstacle is inflated by the radius of the agent to allow the agent to be treated as a point during planning. Each obstacle is then represented as a three-dimensional slanted cylinder, on which tests are performed to determine whether transitions between states are safe or not. A state is defined as a combination of a 2D position and a point in time.

3.1 Probabilistic Roadmap

This approach closely builds off of the work presented in [10]. It begins by constructing the PRM for the 2D environment. A large number of random valid positions for the agent are generated within the environment. Each of these positions, referred to as a node, is then connected with some number of its nearest neighbors, using a KD-Tree for efficiency. At the end of this process, the PRM represents a classical undirected graph of valid two-dimensional positions for the agent, and valid transitions between those positions.

Note that this PRM is only two-dimensional, while the overall environment being planned over is three-dimensional. This is possible because the third dimension, time, is related to the space dimensions via velocity. This allows a transition model to be defined, as in [10]. Put briefly, each state is allowed to transition to any at the same node but a greater time, remaining in place, or to any node that is connected to the current node, with a time value greater than the current one by at least the time needed to move between the nodes. Every such transition is tested against all spacetime obstacles. If a intersection occurs, the transition is not allowed.

With the PRM constructed, and a state transition model defined, a modified A* search can be performed. The agent's current position and time are used as the beginning state for the search. A randomly chosen goal position is generated, and a node, n_{goal}, is temporarily inserted into the PRM. The goal state for the search s_{goal} is defined as the state $(n_{goal}, t_{goal} * k)$, where t_{goal} is the minimum time it would take to reach the goal via a straight line, a $k > 1$ is a constant.

[1]The game is called "Jimstown". A brief demonstration video can be found here: https://youtu.be/BHfTiUOVMjo
[2]More information about Unity can be found here: https://unity.com/

The A* search is then run as usual, with several modifications. The search's termination condition is not reaching the goal state s_{goal}, but instead simply the goal's node. The search also monitors its running time and may terminate early if its runtime exceeds a preset value. Additionally, the search is performed as weighted A*, so the f-value of state s is defined as $f(s) = g(s) + \epsilon * h(s)$, where $\epsilon > 1$. This trades off search speed for its optimality - a valuable trade-off in this environment. Also worth noting is that the g-cost and the h-cost both only consider the 2D positions of the states, not their time values. This encourages the search to reach the goal in less time, as the f-cost is not lowered by remaining in the same location.

3.2 Randomly-Exploring Random Tree

The structure of this approach is very similar to the structure of the basic RRT algorithm presented in [2]. Unlike the PRM-based approach, an RRT requires practically no work to initialize - a state is created for the agent's current position and time, and is added to the RRT. A goal node is randomly generated, and a window of time for reaching that goal $[t_{gmin}, t_{gmax}]$ is computed.

The RRT is then grown step by step. Each step, a target state is randomly chosen. With a chosen probability $p_{goal} < 1$ a goal state is generated at the goal node with a random time between in the window $[t_{gmin}, t_{gmax}]$. Otherwise, a state is generated with a random location and a time between the current time and t_{gmin}. This random target state is then forced to be of a maximum distance distance from its nearest neighbor, with an adequate amount of time to transition from its neighbor, and to be of at least a minimum velocity. The target node is then connected to its nearest neighbor in the RRT, but only if the connection does not intersect any spacetime obstacles. The RRT algorithm terminates once its runtime exceeds a preset value or it has reached the goal position.

3.3 Continuous Path Planning

For both approaches, the agent follows the generated path as closely as possible. If the agent reaches its goal, next time it is asked for its desired velocity, the agent will re-run the path planning algorithm with a new random goal. If at any point a new spacetime obstacle appears that intersects the agent's current path, the path is regenerated, aimed at the same goal.

