Brian Moriarty
HW3
CS389
20-April-2012

TREE BASED MOTION PLANNING

I.Motivation

Tree based frameworks for motion planning can be an effective means of discovering unobstructed pathways from an initial state position to a goal position. In class we discussed a number of algorithms designed to build a tree of valid states by randomly sampling a workspace and then attempting to connect the random state to a vertex in the tree if the state can be identified as reachable through a series of intermediate valid states.

Homework 3 identified three specific variants of tree based planning algorithms to be implemented and tested in simulation. This report provides a high level overview of the three specific algorithms: ExtendRandom, ExtendEST and ExtendRRT. In addition, Homework 3 asked for an additional algorithm called ExtendMyApproach to be designed and implemented for comparison and competition with with the three more common tree based planner variants.

Section II of this report provides a basic description of the three specific algorithms. Section III details of the design of ExtendMyApproach and the types of scene conditions it excels at solving, as well as advantages and disadvantages in relation to the other algorithms. Experimental results are provided in Section IV with additional comparison points. Section V recalls some of the challenges common to any tree based sampling algorithm and offers suggested methods of improving the experimental implementations.

II.Background

Homework 3 specified each algorithm should be implemented following this basic framework:

1. Sample State: choose random state from the goal area with probability ($p = 0.1$), and from the entire bounding box with probability ($1 - p$).

2. Algorithm specific logic about how best to choose the vertex to which the sample state should be added.

3. Extend Tree: take small steps from the chosen vertex to the sample state, adding each valid intermediate step until the state can be added, or stopping earlier if an invalid state or goal is encountered.

Steps #1 and #3 are nearly identical for all of the algorithm implementations. Only ExtendEST varies slightly in order to maintain the total weight summation more efficiently. While it is possible to improve the implementations of #1 and #3 in additional creative ways, Step #2 is the focus area of comparison in HW3. The particular process used to decide how to grow a tree from random samples is what defines the strengths or weaknesses of the planner.

ExtendRandom chooses a vertex uniformly at random from all of the tree vertices. This means that a newly generated sample state could be connected anywhere in the tree with equal probability. This could be considered the baseline algorithm for all probabilistic tree based planners. Connecting the random state to a random vertex in the tree succeeds in making meaningful connections more often when there are few obstacles. Once a number of obstacles have been added and clear lines of visibility between various sections of the workspace are obscured, ExtendRandom can have trouble expanding out of areas it has already explored since the uniform selection process over all vertices favors areas which have a higher density of vertices to pick from.

ExtendEST is a variant of the Expansive-Space Tree algorithm. In general, EST attempts to overcome the shortcoming of ExtendRandom by assigning weights to the vertices in order to increase the probability that a desirable vertex will be chosen as the connection point for the random state. This ExtendEST algorithm assigns a weight based on the number of children a node has, such that nodes with fewer children are more likely to be chosen. The idea is that nodes with fewer children may be likely to extend further into unexplored areas and are good candidates to help expand the range of the tree into new regions. In experimentation, the weighting scheme used by ExtendEST does not always work well because the computation time required to choose the weighted vertex limits the number of samples per second. In addition, ExtendEST is similar to ExtendRandom in that as the population of nodes in the explored areas grows the likelihood of picking a node in that explored area increases because new nodes with no children continue to be added there if it cannot luck out early and find a clear connection point into a new open area. Alternative and more effective weighting schemes have been implemented for EST algorithms, however the scheme used by ExtendEST is what this report uses for comparison.

ExtendRRT is the Rapidly-exploring Random Tree algorithm. The pseudo code for ExtendRRT is listed as Algorithm I in order to provide a baseline for discussion in Section III. This method uses the euclidean point distance between the randomly selected state and the tree vertices to identify which vertex is nearest to the new state. As we shall see in later sections, this approach can work extremely well because choosing closer points increases the chance that the new state can be added without obstruction, which is an enormous help in creeping the tree into new areas. Section III will provide additional details on ExtendRRT as it provides a baseline for ExtendMyApproach.

