COMPSCI 220 Fall 2017
Assignment 8: Rapidly-exploring Random Trees

Out: 14 November, 2017
Due: 21 November, 2017

General Instructions For All Assignments

Assignments in COMPSCI 220 are to be programmed in C++11, using the virtual machine image provided for the class. All assignments must follow the instructions on the handouts exactly, including instructions on input and output formatting, use of language features, libraries, file names, and function declarations. Failure to do so will decrease your grade. All solutions submitted must be formulated by you alone: you must not discuss any part of your submission with other students. Please review the course policies on the course website, `https://people.cs.umass.edu/~joydeepb/220R`.

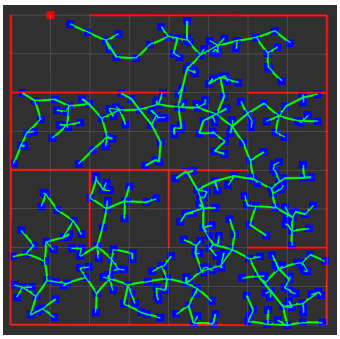

Figure 1: Example of an RRT path planning in a maze.

1 Instructions for This Assignment

In this assignment you will be implementing the rapidly-exploring random tree (RRT) algorithm discussed in lectures 14 and 18. The RRT algorithm seeks to search high dimensional spaces efficiently by randomly constructing a tree that explores the space until it reaches a goal. An RRT grows a tree starting from some initial point in space by exploring randomly generated candidate points in the search space. As each random point is sampled the RRT creates a new node in the direction of the sample from the nearest node already in the tree. This new node is only added to the existing tree if the path to this node is collision free and obeys all constraints.

The goal of this assignment is to write an RRT class that can efficiently find an obstacle free path between any two points in an occupancy grid. For this purpose an occupancy grid will be represented as a grayscale CImg, where each pixel represents a position in the grid. We will treat white pixels as unoccupied, and black pixels as occupied in these images. Your RRT implementation should, given start and end x,y positions (or Nodes), return a collision free path between the two points in the occupancy grid.

2 File Structure and Starter Code

We provide three files for this assignment. **assignment8.h**, **assignment8.cpp**, and **assignment8_testing.cpp**. Signatures for the functions we will be testing should be in **assignment8.h**, your implementations in **assignment8.cpp**, and your tests in **assignment8_testing.cpp**.

We include starter code to simplify testing. This starter code includes a skeleton **RRT** class, and a constructor that takes a single CImg as input representing an occupancy grid. **This constructor is how we will initialize your RRT for our tests. You may modify this constructor, but not its signature.**

Additionally, we include an implementation of a struct **Node** to represent integer x,y coordinates that initially contains only the member variables used for holding x,y coordinates. You may modify the Node struct as necessary, but **do not remove the x,y member variables of Node**, as we will be using these members of Nodes to test the paths your RRT returns.

You are not allowed to use any library or data-structures other than what is provided by the Google Testing library, CImg, `<string>`, `<vector>`, `<stdlib.h>` (for `rand()`), `<queue>`, `<algorithm>`, `<math.h>` and `<iostream>`.

3 Implementation Specification

Your task is to implement the methods of the RRT class listed below. These are the methods we will be testing for a grade, but you may implement other helper methods that you find helpful for the RRT algorithm.

1. A method which, given an input vector of nodes, and an input node, finds the closest Node in node_vector to input_node.

```
Node NearestNeighbor(vector<Node> node_vector,
                     Node input_node);
```

2. A method which given two Nodes, returns true if there is a collision on the path between those nodes in the given map, and false otherwise. Will not be expected to handle input nodes that are outside the map.

```
bool CheckCollision(Node start, Node end);
```

   For the purpose of checking collisions you will need to check specific points in the occupancy grid along the line from the start node to the end node. The algorithm you will use to determine which pixels to check for occupancy is called Bresenham's line algorithm which was discussed in lectures and discussions. For general info on this algorithm you can refer to `https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm` .

3. A method which given two Nodes, a goal radius, and an extension distance for connecting Nodes to the tree, returns a vector of Nodes representing a collision free path from the start Node to a Node within the goal radius of the end Node..

```
vector<Node> FindPath(Node start, Node end,
                      int goal_radius, int extension_distance);
```

4 Example Output

To compile your code with `assignment8_testing.cpp` and create a program called `run_tests`, run:

```
clang++ -std=c++11 assignment8.cpp assignment8_testing.cpp -lpthread -lgtest -lX11
↪  -o run_tests
```

Important: Note that the order of the parameters matters to the compiler because it resolves dependencies from left to right.

5 What To Turn In

Using the provided submission script `submit.sh`, the files `assignment8.h` and `assignment8.cpp` will be checked to see if they compile, and if successful, will be added to an archive, `assignment8.tar.gz`. You must upload this archive to Moodle.