Path-Planning of Miniature Rovers for Inspection of the Hanford High-Level Waste Double Shell Tanks

Sebastián A. Zanlongo, Leonardo Bobadilla, Yew Teck Tan

Florida International University
11200 SW 8th St
Miami, FL 33199

szanl001@fiu.edu, bobadilla@cs.fiu.edu, yetan@fiu.edu

ABSTRACT
The Manhattan Project generated large amount of liquid nuclear waste. This waste is currently stored in million-gallon, double-shell tanks in Hanford. Over the course of 50 years, it is expected that the condition of the tanks would have deteriorated, and one of the tanks has already been found to have leaked. Currently, inspection of the tanks is limited to a visual inspection of the outer perimeter of the tanks using a pole-mounted camera or large wall-climbing robot. In this paper, we propose a methodology whereby a small inspection robot can efficiently navigate a series of small refractory slots located at the bottom of the tanks.

Keywords
robot, robotics, path-planning, path planning, radiation, deactivation and decommissioning, D&D, sampling

1. INTRODUCTION
During WWII's Manhattan Project, large amounts of nuclear waste were generated. Much of this was in liquid form, and stored in large, double-shelled tanks located at the Hanford Site, WA (Figure 2). Over the course of many years, it is expected that the condition of these tanks would have deteriorated. However, inspection of the tanks has been very limited. This is primarily due to the construction of these tanks, where the only way to meaningfully gather data is via visual inspection with a pole-mounted camera, or camera mounted on a wall-climbing robot [1], [2]. As these methods are unable to fully inspect the bottom of the tank, we propose augmenting this inspection to incorporate navigating a small tethered robot (Figure 1) [3] through the series of 2.5in x 1in refractory slots (Figure 1, Figure 3) sandwiched between the bottom of the double shells.

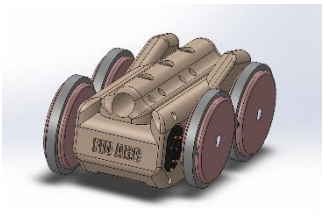

Figure 1: Mini-Rover Robot

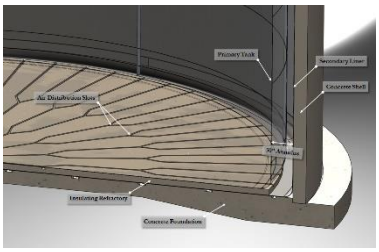

Figure 2: Tank Cutaway

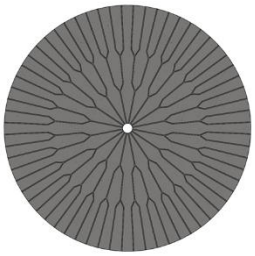

Figure 3: Slot Representation

1.1 Refractory Slots
The refractory slots follow a radial, forked structure. Initially, 16 slots radiate outwards from the center. These split into 32 slots, followed by 64 slots at the end. Apart from the large network formed by the refractory slots, note that there are 26 tanks to inspect.

1.2 Robot
The robot is represented as a point robot, present along one of the vertices or edges of the refractory slot graph. The robot has the following attributes:

- Position (x, y)
- Tether length
- Visit queue representing the locations to visit, in the order in which to visit
- Planned path to take to satisfy visit queue

Here, we present a path-planning algorithm with the goal of reducing the time needed for the robot to survey the tank bottom. The remainder of this paper is organized as follows: in Section 2, we review the related literature. In Section 3, the problem is stated, and the solution explained in Section 4. Simulation results are shown in Section 5, and the conclusion presented in Section 6.

2. RELATED WORK and PROBLEM STATEMENT
Traditional radiation mapping solutions often assume a 2-dimensional planar environment with little or no restrictions on the direction of movement [4], [5], [6]. Being limited to a series of

refractory slots incurs a much higher movement cost, and makes common strategies such as "lawnmower" (zig-zag) movement impractical. Moreover, many of the existing approaches neglect the presence of a tether which constrains the way in which the robot can move. A tether not only limits the distance a robot can travel overall, but may also force the robot to return to "reel in" some of the deployed tether so that it can reuse that tether to reach other locations.

3. PROBLEM STATEMENT

Given a robot, a graph representation G of the refractory slots, and a set of locations S at which we would like to sample, how can we visit these locations in such a way that we can satisfy the robot's constraints, and simultaneously minimize the distance travelled, thereby reducing the sampling time?