Algorithm 1 ExtendRRT

1: $sto \leftarrow$ select random state(x,y)
2: $v_{nearest} \leftarrow v_1$
3: $dist_{min} \leftarrow Max\ Number$
4: **for all** v in vertices[1 .. n] **do**
5: $\quad dist_{test} \leftarrow line\ distance\ from\ v_i\ to\ sto$
6: $\quad$ **if** $dist_{test} \leq dist_{min}$ **then**
7: $\quad\quad dist_{min} \leftarrow dist_{test}$
8: $\quad\quad v_{nearest} \leftarrow v_i$
9: $\quad$ **end if**
10: **end for**
11: ExtendTree(sto, $v_{nearest}$)

III. Algorithm

The ExtendMyApproach algorithm is a modified version of RRT. While ExtendRRT searches for the closest vertex over all vertices, ExtendMyApproach adds a preference to this search using horizontal and vertical bounding boxes, such that the closest vertex within the bounding box will be chosen over closer vertices outside the box. If no vertices are found inside the bounding box, ExtendMyApproach will fall back to choosing the closest over all vertex in the same way ExtendRRT would do.

The pseudo code for ExtendMyApproach is listed as Algorithm 2. A short summary of the pseudo code is:

(1) On each random state selection, it creates a bounding box around sto, stretching either horizontally or vertically to the workspace edges (50% chance of either one for each sto selection).

(2) It then prefers the nearest vertex in the bounding box, even if there are closer vertices outside the box.

(3) If no vertices are found in the box, it falls back to using basic RRT, choosing the closest vertex over all.

The pseudo code listed in Algorithm 2 describes how that idea is implemented. In Line 1 the algorithm makes a random choice between using a vertical bounding box or a horizontal bounding box with equal probability between either choice. The algorithm refers to the bounding box characterization as the "strategy" for choosing a vertex from the tree. In line 2, the random state is selected using the exact same methodology used by ExtendRRT. The for loop at line 7 iterates over all vertices, and lines 8-12 in ExtendMyApproach are identical to the code which makes up the entire loop body in lines 5-9 of ExtendRRT. This block is keeping track of the vertex that is closest to the randomly generated state.

Lines 13-23 is the logic specific to ExtendMyApproach. The algorithm is identifying which bounding strategy was designated in Line 1, and is using the appropriate bounding box for the decided strategy. The algorithm keeps track of the nearest vertex within the chosen bounding strategy. After the loop completes, on Line 25, it checks to see if any vertex was found in the bounding box, and if one was found it uses that vertex to extend the tree. Otherwise, if no vertex was found using the bounding strategy, the closest over all vertex is used on Line 28.

Algorithm 2 ExtendMyApproach

```
1: strategy ← RandomChoice[HORIZONTAL, VERTICAL]
2: sto ← select random state(x,y)
3: v_nearest ← v_1
4: w_nearest ← nil
5: dist_min ← Max Number
6: wdist_min ← Max Number
7: for all v in vertices[1 .. n] do
8:     dist_test ← line distance from v_i to sto
9:     if dist_test ≤ dist_min then
10:        dist_min ← dist_test
11:        v_nearest ← v_i
12:    end if
13:    if strategy is HORIZONTAL then
14:        if dist_test ≤ wdist_min and v_i bounded by horizontal rectangle then
15:            wdist_min ← dist_test
16:            w_nearest ← v_i                ▷ nearest vertex in horizontal box
17:        end if
18:    else[strategy is VERTICAL]
19:        if dist_test ≤ wdist_min and v_i bounded by vertical rectangle then
20:            wdist_min ← dist_test
21:            w_nearest ← v_i                ▷ nearest vertex in vertical box
22:        end if
23:    end if
24: end for
25: if w_nearest ≠ nil then                   ▷ prefer vertex from bounding box
26:    ExtendTree(sto, w_nearest)
27: else                                      ▷ fall back to RRT if no bounded vertex
28:    ExtendTree(sto, v_nearest)
29: end if
```

The advantage of using bounding strategies is that the vertex tree is encouraged to grow in directions defined by the associated bounding boxes. For example, consider the scene in Figure 1 where long horizontal passageways must be traversed in order to reach the goal. The horizontal walls separating the passageways are thin relative to the entire horizontal distance of the scene from left to right. When a valid random state is chosen, ExtendRRT looks for the nearest known vertex, but given the traversal pattern and relative thinness of the walls, the nearest vertex is more likely to be in an unreachable adjacent passageway. The same idea extends to vertical style passageway.

Fig 1: Scene 'hrows' showing long horizontal passageways

ExtendMyApproach benefits from the chance that only vertices along the horizontal section will be considered. When a valid random state is chosen along the center line of a scene wide horizontal passage where tree vertices exist, ExtendMyApproach has at least a 50% chance that one of those vertices will be chosen. This is a significantly improved chance over ExtendRRT whose chance depends on vertices positioned in the random passageway being closer to the random state than vertices in adjacent passageways. Since ExtendRRT's tree must traverse previous passageways in order to reach further passageways, it is very likely that vertices will be found which are almost as close as the thickness of the scene walls, so connecting to a vertex in the desired passageway requires the random state to be fairly close to existing vertices. ExtendMyApproach provides an opportunity to choose vertices along the same passageway as the random state, even though there may be closer vertices according to the euclidean estimation of what is close.