4 Experiments and Results

The experiment conducted aimed to compare the two methods presented above, and evaluate them on relevant metrics. As such, the experiment was designed to control as many variables as possible between the two methods. Both path planning methods were run on a standard level of the game under question that was designed before work on this paper began[3]. A few modifications were made to this level to match the focus of this paper. Most notably, enemies that previously 'chased' the player in such a way as to try and crash into the player were forced to have a constant velocity. The number of bullets fired by the enemies was also effectively increased to present a more thorough challenge to the agents with a wider range of total number of obstacles on screen at any given time.

Due to the randomized nature of both approaches, 10 trials were run for each approach. The runtime cap was set to 19 milliseconds to avoid impacting the game's overall performance. The

[3]Here is a link to a demonstration video: https://youtu.be/PegbrTx9gwc

PRM approach used a PRM with 1000 nodes, each with 30 connections, with an ϵ value of 10. The RRT was run with a p_{goal} value of 0.15, a maximum tree extension length of 0.5, and a maximum agent speed slowdown of 1.5x. The agent's maximum speed was set to a value of 5, the same value used for human players of the game.

For each trial, four values were recorded each time the path planning algorithm ran: the time the run occurred at, the duration (runtime) of the algorithm in milliseconds, the number of obstacles present for that run, and the length of the path (in number of nodes) planned by the algorithm. The averages across the five runs per approach for the recorded values are shown for both the RRT and the PRM in Figure 1.

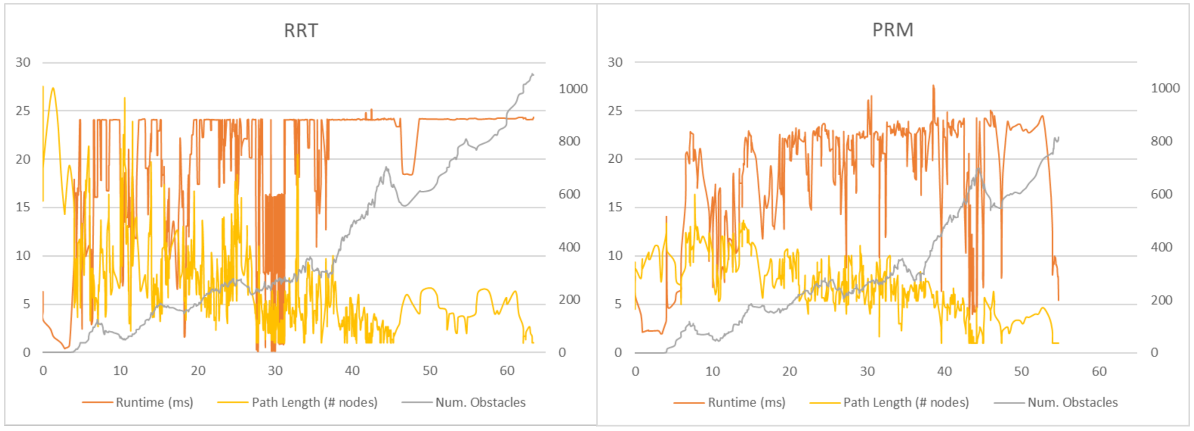

Figure 1: Comparison of algorithm runtime, path length, and number of obstacles over time, averaged over 10 trials

The total time survived by the agent was captured as an overall measure of success. The number of times the agent reached the end of its path before more was generated was recorded as a measure of how well the algorithm stayed ahead of execution. These instances will be referred to as 'No Goal' occurrences, and will be measured as an average per second. As a final measure, the average number of seconds it took for the agent to reach the randomly chosen goals was recorded as a measure of path planning competence, instead of mere obstacle avoidance. These values are shown comparing the PRM approach and the RRT approach in Table 1.

	Time Survived (s)	No Goals/s	Average Seconds/Goal
PRM	45.0	0.766	4.35
RRT	42.5	1.866	5.29

Table 1: High-level results over 10 trials comparing the PRM and RRT methods

5 Analysis

Figure 1 shows several interesting trends. The first is a general increase in algorithm runtime as the number of obstacles increases, alongside a general decrease in path length. This indicates that the primary computational bottleneck, for both algorithms, is checking for obstacles collisions. This finding was separately confirmed by the author using profiling techniques.