4. METHODOLOGY

Our approach is organized as follows: the refractory slot graph representation is discretized such that it reflects the level of resolution desired by operators. Next, sampling locations are selected using either a randomized approach, or an approximate cellular decomposition. Finally, we design a tour that visits each of the desired sampling locations.

4.1 Graph Discretization

The system of refractory slots is represented as a graph. Locations where the slots fork, or intersect the outer perimeter between the primary tank and secondary liner are represented as vertices V. The refractory slots themselves are indicated by edges E joining the vertices.

The starting graph representation G only has vertices where there is a fork or change in angle. As sampling takes place at vertices, we must introduce new vertices along the edges by discretizing them to the resolution required by an operator. This is done by:

1. Selecting a desired minimum edge length
2. Iteratively divide the edge into smaller edges
3. If remainder is less than length of desired edge, amortize it onto the edges

This method produces edges that are of slightly varying lengths, however the average lengths will closely approximate the desired length, and the process can be computed quickly. An alternative approach would be to simply divide the edges into the desired lengths, and leave the remainder as its own – smaller – edge.

The goal of discretization is to allow the operator to select the granularity with which the robot takes samples (samples are taken at each vertex). So, a finer graph with more vertices would indicate that the robot would also take more frequent samples. This is considered here to examine the effects – if any – on path-planning.

4.2 Sampling Strategies

4.2.1 Random Sampling

The simplest method for sampling the tank is to select a random set S of sample locations. These locations are selected by first choosing a random location in the 2-dimensional $\mathbb{R}^2$ environment, and then choosing the vertex in the graph that is closest to that point ("snapping" them into place). To sample these locations, we can visit them in the order in which they are created. Naturally this results in a lengthy completion time as there will be instances where two adjacent sampling locations are visited out-of-sequence.

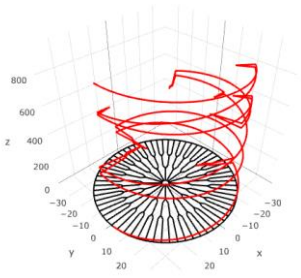

Figure 4: Robot movement over time (isometric view)

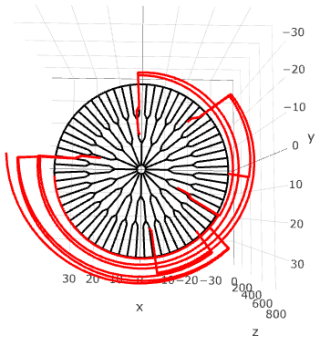

Figure 5: Robot movement over time (overhead view)

4.2.2 Random Sampling with Path Optimization

Given a random sampling technique, we can improve upon it by optimizing the *order* in which the locations are visited. This involves measuring the distances between locations, and selecting the path (or *tour)* that visits each of the locations while minimizing the total distance travelled. One consideration when operating in the refractory slots graph structure is that the graph is not fully connected, nor nearly fully connected. Moreover, travelling between adjacent vertices located in the interior of the tank requires that the robot first return to the perimeter before entering again. This means that a traditional Euclidean distance metric is not suitable. We must instead calculate the length of the path that the robot must take when travelling between the two points. Once this is performed, we can plan a tour that reduces total time.

4.2.3 Grid Sampling with Path Optimization

An alternative to randomly selecting samples is to use an approximate cellular decomposition [7]. This has the benefit of evenly distributing sampling locations regardless of the number of locations selected. Given the grid locations, we then "snap" the locations to the nearest vertex in the graph (as previously described). Next, we again design a tour that reduces total travel distance.

4.3 Path Planning

Provided the set of points to sample, we need to select the *order* in which to visit them. Optimal path planning between multiple locations is a known NP-hard problem as shown in the Travelling Salesman Problem (TSP) [8]. We therefore turn to a heuristic approach. Here, we have selected the nearest neighbor (NN) greedy algorithm. Beginning with the starting location of the robot, we perform Dijkstra's search [9] to each of the points $s \in S$ and measure the length of each path. Next, select the point that is closest, and set that as the next location to visit. Using this point, we then repeat the process on the remaining locations $s \in S$ until finished. On average, this heuristic provides solutions at most 25% longer than the optimal path [10], while remaining fast to compute.