There are similar scenes where these advantages begin to diminish for ExtendMyApproach. Even if there are long horizontal or vertical passageways in a scene, this does not necessarily mean that ExtendMyApproach will outperform ExtendRRT. For example, thicker walls push vertices from adjacent passageways further away, making it more likely that ExtendRRT will find the closest vertex to be in the same passageway as the new random state. In fact, if the walls are thicker than the passageway is long then it is likely that ExtendRRT will even outperform ExtendMyApproach because ExtendMyApproach has almost a 50% chance of choosing a strategy that is not optimal for the given passageway orientation.

ExtendMyApproach also has a disadvantage in spaces where obstacles are spaced somewhat randomly or in other patterns preventing it from taking advantage of long horizontal or vertical traversals. The first disadvantage in this category is simple processing overhead because even if ExtendMyApproach falls back to choosing the same vertex that ExtendRRT would choose, it first has to do more work to confirm there are no bounded points. The second disadvantage is related to the points ExtendMyApproach rules out. There may be a vertex available at a long diagonal for example, but ExtendMyApproach might not notice because of its preference for bounded vertices. In that case, ExtendRRT has an opportunity to make potentially important connections earlier than ExtendMyApproach.

While ExtendRRT and ExtendMyApproach have very different characteristics in the environments discussed so far, the two algorithms share the same advantages and disadvantages in comparison to ExtendRandom and ExtendEST.

ExtendRandom is able to generate and add points to the tree very quickly because there is a minimum of processing done to select the vertex. The amount of work required to choose a vertex does not increase as the number of vertices increase. The work of ExtendMyApproach on the other hand increases linearly in the number of vertices because it must search the entire list in order to identify the nearest vertex to the random point. The minimal work required by ExtendRandom permits the tree to grow rapidly, but does not necessarily lead to progress toward the goal. Over time, the density of vertices in well explored areas increases the likelihood that vertices from those areas will be chosen again and it becomes increasingly difficult to expand to new ares.

ExtendEST is also able to add vertices fairly quickly at first, but slows down in its ability to add vertices to the tree as the number of vertices increases. The reason for the slow down is due to its need to find a vertex which meets the randomly selected weight requirement. This search ends up scaling linearly like ExtendMyApproach, but with lower constant factors. ExtendEST was not observed to offer any advantage over ExtendMyApproach because it chose empirically inferior connection vertices causing its tree to grow much larger before reaching the goal.

IV. Experimental Results

Each algorithm was run 10 times on predefined 2D scenes and data was collected to include whether a solution was found, the processing time to solution and number of vertices in the final tree at the end of each run. The processing timings reflect algorithm processing time, so the actual run may have taken longer in real time depending on graphics settings, but only the algorithm processing time was captured. No algorithm was permitted to spend longer than 120 seconds of processing time. This 120 second time limit was chosen in order to capture a fair success rating for all methods because Extend Random encountered memory constraints after 120 seconds with process sizes approaching 1.5 GB.

The software simulation environment consisted of a point robot with radial buffer, a point goal defined in a circle and obstacles defined as circles. The initial robot position is initially the only vertex in the tree.

METHOD	SCENE 1			SCENE 2			SCENE 3			SCENE 4			SCENE 5		
	AVG	MED	S	AVG	MED	S	AVG	MED	S	AVG	MED	S	AVG	MED	S
RANDOM	0.005152	0.003213	100%	0.017650	0.009904	100%	3.377144	1.737119	100%	-	-	0%	-	-	0%
EST	0.022193	0.013297	100%	0.046122	0.017034	100%	70.4351	88.2111	30%	-	-	0%	-	-	0%
RRT	0.000494	0.000547	100%	0.000737	0.000603	100%	0.010608	0.002932	100%	0.366984	0.336267	100%	2.650627	2.524866	100%
MY	0.001485	0.001176	100%	0.001224	0.000946	100%	0.079512	0.014435	100%	0.221488	0.205036	100%	9.362256	9.175252	100%
BEST AVG	RRT			RRT			RRT			MY			RRT		

Table I: Summary Results of experiment runs

Table I lists the summarized average time, median time and percentage of solutions for each test group combination of algorithm and scene. The average and mean calculations include only timings which succeeded in reaching the goal and exclude tests which were stopped at the 120 second limit without reaching the goal. On most scenes RRT performed significantly better than the competition, with the exception of Scene-4 where ExtendMyApproach performed better.

