Robotic Guard Dog

Gabriel Gladstone, Alexander Sosnkowski, Ryland Buck, Kyle McDonald
Autonomous Mobile Robots

Abstract
We implement a Robotic Guard Dog system to protect a bounded environment through autonomous motion planning and area coverage. We utilize a standard SLAM implementation to generate a pgm file that is fed into a skeleton generation tool from which an optimal patrol path can be extracted in real-time.

1 Introduction

As secure facilities grow larger and cities more populous, it becomes increasingly more difficult to maintain teams of humans to patrol and ensure the security of important buildings - especially at odd hours of the day. Other facilities that require constant in-person monitoring, such as power plants and industrial production sites, can be hazardous and dangerous for human employees. Thus, it is desirable to have autonomous systems capable of patrolling complex environments with total visual coverage. Our proposed system not only reduces the amount of cameras needed to secure an environment, but is ideal for securing temporary and remote environments that don't have the infrastructure to support a complex surveillance system.

2 Theoretical Methodology

We considered many methods for full-coverage path planning. Our first decision was to tackle this problem from discrete information like a grid map or from topological information. Some grid map methods we found were coverage path planners, algorithms that would plan paths to cover an area such as fields for farming purposes[4]. We considered this approach, but because the guard dog can detect objects within a radius, and not just immediately in front of it, we believed this method didn't directly address our problem.

We decided that transforming grid map information to topological information would be the best approach. A topological map represents the environment with vertices and edges. We believe this is a more robust and simpler representation of the environment, as vertices with straight edges provide an abstraction and noise reduction of the grid map information.

Topological information can be used to solve the External/Internal Watchman Route Problem[5], the Art Gallery Problem, and the Fortress Problem[3]. All analogues problems to the one we are attempting. Algorithms addressing these problems use topological information to represent the obstacles, vertices at each corner of the obstacle, and edges connecting them. If a guard dog was positioned at a subset of these vertices, it would be able to fully cover the

environment. This strategy is simple enough for rectilinear monotone environments[2], but when holes and more complicated shapes are introduced, the problem becomes non-trivial. Another problem with this approach is that it is assumed the watchman can be located exactly on the vertex, but this is not the case as the vertex is an obstacle.

Our research led us to representing the topological information of the environment as a filtered Voronoi diagram. This method would give us a skeleton of the grid map, essentially ideal paths in the center of the environment to navigate. By navigating this skeleton, the robot would be able to monitor the full environment without actually traversing the full environment as was the problem with coverage path planning[6]. The skeleton could be dynamically generated from any complex-shaped environment quickly and extraction of the graph information of this skeleton would be much simpler than an ideal solution for the analogous problems. While pathing using this method could be slightly less optimal than using vertex information of the actual obstacles, we thought this method was a good middle-ground for its simplicity, speed, and thoroughness of environment coverage.

3 System Description

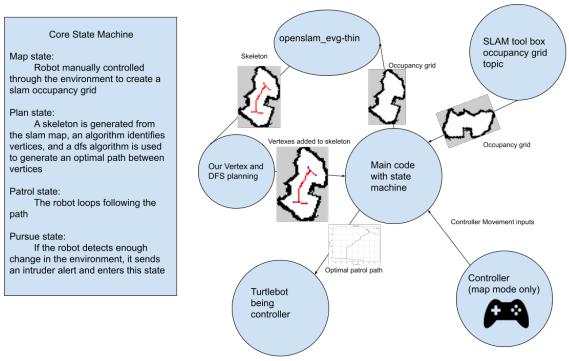

Figure 1: Overall System Design

A diagram of the system implementation can be found in figure 1. The steps outlined ensure robust path planning while also providing modularity making it easy to adjust / modify individual stages. The system is implemented as a four part state machine. Initially, the robot begins in a mapping phase where a user manually controls the robot and maps out the environment they wish for the robot to patrol. Once the user presses the X button to confirm they are done mapping, the robot move into the planning state. Here, it saves the SLAM occupancy grid, gives the saved PGM file to EVG-Thin, receives back a skeleton (red path) PGM file that it can then run a vertex finding algorithm (blue dots) to get a final vertex skeleton. Next, it computes an optimal path between all vertices via DFS. Once it has computed this patrol path, it moves into its patrol state where it will loop through the path until it detects an intruder (which sends an alert / activates the pursue state).

