Truly understanding sampling-based motion planning

This project is intended to make you truly understand and appreciate why sampling-based motion planning algorithms have become *the* dominant paradigm in robotics. You will also see its drawbacks, and that it is not the solution to motion planning problems. You will implement several sampling-based algorithms and answer a series of questions that test your understanding of motion planning algorithms. Your grades will depend on the depth of your understanding. Good luck!

Part 0: Environment creation
Create a motion planning domain of your choice. It can be a 2D or 3D environment. I recommend that you read the below before you set up your environment to plan ahead the kind of environment and robot you want.

1.0 Exploration and exploitation

1.1: An alternative sampling-based algorithm
Implement the following sampling-based algorithm:

1. Initialize a tree with an initial configuration
2. Select a random node from the tree, called x_{tree}
3. Select a random configuration, called x_{rand}, from the collision-free configuration space
4. Extend from x_{tree} to x_{rand}
5. Add the new nodes from the extend operation to the tree
6. If a goal configuration is added to the tree, terminate. Solution found!
7. Otherwise, go to step 2

Visualize its tree-growing procedure, like this video.

Answer the following questions:

1. Prove that this algorithm is probabilistically complete or not
2. Comment on the tree-growth procedure. Does it efficiently explore the configuration space? Why or why not?

1.2: Exploiting domain knowledge

Implement the following alternative algorithm: instead of choosing a random node from the tree in line 2 of the algorithm given in part 1, choose the node with the least heuristic value defined by:

$$x_{tree} = argmin_{x \in V} \text{straight-line-distance-to-goal}(x)$$

where V is the set of nodes in the current tree, and the straight-line-distance-to-goal function measures the straight-line distance to the goal from the given configuration x. Visualize its tree-growing procedure.

Answer the following questions:

1. Is this more efficient than the previous algorithm? Why or why not?
2. How would you improve this algorithm?

1.3: RRT

Now implement RRT. Visualize and compare the tree-growth procedure with the algorithms given in parts 1 and 2.

Answer the following questions:

1. Is RRT more efficient than the ones given in 1.1 and 1.2? If so, describe what makes RRT more efficient.
2. In what cases will the algorithm in 1.2 be more efficient than RRT?

2.0 Limitations of RRT*

2.1 RRT*

Here is a pseudocode for RRT*:

Algorithm 1: $\texttt{RRT}^*((V, E), N)$

```
for i = 1, ..., N do
    x_rand ← Sample;
    X_near ← Near(V, x_rand);
    (x_min, σ_min) ← ChooseParent(X_near, x_rand);
    if CollisionFree(σ) then
        V ← V ∪ {x_rand};
        E ← E ∪ {(x_min, x_rand)};
        (V, E) ← Rewire((V, E), X_near, x_rand);
return G = (V, E);
```

Algorithm 2: $\texttt{ChooseParent}(X_{\text{near}}, x_{\text{rand}})$

```
minCost ← ∞; x_min ← NULL; σ_min ← NULL;
for x_near ∈ X_near do
    σ ← Steer(x_near, x_rand);
    if Cost(x_near) + Cost(σ) < minCost then
        minCost ← Cost(x_near) + Cost(σ);
        x_min ← x_near; σ_min ← σ;
return (x_min, σ_min);
```

Algorithm 3: $\texttt{Rewire}((V, E), X_{\text{near}}, x_{\text{rand}})$

```
for x_near ∈ X_near do
    σ ← Steer(x_rand, x_near);
    if Cost(x_rand) + Cost(σ) < Cost(x_near) then
        if CollisionFree(σ) then
            x_parent ← Parent(x_near);
            E ← E \ {x_parent, x_near};
            E ← E ∪ {x_rand, x_near};
return (V, E);
```

V - nodes in the tree
E - edges in the tree
N - the total number of iterations
Steer - Equivalent to Extend function
Cost(x) - cost from the initial configuration to x
Near(x) - computes a set of configurations near x. It is computed by

$$\texttt{Near}(V, x) := \left\{ x' \in V \ : \ \|x - x'\| \leq \gamma \left(\frac{\log n}{n} \right)^{1/d} \right\},$$

where n is the number of nodes in the tree, d is the dimensionality of the configuration space, and gamma is a user-defined constant

Implement RRT*. Is it faster than RRT? What is the most time-consuming procedure?

2.2 (Potentially) Improving RRT*

Suppose we have the following two functions:

- CostLocal(x_1, x_2) - defined as a straight-line distance between configurations x_1 and x_2
- CostGlobal(x) - defined as a straight-line distance from x to x_{goal}

In the *ChooseParent* function above, instead of choosing the one with the least cost, use the following to determine x_{min}:

$$x_{min} = argmin_{x \in X_{near}} \text{Cost}(x) + \text{CostLocal}(x, x_{rand}) + \text{CostGlobal}(x)$$

In the *Rewire* function, use

$$\text{Cost}(x_{rand}) + \text{CostLocal}(x, x_{rand})$$

instead of using

$$\text{Cost}(x_{rand}) + \text{Cost}(\sigma)$$

in Line 3, and skip the Steer operation.

Answer the following questions.

1. Is this RRT* variant faster than standard RRT*? Why or why not?
2. How would you improve this variant?