Assignment 3 Record Log

By: Team Immortal

Division of Labor:

Robert Pierce: Worked primarily on implementation for ExtendTree and ExtendRandom.

Sara Saad: Also worked on ExtendTree and ExtendRandom for some part; worked primarily on ExtendRRT and ExtendEST

Josh Lilly: Worked on ExtendTree and ExtendRandom

Joe Brock: Worked on the Assignment 3 Record Log, pseudocode for explaining our methodology, and helped with the coding and debugging of ExtendTree and ExtendRandom.

Our Approaches to the Problems provided:

During this project, we had a variety of different methods that we used to implement our Tree structure, our Random function, our RRT, our EST, and our own approach. Here is a brief description of the steps that we took in order to get the most of our functions, as well as the pseudocode it took to create them:

1. Our Tree Extension Function

* A Basic Explanation:

For our Tree Extension function, we started breaking the function down into what we could do. We knew that we had to check whether or not the state that we were going to be moving to had to be acceptable before anything else, so what we ended up doing was that we made two checks. The first check would check to see if we’ve already visited areas that were actually detrimental for our code.

The way that we implemented was that we took our state and then determined how far away our current state from where we were previously. If our radius was greater than our robot’s radius, or if our state is not valid, we would just return nothing so that way it couldn’t execute the second part of the code. If it had to make it past our first check, we would then iterate along the line and check to see if there were any obstacles on the way to our goal. In the case that there were, we would simply add our new vertex to our possible paths.

* Our Pseudo-Code:

> Get the params for vid and our step size

> Set our robot’s state to hypothetically be at where our step size is

> Run a for loop for all of the vertices inside our array of possible vertices

+ If the distance < our Robot’s radius, then we know that our step size is too small and therefore we need to return nothing.

+ If that’s not the case, we continue with our loop and check to see if other vertices within our list are equidistance or are > our robot’s radius

> Get the X and Y value of our random vertex (which is considered our previous node); determine our magnitude from the following formula: sqrt((x^2)+(y^2))

> Check the possible states on the line that we’ve just created to ensure that there are no obstacles within our edge. In the moment that we have one, we must return null (because we can no longer use it!)

> If we’ve reached our goal, we ask our robot that if we have reached our goal

+If we have, we must tell the robot that we have found our optimal path

+If not, we just add the Vertex to our list of possible vertices

1. Our Random Function

* A Basic Explanation:

The way our Random function is supposed to work is that, given a random state generated by our code, we should move our position based on that state. To accomplish that, our first step is to first create a state in which it’s within our radius as of currently. Then, what we do is that we check to see if this is our first vertex or not. If this is our first, then we call extend tree and extend the tree. However, if not, then we create a random vertex for our possibility space based on our max number of vertices available, and then call extend tree with.

* Our Pseudo-Code:

> Create a sample state

> Have our robot then assert that this is now a possible position on our tree

> In the case that our max != 0, create a random vertex index for our Vertex array

+ In the case that max == 0, don’t create a random vertex just yet

>Extend the tree until we reach goal

1. Our RRT

* A Basic Explanation:

For our RRT, we designed is so that way it would be centered more around optimality instead of anything else as opposed to something like EST. The way that we would implement RRT, is that first we would have to generate a random sample and then find the closest index possible. Then, what we do is that we get the smallest possible distance between our random sample and other samples that we input into our code. Then, we extend our code to tree and run until we hit goal.

* Our Pseudo-Code:

> Create Our Sample

> Get the Closest Vertex to our Sample

> Get the distance between our closest Vertex

> Run a for loop for other possible vertices

+ If any other vertex happens to be smaller than our current distance, our smallest distance comes to that and our closest vertex changes

> Then, extend the tree

1. Our EST

* A Basic Explanation:

Since EST is more exploration-based, what we do with this tree is that first we first make sure that we have a random sample that we can choose from. Then, what we do is that we create a value called “weight” and sort through all possible vertices. Within our loop, we’ll continuously add to our weight based on the following equation: (1/(vertex’s children)^2). When we’re done with that, we generate a pseudo-random number between 0 and our weight, and then create another for loop where we do the exact same thing as we did in the previous one, but instead we check when our weight is greater than rand so that we can generate our random index and feed to our ExtendTree(vid,samples).

* Our Pseudo-Code:

> generate random sample, random index and weight (which equals zero, initially)

> run a loop through every vertex

+Here, we continuously add (1/(Vertex’s children num)^2) to our weight

> generate a random number afterwards (known as rand)

> run another for-loop through every vertex

+Do exactly what we did in the first, but this time, if our weight is

greater than our random number, we set our new random index to

our current position within the vertices, and then break

> extend the tree

1. Our Optimized Approach

* A Basic Explanation:
* Advantages/Disadvantages:

One of the biggest advantages to this algorithm is also its biggest disadvantage. Since we’re mostly concerned with how quickly we can get to the end goal with our optimized approach, we are willing to skip possible vertices if we were choosing a more exploration based design. Furthermore, though it is much faster than RRT, certain problems may arise in the instance where we may have multiple goals that we have to go to.

Our Pseudo-Code:

>

>

Experimental Results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Scene1.txt | Scene2.txt | Scene3.txt | Scene4.txt | Scene5.txt |
| Random Runs |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| RRT Runs |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| EST Runs |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Approach Runs |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

\*Note: Params divided up as Average time, Mean time, and the Percentage of Solutions