4 Design and Implementation

4.1 Initial Mapping

In later versions of this system, it is possible to support autonomous exploration of a bounded-environment. However, we have decided for the sake of flexibility and time to allow the user to define the environment. In the initial exploration phase, the user will remotely control the remote as it maps the environment with SLAM. After mapping is completed they will press a button on the

controller to finish the move to the next state. The saved occupancy grid will then be used to generate the path and motion planning for the guard dog system. In lab 3 we designed a custom rudimentary SLAM system, but have decided to use SLAM TOOLBOX for this project because of its robust state estimation handling, and ease of export to PGM files[1].

4.2 Skeleton Creation

With the bitmap stored as a PGM file, we used the EVG-THIN tool to generate a skeleton of the bitmap. EVG-THIN implements a Voronoi diagram on an occupancy grid with added parameters that affect the shape of the skeleton. A Voronoi diagram is a partition of a plane into regions that are equally spaced from a set of objects.

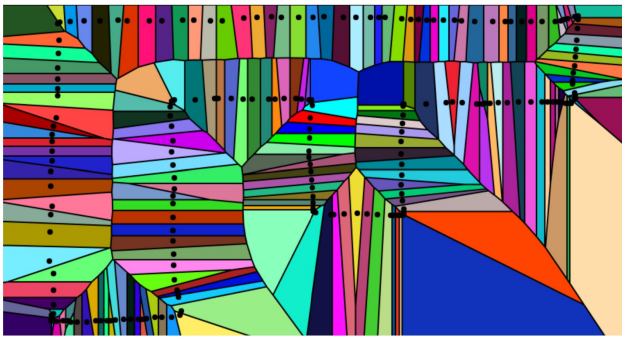

Figure 2: Voronoi Diagram

EVG-THIN will create a Voronoi diagram, then eliminate paths that are too close to objects or don't meet other parameters such as not connected to the robots initial location. A skeleton will then be generated as shown.

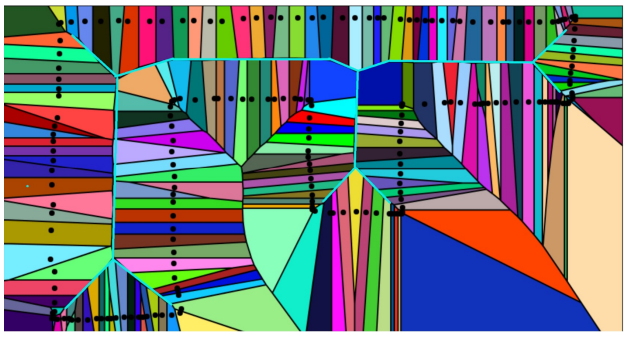

Figure 3: Possible EVG-Thin Thinning (Light-Blue)

We have tested the EVG-THIN tool in different environments and believe that specifying *-min-distance*: minimum distance of a line to an object, *-robot-loc*: location of the robot within the grid map, and *-pruning*: turn off pruning which removes edges that don't touch the unknown cells of the edge of the grid. The results of our tuning can be seen below. As shown in the images, EVG-THIN provides accurate and fast skeleton creation that will enable our guard dog to monitor varied environments.

