Path Planning Challenge Summary

I was assigned with the task of developing a path planning algorithm for a robot to traverse a set of uniformly distributed random points. The robot was constrained by its ability to only travel 3 units of distance before having to travel home and recharge. A number of solutions were explored and a proof of concept (POC) was developed for two candidates, A and B. Solution A (see `cobalt.py` and Figure 1), was the most developmentally simple and computationally efficient; it selected points at runtime by finding the closest point to the previously visited point, verifying in each iteration that the distance to that point and back home was within the 3 unit constraint. Solution B (see `solutionB_try.py` and Figure 2), explored the possibility of selecting points based on a directional vector and a calculated return path that was optimized to include the most unvisited points. Both of these solutions were programmed and compared with respect to computational performance (seconds to computer), total distance traveled by the robot, and the number of times that the robot returned home. Surprisingly, Solution A outperformed Solution B in all aspects.

After determining a best solution candidate, the program was rewritten to support class structures (see `cobalt.py`) to add an object oriented programming approach to the script. The use of class structures allows for easier integration of unit and systems testing later on. The abstract class `Solution` defines the base methods of a path planning solution with the only abstract method being `find_next_route`. This means any extending classes (i.e. `SolutionA`) differ in their path selection process, therein maximizing code reuse and making exploring new solutions easier. Solution B was first implemented as a script, but should it become a viable solution for a different edge case(s), it could be rewritten into a `Solution` subclass for testing and simulation. Command line argument parsing (lines 306-324 of `cobalt.py`) was added to improve the maintainability of the program (i.e. run `python cobalt.py --help` to see the usage of this program). Furthermore, PyDoc Documentation was added to provide an easy to read code summary for future developers (see `cobalt.html`).

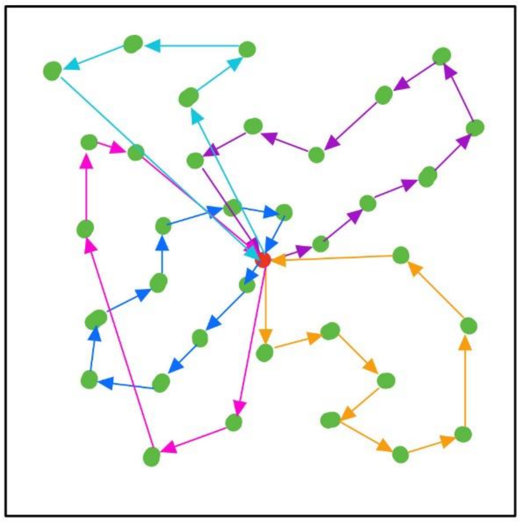

Figure 1 – Solution A. Expansion points are chosen by proxomity to last visited point taking into account the distance to that point and its return distance to home.

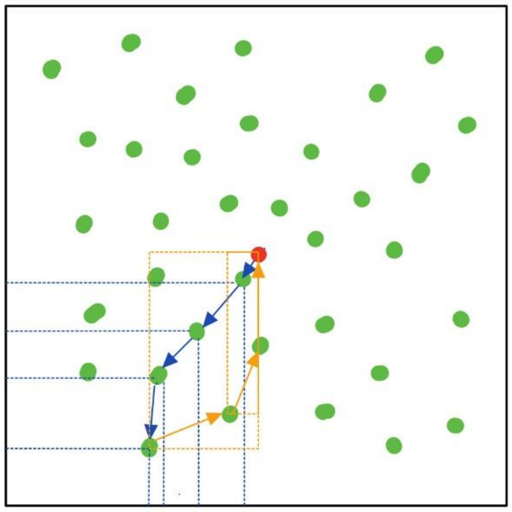

Figure 2 – Solution B. Expansion points are limited to vector quadrant and return paths are calculated at each point traversal (one iteration only)

In conclusion, this was a very rewarding and fun programing challenge. I enjoyed brainstorming and conceptualizing different approaches for tackling the problem, while at the same time imagining various use cases for certain solutions that have different business requirements (i.e. robot cleaning system with multiple passes, surveying robot that looks to minimize distance traveled, etc.). I developed two approaches for solving the problem assigned, Solution A proved to be the best approach as it out performed Solution B in time, distance and amount of times the robot went back home. Both solutions could be improved computationally by a linear factor through the incorporation of multithreading. This challenge allowed me to present some of my programming skills and provide a well-documented, maintainable and scalable program. I can see myself taking on more tasks like these and contributing to something bigger than me.