It is also visible from Figure 1 that while the RRT continued to use its full runtime until termination, the PRM often did not. This is really due to the fundamental nature of the PRM; the agent's choices are limited to those available on the graph. Thus, when the agent becomes fully surrounded by obstacles, it only checks its current node's connections for valid movements. If none of these movements are valid, the search terminates early, not using its full allotment of computation time. In contrast, the RRT has no limit to the number of transitions it can generate, because each attempted addition to the RRT is a new random sampling of the search space. It thus continues to attempt to find ways out of predicaments until the very end.

It is evident from Table 1 that the PRM outperformed the RRT across the board. On average, the PRM approach survived a little bit longer, had less than half the frequency of no goal occurrences, and was more expedient at reaching its goals. What this really means is that the PRM ultimately spent its time more efficiently than the RRT, doing computations most likely to aid in the continued survival of the agent. This is once again due to the core purpose of a PRM: limiting options. When the agent is fully surrounded, this may not be beneficial, but on the whole, it evidently serves to focus the computations the agent performs in a way that ultimately proves useful. Where an RRT randomly chooses points in the environment, hoping they prove helpful, the PRM already has a set of neighbors to choose from. Moreover, the PRM-based approach is more goal-oriented than the RRT-based approach because of its use of A*. It is always explicitly searching for the goal. The RRT, in contrast, only attempts to move towards the goal with a certain frequency. When an RRT is allowed to run its full course, this tends to work out. However, when the RRT is frequently cut off because it does not have enough computation time, it's possible that it doesn't even attempt to reach the goal for random periods of time.

6 Conclusions and Future Work

We have presented a comparison between two common approaches to motion planning when applied to a spacetime context for a highly dynamic environment. The first approach is based upon the concept of a probabilistic roadmap, generated once during initialization, which is then used as a graph for an algorithm like A* to search over. The second approach utilizes the concept of a rapidly exploring random tree, which has no need for a search algorithm; the very generation of the RRT generates the path. It was found inside the context of a specific game, that the PRM survived longer than, less frequently left the agent without an immediate task than, and more quickly reached randomly generated goals than RRT. This difference likely stems from the fundamental natures of the two algorithms, specifically, the deliberate limitations that serve to focus the PRM-based approach.

Future work could include comparisons to other approaches to motion planning in dynamic environments, such as one based on the concept of velocity obstacles, or perhaps one based on potential fields. More extensive research could also be done into finding the optimal parameters for each individual algorithm, as these may significantly impact those algorithm's performance. Additionally, versions of the PRM approach and the RRT approach exist that aim to reuse previous computation so as to avoid needing to re-plan from scratch if the planned path becomes nonviable. These could perhaps outperform the simpler versions of those algorithms implemented for this paper.

References

[1] J. Borenstein and Y. Koren. High-speed obstacle avoidance for mobile robots. In *Proceedings IEEE International Symposium on Intelligent Control 1988*, pages 382–384. IEEE, 1988.

[2] D. Ferguson and A. Stentz. Anytime, dynamic planning in high-dimensional search spaces. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1310–1315. IEEE, 2007.

[3] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[4] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[5] L. Kavraki, P. Svestka, and M. H. Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, volume 1994. Unknown Publisher, 1994.

[6] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[7] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[8] S. Petti and T. Fraichard. Safe motion planning in dynamic environments. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2210–2215. IEEE, 2005.

[9] M. Phillips and M. Likhachev. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*, pages 5628–5635. IEEE, 2011.

[10] J. Van Den Berg, D. Ferguson, and J. Kuffner. Anytime path planning and replanning in dynamic environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2366–2371. IEEE, 2006.

[11] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.