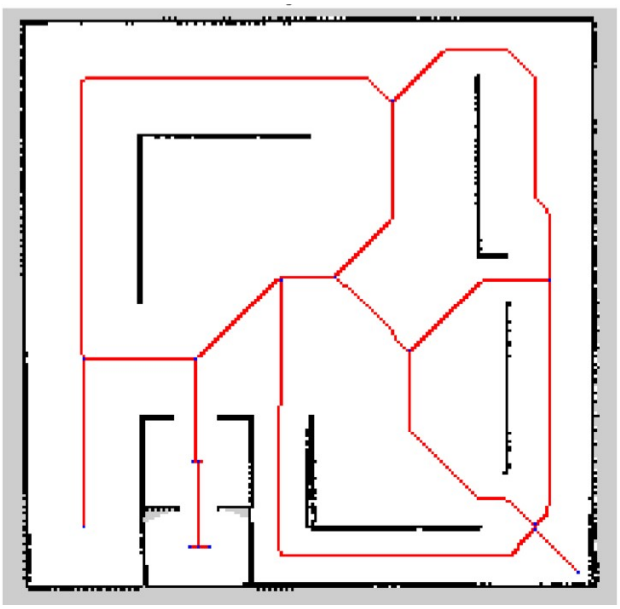

Figure 4: EVG-Thin Pathing (Arena)

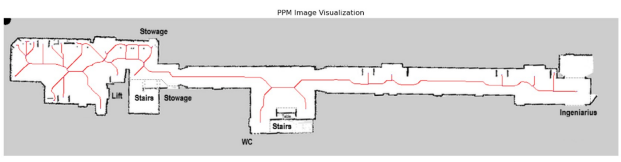

Figure 5: EVG-Thin Pathing (Building Floor)

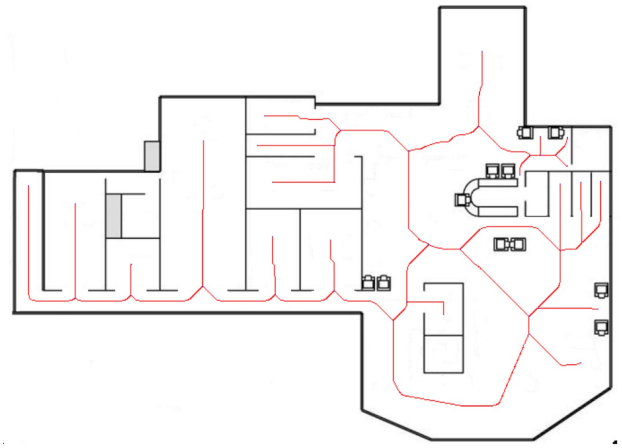

Figure 6: EVG-Thin Pathing (Home Floor)

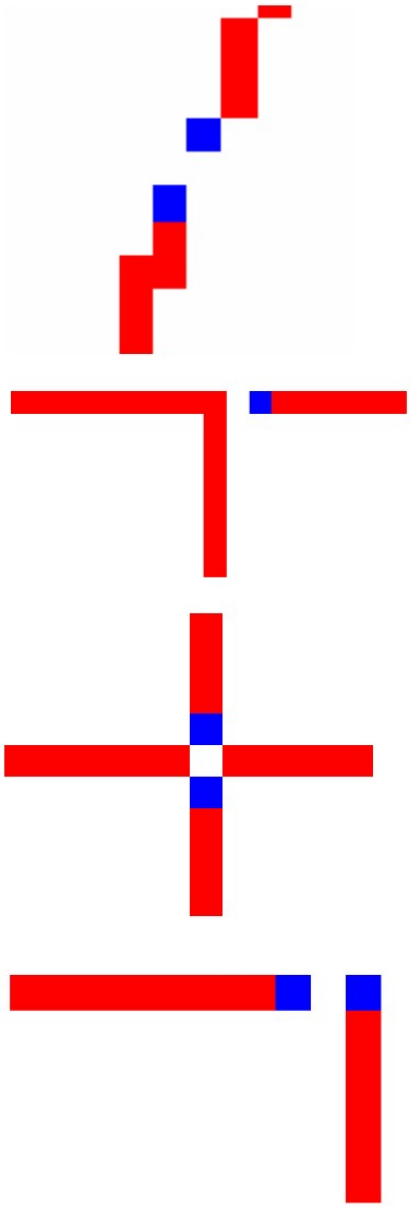

Figure 7: Missing Red Pixels Effect on Vertex (Blue) Generation

4.3 Skeleton / Bitmap Cleaning

The generated skeleton provides good coverage of the environment for monitoring. However, the skeleton is not perfect and occasionally has missing pixels, which will mess up our graph formation and thus our traversal of the environment.

