Multi-Robot Persistent Coverage Under Fuel and Stochastic Failure Constraints

By:

Yaşar İdikut
Camden Cummings
Samara Holmes

An Undergraduate Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the Degree of Bachelor of Science in

Computer Science,
Robotics Engineering

Project Advisors:

Carlo Pinciroli

Sponsored By:

BAE Systems

April 2024

APPROVED:

Carlo Pinciroli, Advisor

Abstract

In this project, we study robot swarms' application to agriculture. Specifically, we focus on (i) persistent coverage, (ii) fuel constraints, and (iii) failures. We present a baseline Mixed-Integer Linear Programming (MILP) model, as well as heuristic algorithms trading the optimality of MILP for scalability and resilience against failures. We compare the solutions based on metrics about visitation frequency and coverage. Finally, we present visualization, simulation, and real-world evaluation (using Khepera IV robots) to validate our solutions. A summarized version of our methodology and results is available at https://nestlab-bae-mqp-2024.github.io

Acknowledgements

We would like to thank Dr. Carlo Pinciroli, Dr. Benjamin Cooper, and Dr. Daniel Zwillinger for guiding this project and showing tremendous support throughout its completion.

Contents

1 Introduction

1.1 Robots in Agriculture

Robotics being applied to the agricultural field is a familiar idea. The three major applications of robotics in agriculture include (1) planting and field preparation, (2) weed management, and (3) harvesting and processing [1]. As the global population grows, the agricultural sector faces the impending challenge of how to sustainably and efficiently feed the population. The integration of robotics in agriculture provides a compelling solution. Robotics promises increased precision, reduced labor demands, and possibly more sustainable practices. The main hurdles that come along with applying robotic technology to agriculture include the need for more dexterity in the field, the lack of ability to scale to the size of a larger operation, and the lack of affordability and efficiency [2].

One key issue in applying robots to agriculture is the ability to scale to the size of an expansive farm. While small robots may excel in controlled environments or small farms, the ability to meet the requirements of a larger agricultural operation provides a vastly different challenge. To attend to the requirements of a larger-scale operation, there needs to be a balance between the costs and precision of the system. Larger operations generally require integration with other agricultural machinery and processes which can also hinder the ability to solve the scalability issue.

Lastly, while robotic technology over a long period can result in large cost savings with the reduced need for human labor, the upfront investment in a robotic system can be difficult for many farms to handle. Additionally, the robotic system needs to be as robust as possible to avoid halting processes and requiring frequent maintenance. As the robotic system grows and becomes more complex, it becomes increasingly important to have a robust fleet of robots that need little to no supervision. For a successful, scalable robotic system to be applied to agriculture, it needs to be robust and provide reasonable returns on investment.

All of these challenges in agriculture robotics push the solutions towards multi-robot or swarm systems. These systems can be set up to provide a parallelized non-stop workforce, be prone to an individual agent's failure, and be less costly to maintain if designed robust enough both on the system and individual level. Approached from this perspective, the problem at hand is a constrained persistent coverage problem with multiple robots.

1.2 Persistent Coverage Problem

The primary goal of this project is to achieve persistent coverage. Persistent coverage is defined as the capacity for a robotic system to indefinitely surveying an area. We add two more constraints in to the problem: fuel and failures. Each robot has a certain fuel capacity, so it constantly needs to refuel to continue covering. When a robot fails, other robots can continue surveying the whole area. Resilience to failure is key to any long-running robotic system and is considered especially important for robotics applications in agriculture. Key considerations when designing for persistent coverage in agricultural robotic swarms include the level of decision-making, redundancy, adaptive coordination algorithms, and communication.

Decisions can be mainly taken at the system and agent levels. A solution can either have a centralized agent make the decisions and forward them to the agents, or those decisions can be made at the agent level. It is thought that a centralized solution is easier to implement but decentralized solutions usually thrive in robustness. Swarm robotics is a sector of multi-robot systems (MRS) focusing on implementing an exploration strategy in a decentralized manner. The importance of using a decentralized system is to prioritize flexibility, scalability, and fault-tolerance [3]. Robots in a swarm are going to fail in any number of ways, and designing algorithms that are resilient to failure is vital. Swarms can be useful when performing several small tasks to work towards a greater goal. These characteristics make it especially interesting when looking to apply robotic swarms to agriculture.

The level and degree of communication would also affect the decision-making capabilities and robustness of individual agents. If complete communication is assumed, each agent would be able to make any centralized decision on its own, however, this solution could lack robustness in the absence of this communication. On the other hand, if communication between agents is constrained, the performance and efficiency of coverage can damaged.

