RBE550 - Theft Homework

Nathanael Vickery

Due date: December 6, 2021

The goal of this homework was to use a Rapidly-Exploring Random Tree (RRT) to feel out the obstacle-space of our simulated vending machine. By expanding the tree from inside the main chamber and through two windows, the RRT would build a network of states and finalize a collision-free path from a start state to an end state. Many of the methods used in this homework were adapted from Lavalle's Motion planning [LaV06] and the Theft program was created in Python using Libraries such as Numpy, Scipy, and Matplotlib. For the reference of the reader, an example soda can, Fig. 1, is shown. This soda may also be the greatest soda in all of humanity's history (Thank you Waco, Texas). The following sections include detailed explanations of the development, implementation, and experimentation involved with this homework.

Figure 1: Dr. Pepper: The Soda of Texas

Development

The first part of this homework was developing an obstacle space and environment. Both the floor, ceiling, sidewalls, and barriers were formed from arrays of 3-dimensional coordinates spaced 50mm apart. Although these walls are not shown for clarity, the windows in each barrier are shown in Fig. 2 and in all subsequent plots.

The second part was providing a pool of samples for the RRT to call when needed. Using various random seeds from Python's Random library, an adequate pool of 200 samples was drawn from points within the boundaries of the environment, shown as gold stars in Fig. 2. As recommended by Lavalle's book Sec. 5.5.3, the goal state was added in several times to the sample pool to create a 20% bias towards the goal state [LaV06]. This bias would allow the RRT tree to continually grow while occasionally moving towards the goal state.

The state of the can is kept within a coded node class. The state includes the three positional coordinates (x,y,z) and three orientation coordinates, retained as Euler angles (Z,Y,Z). Each node retains it's parent node and the path between its own state and it's parent state. Additionally, the Collision Detector and RRT Path Planner were implemented and are explained in the next section.

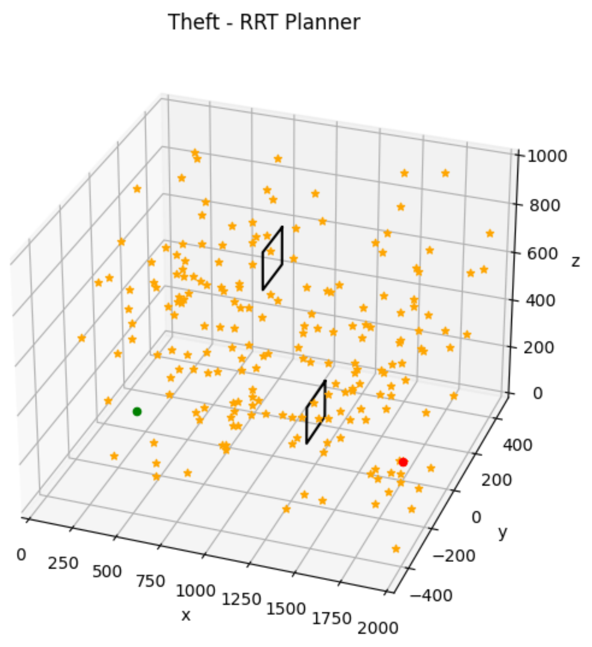


Figure 2: Sample Pool (Includes Goal Configuration at 20% Bias)

Implementation and Experimentation

0.1 Collision Detection

Collision detection was done in two phases as recommended in Sec. 5.3.1 in Lavalle's book [LaV06]. The "broad phase" involved using a Sphere-Bounding Region around the can and would notify the program that the cans state was in collision when this sphere came within range of any of the obstacle space set of coordinates. The "narrow phase" only applied when the can came within a 100mm of the window. In this phase, the bounding region switched to an Axis-Aligned Bounding Box (AABB) as shown in Fig. 3. If at any state the can was found in collision near the window, the path planner would opt to rotate the can and recheck collision. When a suitable collision-free state was found, the state node would be added to the RRT.

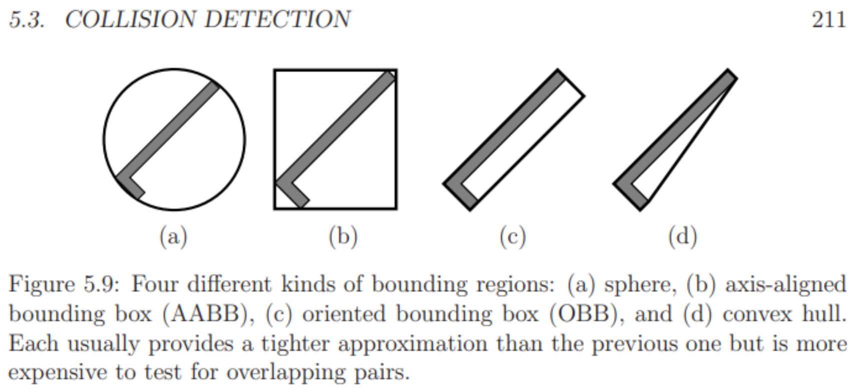

5.3. COLLISION DETECTION 211

Figure 5.9: Four different kinds of bounding regions: (a) sphere, (b) axis-aligned bounding box (AABB), (c) oriented bounding box (OBB), and (d) convex hull. Each usually provides a tighter approximation than the previous one but is more expensive to test for overlapping pairs.