Shown in Figure 7 is the effect of a missing pixel on our vertex (blue pixel) formation. Our graph formation algorithm assumes that all paths are continuous and 1-pixel wide. If a path ends early, that is considered a dead end. This is a problem, as vertices not connected by a path (red pixels) will not be neighbors and thus a non-optimal path will be generated in the path planning step.

To add in the missing red pixel before graph formation, we had to handle multiple configurations a missing pixel could take,

while maintaining the 1-pixel wide path assumption. We solved this by analyzing a 9-pixel window centered on each pixel to check if the red pixels in the window met certain count and adjacency checks. The resulting effects on the graph can be seen below.

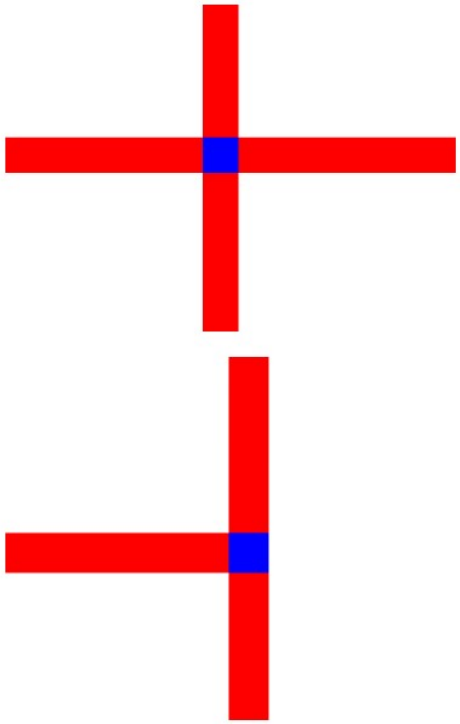

Figure 8: Added Red Pixels Effect on Vertex (Blue) Generation

4.4 Graph Formation

There are two parts to extracting a graph from the occupancy grid: first we determine vertices of the graph, then we find neighboring vertices. The resulting graph will allow us to generate optimal path planning for environment coverage.

To determine vertices of the graph, we followed the methodology outlined in Portugal's 2012 paper on extracting topological information from grid maps. Essentially, we analyze a 9 pixel window around a given pixel and using the count of red pixels within that window, the count of sets of adjacent 2 pixels around the center, and the count of adjacent 3 pixels around the center, we are able to determine if a given pixel was a vertex at an intersection, or a dead end. Additional logic was added to ensure that vertices weren't adjacent.

To extract neighbor information, we performed a BFS search along each adjacent path of a vertex to the closest neighboring vertex. Even though we assume that paths are 1-pixel in length, we added an adjacency check around the vertex to ensure that the BFS search doesn't traverse an unexpected path. We also store pixel positions of each path between vertices for easy motion planning. From these techniques, we generate an adjacent list with vertex, neighbor, path, and position information to be able to map all points to the physical environment.

4.5 Path Planning

At this point, we have extracted a weighted graph from the skeleton and want to traverse the graph in a way to reach all nodes while minimizing distance. The problem is a form of the traveling salesman problem (TSP). However, TSP assumes that all nodes will be traversed only once and that cannot be guaranteed in our graph. We wanted a simple and fast solution to this problem, so we decided to implement a form of the nearest neighbor algorithm. We used the following algorithm:

1. Mark all nodes as unvisited.
2. Choose a node at random.
3. Find the shortest neighbor and set this to the current node.
4. Repeat step 3 until no unvisited neighbor is present at the current node.
5. Backtrack to the previous node and repeat step 3.
6. Repeat step 5 until all nodes have been visited.