Redundancy is crucial in enabling resilience in the robotic swarm. Whether these failures happen through the environment, hardware issues, or even battery depletion, having a backup for these failures will be necessary [2]. Additionally, an online algorithm would be needed to detect failures and re-plan paths. It is important to note that both unpredictable and predictable failures need to be accounted for. Predictable failures can be failures relating to battery depletion, charging station crowdedness, and scheduled software updates where the robot becomes inactive. Unpredictable failures are failures relating to environmental changes, hardware issues, etc, and can't be accounted for, but can be minimized through proper sensor usage.

Adaptive coordination algorithms are a necessary application to this project's success because

agents will need to work and complete tasks in a dynamic and uncertain environment. Such algorithms will allow the agents to adjust in real-time to optimize their movements. This optimization can help with task allocation, battery depletion, and navigation in the ever-changing environment.

1.3 Problem Statement

We take an environment that can be divided via cellular decomposition into several target positions that robots must visit. These robots have a constant velocity and a fuel capacity. Robots must 'recharge' by going back to a charging depot before they run out of charge. There is assumed to be one depot in the environment, in one corner of the map. All the robots are assumed to start at this position.

Formulated as such, the problem is similar to the Multi-Robot Coverage Problem proposed by Mitchell et al. [4]: here we introduce one more complexity, failures. Failures are assumed to occur at unknown times, and the chance of robot failure is calculated per robot, and is a constant rate. We assume that when a robot fails, another one is instantiated at the depot, at some time after the robot fails. After a period of being in the "failed" state, the robot becomes healthy again and joins the coverage efforts. Our goal is to design and evaluate algorithms that allow for persistent coverage under stated constraints.

1.4 Contributions

In this project, we present novel strategies to persistently survey an area. The problem we solve has been formulated with fuel constraints and failure constraints. The combination of fuel constraints and failure constraints with persistent coverage has not been shown previously. Typically, two of the three are examined. To solve the problem, we provide a MILP formulation and two heuristic algorithms. We provide validation of our algorithms through point mass simulation, physics-based simulation, and real-robot testing.

2 Related Work

The problem presented is, at its core, a coverage task. We consider multiple robots with energy depletion, and both an offline and an online method to handle failures. These constraints make this problem unique. Problems and solutions similar to ours exist, however, energy depletion [4] [5], failures [6] [7], a large and non-infinite time [4], and real-world testing have not been considered all together, and provide a unique set of challenges. In particular, in the literature covered, we found that failures occurring in persistent coverage problems were not typically considered.

A very similar problem with a MILP-based optimal solution and a heuristic is presented by Mitchell et al. [4]. Initially, a new category of problem is defined. They describe the Multi Robot Persistent Coverage Problem (MRPCP). In this type of problem, a number of robots must go to a set of target nodes. These robots have a limited fuel capacity and must operate for longer than their fuel capacity allows. Our problem is a type of MRPCP, with the variation that we consider the possibility of robots failing.

Mitchell et al. [4] solve the MRPCP in the discrete state-space by constructing nodes (of two types: target and depot) and requiring robots to travel through them. Their MILP-based optimal solution is formulated through an objective function that minimizes the maximum path length for each robot with given constraints. The heuristic function calculates a single sub-optimal Hamiltonian path and then splits it between multiple robots. The problem considers a linear battery depletion model and multiple charging nodes. However, these solutions lack scalability with high-resolution maps or maps with many nodes. Given a highly complex computation time, it is not practical to re-plan each time an agent in the multi-robot fleet fails. In our testing, we find that the MILP-based solution struggles to find solutions especially when the fuel capacity is low. We extend their formulation to fit to our problem, then use it as a baseline.

A similar problem with a solution in the continuous state-space is presented by Meng et al. [5]. Here, each point in a 2D map is assigned a value based on the product of that point's closeness to the robots for all robots. This way, to maximize the objective function, robots are encouraged to travel in the direction that maximizes each point's visitation and sample-collecting probability. Each robot travels in the highest gradient of the reward function until the battery levels become low enough to require recharging. Before this point, robots communicate with the charging station which requires scheduling. The battery depletion is modeled as a function of speed squared, which is more closely applicable to real-world settings given kinetic energy is also a function of speed squared. The optimization problem is then solved using a method called Infinitesimal Perturbation Analysis (IPA). The benefit of this work in comparison to Mitchell et al. [4] is

in the online planning in the continuous state-space. Meng et al.'s work doesn't consider individual agent failures but could be extended to handle them [5].