We augment Dijkstra's algorithm with the constraints that the robot has:

- a limited-length tether
- to travel backwards to exit a slot
- to enter and exit via the same slot, before re-entering an adjacent slot (the robot cannot enter a slot and exit via another due to the space constraint at a fork)

In Figure 5, we show the pseudocode for the modified Dijkstra. The inputs to the algorithm are the adjacency graph G**, start** and **goal**

```
tethered_dijkstra (G, start, cost, goal, tether, max_length)
01: frontier ← PriorityQueue
02: came_from ← Dictionary
03: tethers ← Dictionary
04: frontier.put(start, 0)
05: came_from[start] ← cost
06: cost_so_far ← Dictionary
07: tethers[start] ← tether
08: while frontier not empty
09:     current ← frontier.pop
10:     for neighbor in G[current]
11:         tentative_tether ← tethers[current]
12:         if neighbor == previous_position
13:             tentative_tether.pop
14:         else
15:             tentative_tether.append(current)
16:         new_cost ← cost_so_far[current] + movement_cost
17:         if (length(tentative_tether) < max_tether_length)
and ((neighbor not in cost_so_far)
or (new_cost < cost_so_far[neighbor])
or (length(tentative_tether < length(tethers[neighbor]))))
18:             cost_so_far[neighbor] ← new_cost
19:             frontier.put(neighbor, new_cost)
20:             came_from[neighbor] ← current
21:             tethers[neighbor] ← tentative_tether
22: if goal in came_from
23:     path ← reconstruct_path(came_from, start, goal)
24:     return path
25: else
26:     return None
```

Figure 5: Tethered Dijkstra Pseudocode

locations, **cost** of reaching the current location, the current **tether** occupancy, and the **max length** of the tether. In lines 1 – 7, we initialize a frontier priority queue that contains the vertices to explore in the order provided by a priority heuristic. The *came_from* dictionary contains the relationships showing how vertices are connected to each other; *cost_so_far* indicates the cost to reach each explored vertex, and the tethers dictionary shows the cells occupied by the tether to reach each explored vertex. Each of these is initialized with the starting location of the robot, the tether occupancy, and the cost to reach the current location.

In line 8 and 9, we pop the frontier for unexplored vertices. Line 10 checks each neighbor of a vertex that is currently being expanded. In lines 11 – 15, we keep track of the stack of the *tentative_tether*, extending or shortening it as the robot moves along. Line 16 considers the cost of reaching the neighbor given the cost already expended, and the additional effort of moving to the next neighbor. In line 17, we check to see if the route under consideration exceeds the maximum tether length. If not, we also check if that neighbor has not already been explored, or if the route under consideration has a lower movement cost or tether length than the previously examined route. If these conditions are met, save the new route in lines 18 – 21. Finally, we check in line 22 to see if the goal has been reached. If so, we reconstruct the path from the saved neighbors in lines 23 – 24. Otherwise, no valid path exists, and we return None in lines 25 – 26.

The solution here only takes the robot between two locations, **start** and **goal**. To plan a path through all the desired sample locations, the planning process is repeated in a sequential manner where the goal location of the previous search is assigned as the start location for the next search, until all the sample locations have been visited. The corresponding cost and the tether occupancy of the robot are also updated along the searching iterations. Figure 4 shows the movement of a robot throughout the slot network, with the z-axis representing the order of sampling sequence.

5. SIMULATION SETUP

The algorithms described in Section 4 were implemented in Python, and tested using multiple scenarios.

These simulations studies consisted of:

- Discretization scales of: 1, 2, 4, 8
- Tether lengths: 200, 300, 400, 800
- Strategies: Random Sampling, Random Sampling with Path Optimization, Grid Sampling with Path Optimization

The discretization scales were selected as they cover the range of edge lengths. At a scale of 1 unit, the smallest edges were unaffected, while the longer edges were discretized; at the 8-unit scale, longer edges were also unaffected.

Tether lengths were chosen so that at 200, the robot had the minimum tether needed to reach every location in the graph. At 300 and 400, the robot would be able to take longer paths before having to return, with 800 providing significantly more play in the tether.

In each of the studies, the robot was tasked with visiting 64 locations. Each of these trials was repeated 40 times, so that the results of the random locations would average out. This gives a total of 1,920 trials.

6. RESULTS

In Table 1, we show the distance travelled by the robot with different tether lengths under different strategies. As expected, the grid approach does not change as it always follow the same shortest path across trials. Tether length did not contribute significantly to the total distance travelled, regardless of the strategy. This is likely due to the proximity of the sampling locations, where locations that are far apart require the entire tether to reach, and nearby locations being so close that the tether did not need to be readjusted.