Scene4 and Scene5 depicted below contain identical obstacles, but different start and finish locations. Red dots indicate the robot and green dots are the goal. The performance difference between ExtendRRT and ExtendMyApproach is large enough that it is due to more than random number generation, but it is difficult to identify the reason for the difference in performance. Observations of the generated tree suggest that the time difference may be due to ExtendMyApproach getting out of the top left corner more quickly using a vertical bounding strategy and then quickly traversing horizontally from the bottom right corner to the goal using a horizontal bounding strategy.

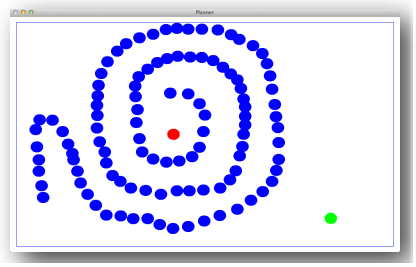

Fig 2: Scene 4

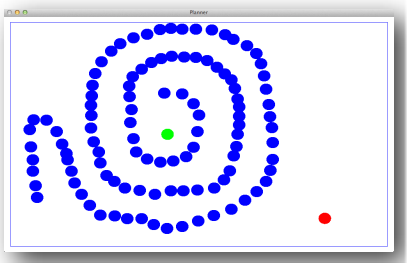

Fig 3: Scene 5

In order to demonstrate a direct comparison of RRT and ExtendMyApproach, a side bar experiment was conducted in order to collect data for 2 additional scenes using only those two algorithms. Figures 3 and 4 respectively show scenes "htiny" and "vtiny" with long horizontal and vertical passageways. The scenes were designed with the explicit intent of exposing RRT's weakness and highlighting the strength of ExtendMyApproach. This is a bit unfair to RRT, but the intent is to provide experimental evidence that ExtendMyApproach can outperform RRT for the scene conditions it was designed to perform well in. The side bar experiment maintained the same 120 second run time limit as the other experiments, and each algorithm was tested 10 times on each scene.