This solution is a DFS nearest neighbor + backtracking greedy approach that will produce a somewhat optimal vertex traversal

order (tour). The robot will follow red paths between vertices to traverse this tour. If the next vertex in the tour is not adjacent to the current vertex, we find the best path from one vertex to another with A*. The robot follows the red pixel paths because in a static environment, these paths are guaranteed to avoid obstacles.

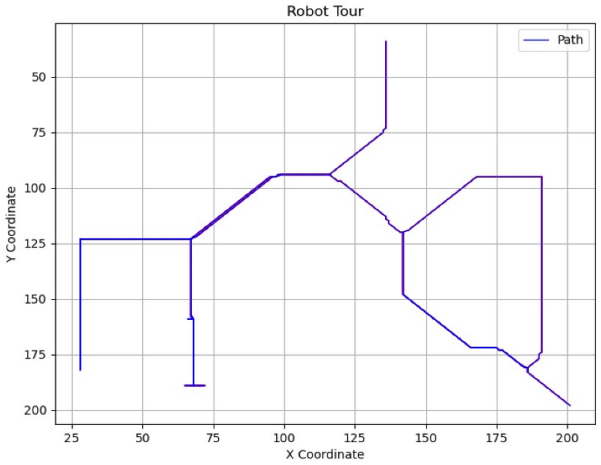

Figure 9: Robot Tour of Arena

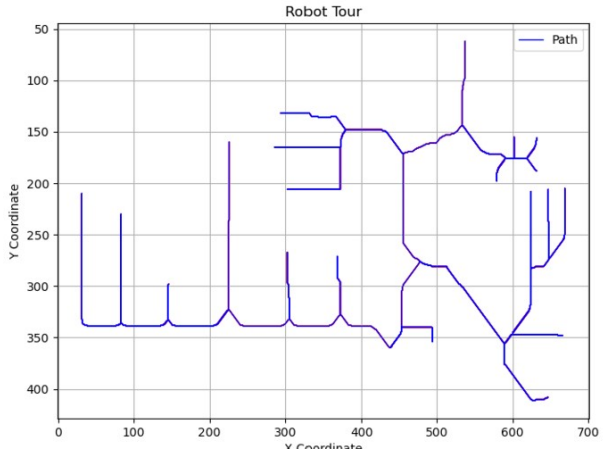

Figure 10: Robot Tour of Home Floor

Figure's 9 and 10 show generated tours starting at vertex 0 from their associated skeletons. It can be seen in the tour of the Arena that the top left route is ignored because it is long. However, even though some paths are ignored, by visiting all vertices, the robot should be able to cover the full area. Because there are less multiple connections, it can be seen in the tour of the Home Floor that most of the skeleton is traversed.

4.6 Obstacle Detection

We attempted a number of different obstacle (intruder) detection methods during the testing of this project. Our first approach was to use the difference between the occupancy grid created when the area was first mapped, and the current occupancy grid being updated as the robot drives along the skeleton path. We found that during travel, the differences in the map became too large to use as a method of detecting only a small change in the map. To combat this issue, we attempted to decay the original map over time replacing the original map with the updated occupancy grid by a percentage every iteration (i.e 80 percent old, 20 percent new). This helped to combat the large errors, but we found that during movements like turning, it still introduced large enough instantaneous differences in the map that it couldn't be used effectively as a method of detection. We eventually settled on a rudimentary approach that used the IR sensor to monitor intensity in order to detect unaccounted for obstacles (which indicate a change from the initial mapped state, and thus an intruder). While this was not an ideal for our approach since it limits the robots range of detection, it was still able to reliably detect obstacles that appeared on or around the path and trigger our state machine to create a security alert that there was an intruder. Due to the robustness of our path planning and its coverage of all areas within the map, we believe this to be sufficient.