Table 1

Distance Travelled (average over scales)			
Tether Length	Random	Random TSP	Grid TSP
200	5731.307951	2070.933453	1829.821503
300	5636.790104	2084.391256	1829.821503
400	5692.83821	2063.240204	1829.821503
800	5703.329017	2099.663685	1829.821503

Next in Table 2, we look at the distance travelled by the robot with different graph discretization scales. As expected, the results

closely mirror the results from Table 1. The discretization resolution should only affect the granularity of sampling, and the results of path-planning should not be affected. This confirms the intuition that granularity of the graph can be tuned as needed depending on how the operator wishes sampling to be performed, and can be modified independently of the path-planning strategy.

Table 2

Distance Travelled (average over tethers)			
Scale	Random	Random TSP	Grid TSP
1	6069.814548	2092.21192	1844.3353
2	5819.408242	2062.5826	1992.456128
4	5607.091291	2067.416216	1892.107295
8	5267.9512	2096.017862	1590.387289

Finally, Table 3 shows the distance travelled by the robot under all three strategies, averaged over all trials, scales, and tether lengths. Here, we see that the random TSP solution provides a 63.46% improvement on the completely random approach, while the grid TSP provides a slightly better 67.85% improvement.

Table 3

Distance Travelled (average over scales & tethers)		
Random	Random TSP	Grid TSP
5691.066321	2079.55715	1829.821503

7. CONCLUSION

In this paper, we have provided a methodology for path-planning for a tethered robot to navigate the series of refractory slots located at the bottom of the Hanford site tanks. We show how Dijkstra's algorithm can be modified to incorporate a tether constraint, and how this search strategy can significantly reduce the distance that the robot travels, and by extension, the amount of time needed for the robot to survey the tank bottom.

In the future, we would like to focus on:

- Using other path-planning strategies such as A* and Christofides algorithm [11] with the additional tether constraint, and comparing these approaches
- Analyzing how these approaches affect the quality and amount of data gathered by the robot
- Consider additional constraints such as the cost of rounding a sharp corner.
- More analysis of how different tether lengths affect path-planning, especially in edge cases.

8. ACKNOWLEDGMENTS

Mr. Zanlongo would like to thank FIU's Applied Research Center (ARC) and the Department of Energy Office of Environmental Management for providing the opportunity to be a DOE Fellow under the Science and Technology Workforce Development Initiative and facilitating research opportunities under the DOE-FIU Cooperative Agreement DE-EM0000598.

9. REFERENCES

[1] C. Girardot, J. Engeman, D. Washenfelder, and J. Johnson, "Double-Shell Tank Visual Inspection Changes Resulting from the Tank 241-AY-102 Primary Tank Leak – 14193," *Waste Manag. Conf.*, 2014.

[2] D. Bryson, V. Callahan, M. Ostrom, W. Bryan, and H. Berman, "Life Extension of Aging High Level Waste Tanks," *Pap. HL*, 2002.

[3] R. Sheffield, D. McDaniel, and S. Tosunoglu, "Development of a Prototype Miniature Motorized Inspection Tool for Hanford DOE Site Tank Bottoms," *Florida Conf. Recent Adv. Robot.*, 2015.

[4] C. Cai, B. Carter, M. Srivastava, J. Tsung, J. Vahedi-Faridi, and C. Wiley, "Designing a radiation sensing UAV system," *2016 IEEE Syst. Inf. Eng. Des. Symp. SIEDS 2016*, pp. 165–169, 2016.

[5] J. Towler, B. Krawiec, and K. Kochersberger, "Terrain and Radiation Mapping in Post-Disaster Environments Using an Autonomous Helicopter," *Remote Sens.*, vol. 4, no. 7, pp. 1995–2015, 2012.

[6] P. G. Martin, S. Kwong, N. T. Smith, Y. Yamashiki, O. D. Payton, F. S. Russell-Pavier, J. S. Fardoulis, D. A. Richards, and T. B. Scott, "3D unmanned aerial vehicle radiation mapping for assessing contaminant distribution and mobility," *Int. J. Appl. Earth Obs. Geoinf.*, vol. 52, pp. 12–19, 2016.

[7] H. Choset, "Coverage for robotics–A survey of recent results," *Ann. Math. Artif. Intell.*, pp. 113–126, 2001.

[8] C. H. Papadimitriou, "The Euclidean travelling salesman problem is NP-complete," *Theor. Comput. Sci.*, vol. 4, no. 3, pp. 237–244, 1977.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.

[10] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study in local optimization," *Local search Comb. Optim.*, pp. 215–310, 1997.

[11] N. Christofides, "Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem," *Rep. 388, Grad. Sch. Ind. Adm. C.*, no. February, 1976.