**COMP 417 Assignment 1 Report**

**1A**

With uniform distribution, the tree spreads across the map in an unbiased way. For picking the nearest node I just checked one by one their distance to the random point. For obstacle detection, I used signed area of a triangle test.

**1B**

Using Gaussian distribution was a lot faster, because the trees tended to branch towards the target rather than all around the map. Finding the target should be a lot faster.

**1C**

Note: x-axis is the step size

Trends

Path length seems mostly unchanged. My theory is that with a smaller step size, deviations from the optimal path are less drastic, but there are more deviations. It’s like how walking north by 1m then east by 1m is the same as walking north by 0.5m, east 0.5m, north 0.5m, then east 0.5m. Path length does seem to get smaller from step size 100 onwards, this may have to do with the fact that the target circle is so large, that the tree is very easily drawn to it.

The bigger the step size, the less the number of iterations. Because with a larger step size, you take fewer steps towards the target. Also, radius around the target is larger.

Proposing a good step size

If you want speed and less computation, go for the higher step sizes. If you want granularity (more nodes and branches), go for the smaller step sizes. Granularity is important for going through narrow unstraight crevices.

A step size of 10 is a good blend of granularity and speed, since step size 5 is 3-4 times slower (way slower than comparisons of larger step sizes due to the exponential nature of the graph). Step size 20 gets an honorable mention for being twice as fast as 10.

Outcome of step size 10:

A picture containing diagram

Description automatically generated

**2A**

The code for this almost the same as the point robot, except that I included an extra dimension for theta. The middle of the line is the reference point. And then theta determines how much the line rotates around the reference point. Rotating pi will bring it back to where it started. I only allow it to rotate between negative half pi and half pi. To avoid having to calculate complicated arcs to determine obstacle detection, use small step sizes.

**2B**

Note: x-axis is the robot length

Note: I used a step size of 10. Ideally, I would have used something smaller like 5, but it would have taken a lot longer to gather the results.

For the more complicated world, a longer robot means a more iterations. That makes a lot of sense, with a longer robot, there’s a higher chance of the line hitting an obstacle, leading to more iterations that don’t generate nodes.

However, this isn’t true for the simpler world. Now why is that? I am attributing it to the variance of the data. If you take a look at the trials for shot.png in the associated spreadsheet, for line size 25, there is a trial of 1884 iterations and one of 9039. That’s 37% and 180% of the average. Now look at simple.png, for line size 35, there is a trial of 263 iterations and one of 6134. That’s 14% and 330% of the average. Therefore, I can’t make conclusions for the simple world with just 10 trials per line size.

Why is it more random in simple.png? For shot.png, the path is predictable, it pretty much always goes up the top to the right:

A picture containing icon

Description automatically generated

For simple.png, usually either two things happen. It finishes quick:

A picture containing shape

Description automatically generated

Or it takes a long long time:

Shape, square

Description automatically generated

And why is that? Because the path to the target is quite straightforward, so it can finish it very quickly. However, if it misses the target, the tree has more room to spread out than in shot.png. And that takes up a lot of iterations.