5 Results

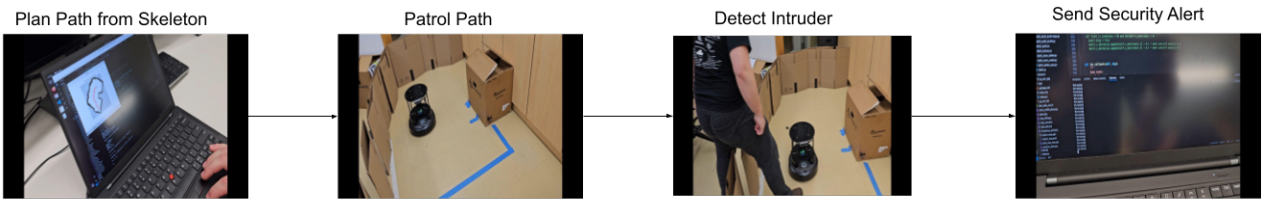

Figure 11: Testing Results

Our robot performed sufficiently when tested against a number of different maps / configurations created within our robotics lab. One such example run can be found in figure 11. For each of them, the robot was able to successfully map the area using the SLAM library and produce a path skeleton using the occupancy grid output. The robot was then able to patrol the area in a repeated pattern until a potential intruder entered the area it was patrolling or was manually stopped during testing.

6 Conclusions

Our proposed algorithm offers robust path planning for complex environments. It is capable of quickly generating optimized patrol tours that navigate safely around obstacles and offer full visual / sensor coverage of a designated area. This approach can be utilized to solve numerous challenges (including domains outside of security) that require continuously monitoring an area via an autonomous mobile robot. Our proposed system is modular and easily adaptable while remaining robust with minimal computational cost.

7 Future Work

An autonomous robotic guard dog system is non-trivial. There are many challenges in this system that could be expanded, improved upon, or customized. Our team's primary focus was to make a minimal viable product of a guard dog system focused on optimizing the patrol path to minimize distance patrolled.

One aspect of the system that could be expanded upon is the initial exploration. Effectively exploring complex terrain is a problem in itself, so to simplify this phase, we had a human operator remotely control the robot. An expansion of this project would replace this phase with autonomous exploration algorithm.

Another aspect that can be expanded upon is the obstacle and intruder detection. As this wasn't the primary focus of our project, we kept the implementation pretty minimal. A future work could expand upon this, add tracking for the robot, and video recording. Detecting when the environment has changed because of an intruder (i.e when to send a security alert) is also non-trivial as it

must account for drifting occupancy-grids and environmental noise. Machine learning techniques on Lidar time series data or computer vision models could prove to be effective.

Future improvements to the project could also include making dynamic parameter changes with the EVG-THIN tool, custom tailoring the source code of the EVG-THIN tool, or handling the path generation through an entirely different method. For example, a traveling salesman approach that is more optimal. There is a lot of work to make this system more robust, user-friendly, and usable in a real-world environment, however, we hope our project establishes a strong foundation to build upon.

8 References

References

[1] ROS package: nav2_map_server.

[2] W Chin and S Ntafos. Optimum watchman routes. In *Proceedings of the second annual symposium on Computational geometry - SCG '86*, pages 24–33. ACM Press.

[3] Lawrence Erickson. The fortress and prison yard problems in arbitrary 2-manifolds.

[4] Gonzalo Mier, João Valente, and Sytze de Bruin. Fields2cover: An open-source coverage path planning library for unmanned agricultural vehicles. 8(4):2166–2172. Conference Name: IEEE Robotics and Automation Letters.

[5] Simeon Ntafos and Laxmi P Gewali. External watchman routes.

[6] David Portugal and Rui P. Rocha. EXTRACTING TOPOLOGICAL INFORMATION FROM GRID MAPS FOR ROBOT NAVIGATION:. In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence*, pages 137–143. SciTePress - Science and and Technology Publications.