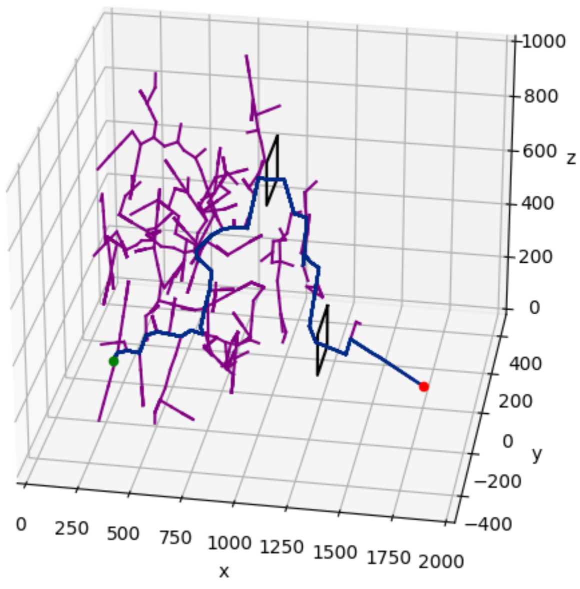
Figure 3: Collision Detection Examples ([LaV06])

0.2 RRT Path Planner

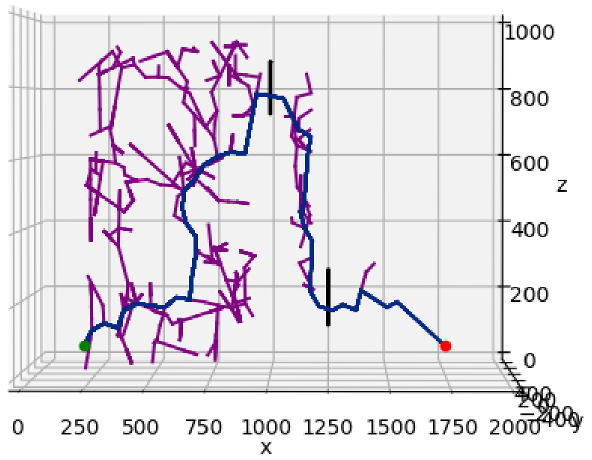
The RRT path planner started at a start state. By randomly pulling sample coordinates from the sample pool, the RRT would select the nearest node in its tree (via Euclidean Norm), step 60mm towards the sample, and, if collision-free, would add the new state to the RRT. As previously explained, each collision-free state was retained as a node in the RRT. Because this RRT continues to grow outward, a uni-directional tree is created starting from the start node and branching either to obstacles or the goal state. The RRT was programmed to stop expanding randomly when one of its nodes came within a radius of 60mm of the goal state. The RRT would then add the goal state to itself and return a the graph and final path. This method was adopted from Lavalle's book Sec. 5.5 [LaV06]. The RRT and final path can be seen in Fig. 4 and Fig. 5.

0.3 Results

Through various testing of sample amounts and RRT iterations, a solution was finally found. At just under 1500 iterations, the RRT expanded to about 540 nodes (if a node was in collision, the RRT continued to the next iteration without adding a new node) and found an adequate solution to the path planning problem. The RRT did feel out the obstacles as can be seen from the side views in the figure but overall, the RRT itself was not very dense. Given more iterations and a decreased goal-bias in the random samples, the RRT may have grown denser, but the solution would have most likely remained the same. The RRT and final path are shown in Fig. 4 and Fig. 5.

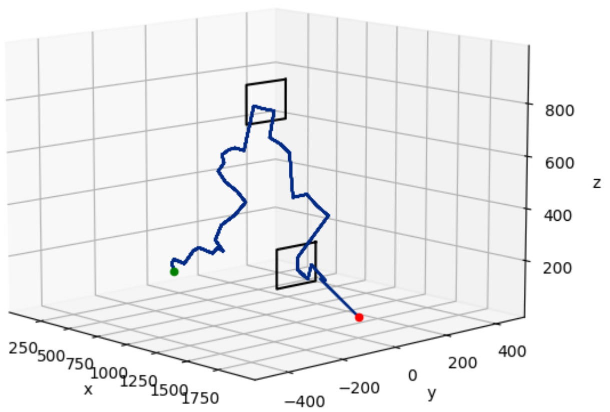


(a) RRT and Final Path (Orthogonal View)

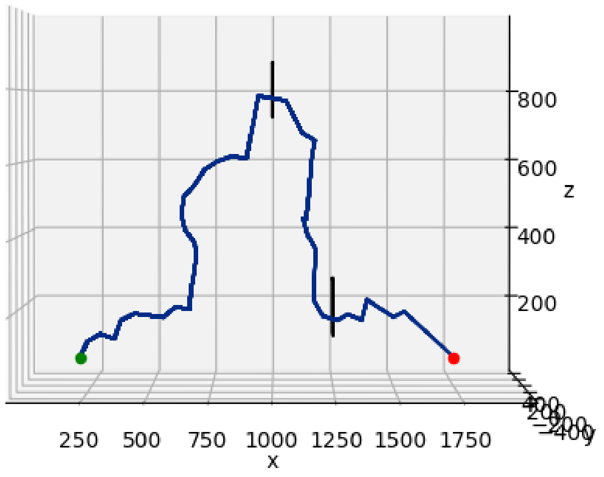


(b) RRT and Final Path (Side View)

Figure 4: Depth First Search Results



(a) Final Path



(b) Final Path: Side View

Figure 5: Depth First Search Results

Appendix

Both codes for the Collision Detection and the RRT Path Planner are written as functions in the accompanying Python code.

References

[LaV06] Steven LaValle. *Planning Algorithms*. Cambridge University Press, 2006. ISBN: 9780511546877. URL: `%7Bhttps://doi-org.ezpv7-web-p-u01.wpi.edu/10.1017/CBO9780511546877%7D`.