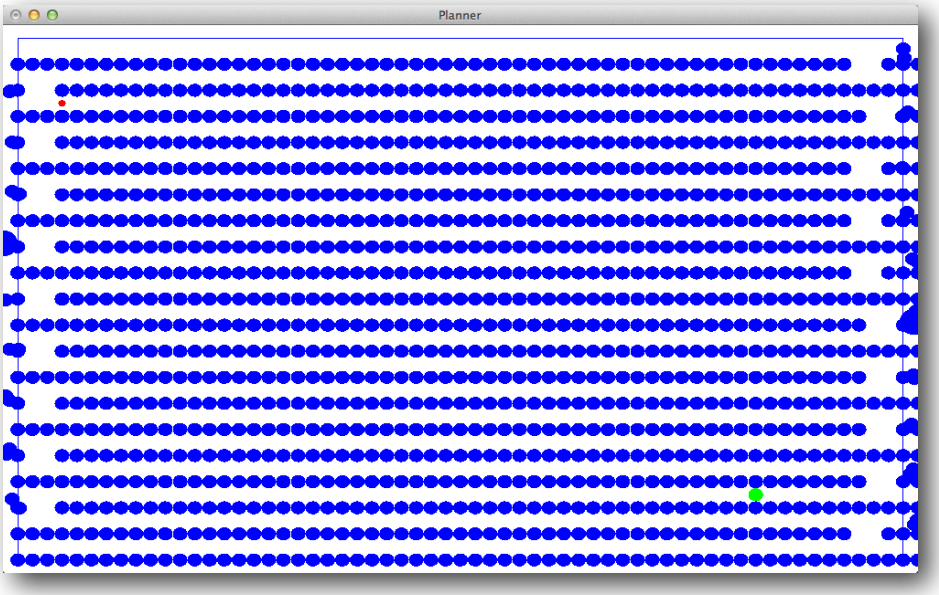

Fig 4 Scene 'htiny' (above) showing long horizontal passageways

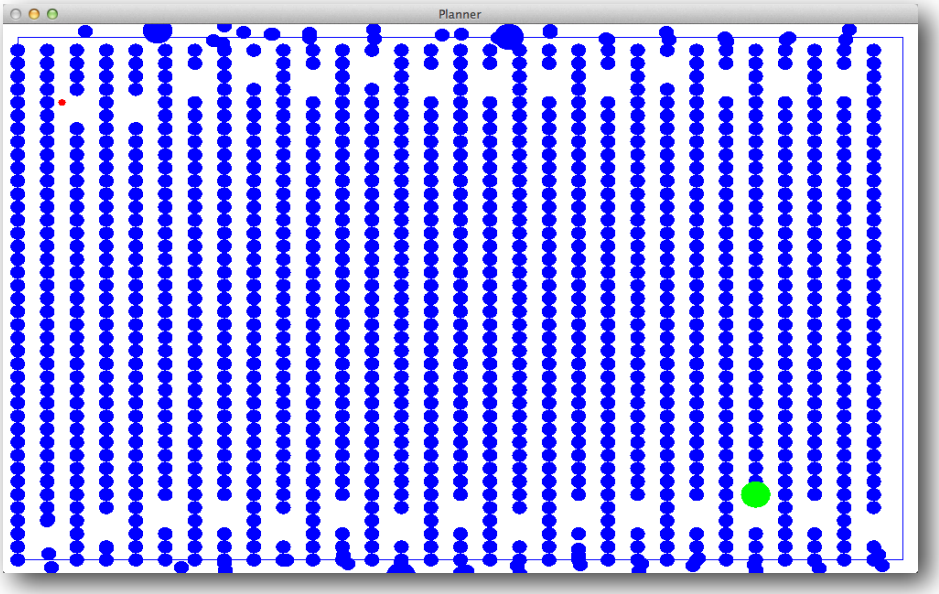

Fig 5: Scene 'vtiny' showing long horizontal passageways

Table 2 contains the results of the side bar experiment. The obstacles and passageways in these experiments are very thin relative to the other scenes, making the scenes ideally suited to the design of ExtendMyApproach. RRT was not able to find a path within the time window provided, further demonstrating RRT's weakness when it comes to traversing both sides of a long thin wall.

METHOD	HTINY			VTINY		
	AVG	MED	S	AVG	MED	S
RRT	-	-	0%	-	-	0%
MY	25.902668	24.727953	100%	16.117138	13.770756	100%
BEST AVG	**MY**			**MY**		

Table 2: Demonstrating MyApproach advantage under specific scene conditions

V. General Challenges

ExtendRandom does not get slower as vertices are added, but all of the other algorithms do. Because the algorithms attempt to search the tree to find a vertex connection point, additional vertices cause later iterations of the algorithm to take more time. Scaling of these implementations might be improved with additional performance tuning, including techniques such as caching or bounding vertex locations with workspace grids, but the root of the problem would be unchanged.

ExtendEST implements a weight caching mechanism to remove the need to sum the entire tree on each iteration. This enables ExtendEST to increase the number of vertices generated per second by nearly a factor of 2. However, after about 20,000 vertices the total weight begins accumulating observable floating point errors when compared to a direct summation. Since both mechanisms contain floating point errors inherent to the environment, the weight caching mechanism was retained.

Tree growth on any of the algorithms can be limited by skipping the addition of vertices which represent robot configurations close to a configuration already in the tree. Informal testing demonstrated that both ExtendRRT and ExtendMyApproach found the path noticeably faster using this approach. More formal experimentation would be needed to say whether ExtendRandom and ExtendEST benefit from this approach.

Another strategy for limiting tree growth is to add only the generated random state and not the intermediate states. While this limits growth, it was observed to have negative effects on ExtendRandom and ExtendEST during informal experimentation who benefited from the intermediate states when seeking paths around obstacles.

The benefit of the bounding strategy used by ExtendMyApproach is somewhat rigidly tied to horizontal and vertical pathways. A similar idea which is not tied to predefined shapes is described by other researchers in Dynamic Domain RRT where Voronoi regions are used to bias the decision toward particular vertices, such as encouraging samples in the space around the frontier of the tree as it grows.

VI. Conclusion

RRT provides a versatile flexible solution for many arbitrary scenes, but its performance can be degraded under scene conditions where nearby vertices are frequently blocked by obstacles. ExtendMyApproach exhibits many of the strengths of basic RRT with a performance cost on general scene layouts but can excel at scenes characterized by long passageways and thin boundaries. EST and Random both have trouble extending their respective vertex trees around obstacle boundaries into new areas.

Random's strength is in its ability to generate many vertices extremely fast, but its ability to continue doing that for very long is constrained because of the memory required to store the tree it generates. EST's attempt to choose vertex connections more intelligently is hindered by the amount of time it takes the algorithm to choose the vertex as its tree grows larger.