Simulating Pathfinding in Unity3D Using RRTs

Anthony Sermania
George Mason University
asermani@gmu.edu

Abstract

In this project, I create a simple simulator for navigational agents utilizing Rapidly Exploring Random Trees to explore and build paths within an environment in order to better understand the concept of RRTs and their implementation and usage in robotics. By doing so using Unity3D for easy visualization and motion implementation, the problem was reduced to implementing the pathfinding algorithm and witnessing the results. The agents performed reasonably well, even in complex or large environments and clearly demonstrated the fundamental capabilities of this approach.

1. Introduction

The challenge investigated in this work is that of reasonable pathfinding using the Rapidly Exploring Random Trees (RRTs) method presented by LaValle and Kuffner [1]. The purpose of this effort is to further my personal understanding of the concept and verify my ability to emulate the broad application of this algorithm in a fairly practical setting.

There is little by way of innovation in the effort made here, and any open issues in the technology as a whole are better discussed in [1].

The original intent had been to compare A* pathfinding in a grid to some other (unspecified) grid-based pathfinding algorithm. This however was discarded in light of affirming my knowledge about a topic more relevant to the course directly that felt largely unlearned. It was also more direct to implement RRTs in the Unity3D engine by virtue of spaces already being represented continuously (rather than as a grid; more unnecessary components would need to be built to serve as the foundation for running the algorithms before even implementing them).

There were limited challenges in developing the simulator, and on the whole, it went quite smoothly after fixing a few minor logical errors in one of the scripts developed. The resulting simulator is a simple, multi-situation engine for easily visualizing the results of using an RRT to develop a planned path with which to navigate an environment. Improvement options remain to allow the simulator to become more robust. It could be expanded to allow development of environments and situations for more customized experimentation, for example.

2. Approach

1.1 Core algorithm

The approach taken was quite straightforward: implement a RRT-like algorithm to generate pathing nodes to assign as sequential destinations for an agent navigating the space, starting at the agent's initial location and ending at some goal. The actual script that builds the tree is fairly short, as the algorithm given by LaValle and Kuffner [1] is impressively simple.

Given a starting position and a goal position, the agent builds a path between the two. A list of nodes is maintained for ease of reference, and a stack (initially empty) is created to build the actual path later. The first node made is for our start position. Hereafter, X and Z values (Y is the "up" axis in Unity) are randomly generated to create random points in an iterative process.

At each iteration, the new random point is checked against all existing nodes to find the nearest node to which a clear line of sight (i.e. no obstacles are detected) can be established. If such an existing node can be found, a new node is made for the random point (which is then added to the total list of nodes) and the discovered nearest node is made the parent of the new node. This establishes a one-way connection up the tree which is taken advantage of in the final pathing step.

At each iteration, before generating a random point, every node in the list is checked to see if there is a clear line of sight to the goal position. If there is, we create a new node at the goal position and assign the successful node as the parent of the goal node. This is very much a brute force approach, and a small performance gain could be made by not checking nodes that have already been discovered to not have line of sight to the goal.

Once the goal node has been created, the agent can build a path by tracing the lineage of the nodes backwards via the parents of each node starting from the goal. This is done with a simple loop that iterates over what is essentially a linked list of nodes starting at the goal and going back by the parent link until we reach the starting node, for which no parent was defined. Each node reached in this way is pushed onto the path stack created at the start. Once the path is made, the agent is permitted to begin moving, and each time a node position is reached, the next node is popped off the stack.

It is worth noting at this time that we take a measurement from each node position as if the agent was there to establish whether or not there is a line of sight. This assumes we have pre-defined knowledge of the environment. We could instead only operate as far out as the agent can currently perceive, and continue to build paths with that limitation, and moving along generated paths while still building in real-time. This however requires some other mechanism be put in place to help guide us to our goal, and likely relies on heuristic traversals and potential backtracking. While this is certainly an interesting concept to explore, I decided to make the assumption the agent has knowledge about the environment to simplify the problem for the sake of time and sanity during these weeks of finals.

1.2 Additional components

Further development was required to allow the user to define the start and end positions for the agent within the simulator. To this end a simple user interface (UI) was implemented that utilizes sliders to control the position of both the agent and a visualized target on both available axes. There are slider sets for each agent and its goal in the current simulation.

Several environments were constructed to provide different options for nontrivial navigational trials. The more complex the environment, the longer the building phase takes.

Generated nodes are visualized by small objects displayed to the screen, and the selected path is highlighted once established to demonstrate the functioning algorithm.

2. Results

The resulting simulator functions pretty much on point with what I hoped to accomplish. I had struggled before to really grasp the implementation of RRTs despite their simplicity. I suspect this difficulty came from the sudden jump from theory to application without much segue, and the additional complications of juggling values in MATLAB for controlling robot motion on top of that. Operating in Unity3D, a more familiar environment that let me abstract and separate various control structures and values more easily helped solidify my basic understanding on the core concept greatly.

Overall, it's interesting to watch the paths the agents take during each run of a simulation. Due to the randomness involved its possible for incredibly suboptimal paths to be taken, though this occurs rarely. More often than not the lack of optimality comes from the generation of so many unnecessary nodes, rather than the completion of a suboptimal path. That is, in a sense, part of the point of this method however: to uniformly explore the available space to find a solution.

Additionally, it is worth noting, albeit it trivially, the vast increase in number of nodes created as environments get larger.

3. Related Work

While I am certain a fair amount of work has been done regarding RRTs (indeed, LaValle has a great many papers on the subject), there was very little that bore relevancy to my highly-simplified objective. I wish I had felt more comfortable with this subject to push the boundaries of it and explore various perspectives on a topic like this to cobble them together and attempt to cross (even in failing) some profound technical gap, but it was not to be. In the effort to simplify the problem, exploration of related work went undone. I am not convinced I am any worse off for it. My only goal was to understand the rapidly exploring random trees approach and process how it was even remotely efficient or effective at generating paths.

4. Summary

To briefly conclude, I built a pathfinding simulator based around LaValle and Kuffner's Rapidly Exploring Random Trees algorithm [1]. This simulator featured several environments in which one or several agents could be positioned and given goals and left to find their way using the RRT algorithm.

While there is little to say about what was done, what remains to be done in future iterations of this simulator is far more interesting. Adding a toolset for building an environment and specifying the number of agents is a strong contender for the next thing to implement. Tackling other continuous pathfinding techniques, or removing the assumption of knowledge about the environment as options for the simulator are also good steps towards expansion. This latter especially would open up a world of research to delve into and discover.

References

[1] S. M. LaValle and J. J Kuffner. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293-308. A K Peters, Wellesley, MA, 2001.