COMP 450 Project 2

Taylor Vaughn, Laura Bierwirth, Nick Merritt

Problem Statement

Given a 2-D environment filled with axis-oriented obstacles, we wrote exact collision checking algorithms to check the obstacles' intersections with the three types of robots described below.

Robot Geometry and Configuration Space

The three robot types were represented as a point, and circle, and a square. The point robot was a 1-D robot represented by a single coordinate point. The configuration space for the point robot was identical to the workspace. The circle robot was a 2-D robot represented by a center coordinate point and a radius. The configuration space for the circle robot can be thought of as the workspace extended by the radius and with rounded edges. The final robot was a square robot represented by a center coordinate point, a width, and a rotation θ. The configuration space for the square robot is more complex than the other two due to the introduction of rotation. Therefore, we perform collision checking not in the configuration space but in the work space.

The Algorithm and Testing

The point collision checker was the easiest, with the circle checker being slightly more difficult and the square checker being the most difficult to implement. The point checker was fairly trivial, and the circle checker just built upon it with the addition of radius information. The square checker, however, was more difficult because we couldn't work in the configuration state; instead, we had to check every combination of line segment intersections as well as check if the square was ever inside a rectangle. Additionally, the transformations required to rotate the square required more computation than the circle or point checkers required.

We did not run into any numerical precision issues.

Our implementation accurately classifies the given test sets, including the optional test set.

We made liberal use of cout to determine the precise point where the algorithm failed to assist in debugging. We also drew a configuration space to aid in visualization (see figure 1). One of our main issues was not counting robots entirely inside of an obstacle as invalid configurations. By drawing out the positions, it was trivial to see it issue. Once we reached a relatively high level of accuracy, we pulled out the specific cases that were failing into a test case file. This helped immensely with parsing through our print outs.

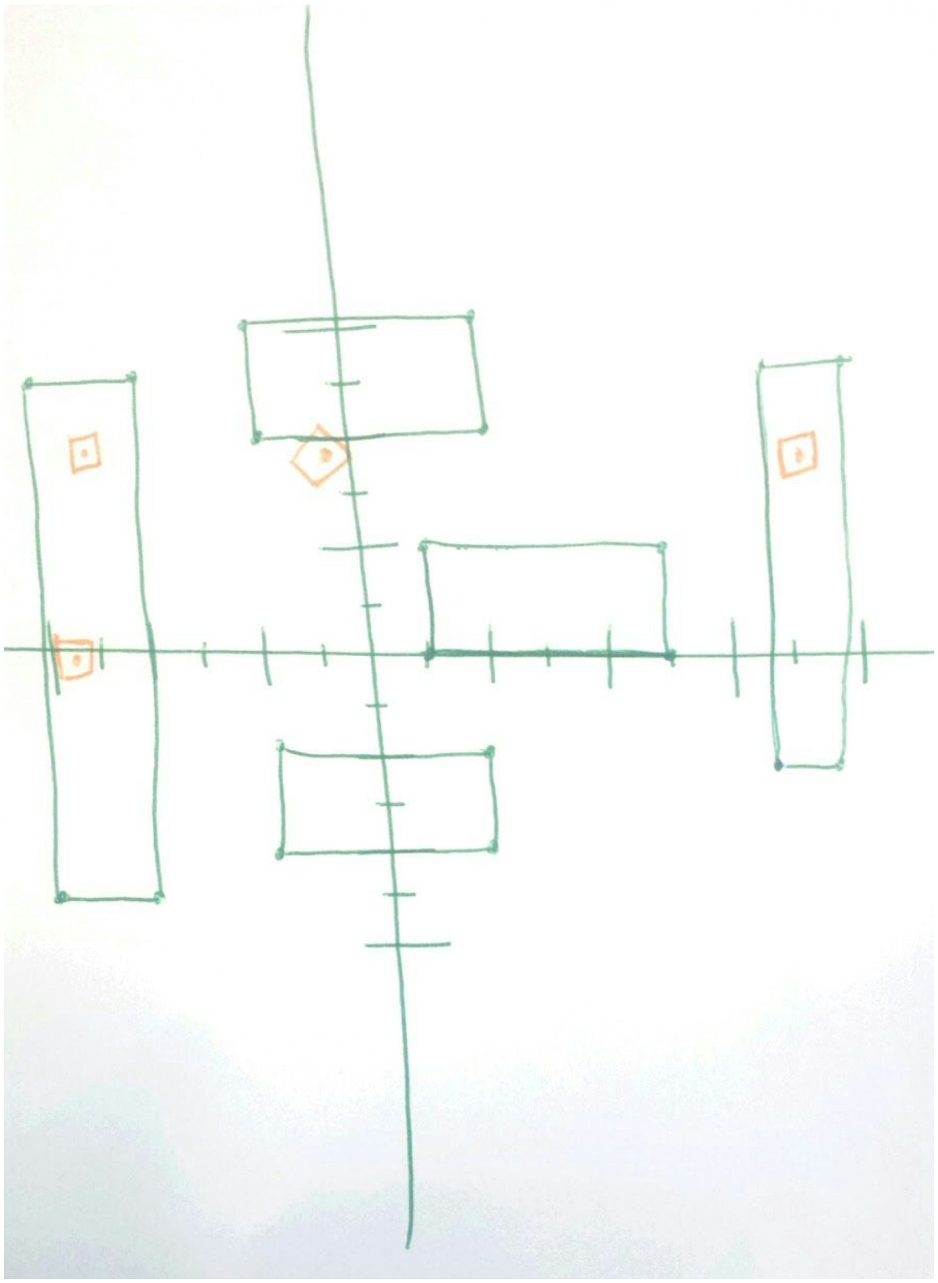
Figure 1: Visualization Software Output of Workspace and a Few Robots

4. Concluding Remarks

Implementing the point collision checking routine was trivial (1/10, 10 minutes). Implementing the circle was a bit harder since the configuration space was more interesting, but it was still straightforward (2/10, 30 minutes). The square was more difficult (5/10, 4-5 hours). Our logic was correct, but our mathematical implementation was a bit off at first and it took a while to debug. Initially we also forgot about a corner case in which the square lies entirely within an obstacle. Once we solved these issues, however, all tests cases ran quickly and correctly.

We enjoyed working together and easing into C++, and working with Linux instead of the abominable Windows command line was great because linux == bae. How to incorporate the matrix math presented in the handout wasn't immediately clear to us, and it took us some trial and error to figure it out.

Writing this document was simple and not incredibly tedious (1/10, 30 minutes).

What we learned from this project:

```
if (Linux == bae ) {
        std::cout << "<3<3<3\n";
}
```

As a haiku:

If Linux is bae
S-T-D-C-O-U-T
heart heart heart new line