When planning for failures, Zhou and Tokekar provide a useful taxonomy to approach the different types of algorithms one might consider. They define two separate processes. Firstly, they consider designing coordination algorithms that are resistant to failure, and secondly, they consider designing reconfiguration algorithms to optimally recover after a failure [8]. One algorithm that takes the latter approach utilizes a collaborative mode, modeled as a finite automata to make decisions. Each UGV is sent into this mode in the event of failure, allowing robots to make individual decisions, rather than having an overall leader that decides where to send each UGV [9].

Our algorithm considers a robust planning algorithm with a redundancy parameter to plan for failures to occur, and in so doing make it easier to recover from them, following the first approach. This allows the robot system to still use the initial path even when some number of robots fail. We also consider a recalculation algorithm if there more failed robots than planned, which takes the nearest robots' positions at the point in time of the failure, and redirects them. The recalculation algorithm takes into account similar factors to the planning algorithm, trying to make sure that when the next failure occurs, it will be easy to plan for it. It also balances individual robot goals, such as not running out of fuel.

Zhou et al. provide one method of the first type, where failures are planned to take place. They cover resiliency from failures in multi-robot problems by determining a set of trajectories, then determining the absolute worst-case of robots that could fail for area coverage, above some value alpha. Then they assume that the robots do fail. They select the rest of the robot trajectories greedily to maximize area [6]. Ishat-E-Rabban and Tokekar built on this work and further improved the accuracy and runtime of the initial results by optimizing the trajectory selection method used [7]. As is commonly the case with MILP-based solutions, when they compared their solution to a MILP-based solution, it performed worse on accuracy but better on time. In contrast to our solution, which attempts to surveil an area indefinitely, neither Zhou et al. nor Ishat-E-Rabban and Tokekar consider time to be some large non-infinite value [6] [7]. Additionally, neither test their solutions in a real-world setting.

Cabreira et al. [10] present a solution to the problem of Coverage Path Planning (CPP), and while this is presented in the context of using UAVs, it provides valuable insights into coverage path planning techniques that can be relevant to the MRPCP. The CPP specifically refers to finding optimal paths for minimizing collisions, charge depletion, path length, and a number of other factors.

Both share a fundamental goal of achieving efficient coverage, emphasizing continuous surveillance

over a designated area. They employ path-planning techniques to optimize trajectories, adapting to dynamic environments, and minimizing redundancy. Optimization strategies are essential in both domains to achieve coverage goals with minimal resource consumption. Sensor usage, particularly for data collection, is a common aspect, whether for UAVs surveilling areas or ground-based robots engaged in MRPCP tasks such as pickup and delivery. However, distinctions arise in mobility, as UAVs navigate in three-dimensional space, while the MRPCP typically involves two-dimensional navigation.

A related approach in solving MRPCP problems could include Multi-Agent Path Finding (MAPF) based approaches. Stern provides an overview of MAPF [11]. MAPF primarily addresses the challenge of efficiently navigating multiple agents, each with its own goals, through a shared environment. The agents need to find collision-free paths from their initial positions to their respective goals while avoiding conflicts with each other. The emphasis is on coordinating the movement of agents to prevent collisions and ensure efficient trajectories. MAPF is applicable in various domains such as robotics, video games, traffic management, and logistics. However, it has not been shown that these methods could be applied to MRPCP.

MRPCP, on the other hand, is a broader term that encompasses various path coordination challenges involving multiple robots. While it includes scenarios similar to MAPF, MRPCP can extend to problems where coordination involves not only pathfinding but also task allocation, resource sharing, and synchronization among multiple robots. MAPF specifically deals with finding collision-free paths for agents in a shared environment.

MAPF has direct applications in scenarios where agents need to navigate shared spaces, like warehouses, factories, or urban environments. MRPCP, being a broader framework, could include applications beyond pathfinding, such as collaborative manipulation, exploration, or surveillance tasks.

By formulating our problem as needed by the MILP framework, we will be able to leverage existing solvers and come to optimal solutions. By doing this first, we can compare and test our heuristic solutions against the optimal. The solution will continue to some large non-infinite time t. Our problem will take into account fuel, failure, and obstacles, with implementations in both simulation, done in ARGoS, and in real-world testing.

3 Methodology

In this section, we present the formulation of the problem, the metrics we considered and utilized, the solvers we developed, and the environments we tested the solutions on.

3.1 Problem Formulation

Our problem occurs in a graph, where nodes are placed in a lattice as shown in Figure 1. The graph is fully connected, i.e. every node is allowed to connect to every other node. The solution could easily be extended to real-world by decomposing the area into bins/cells and assigning the cells to each of the nodes.

Field Setup

The field is square and it has a side length of **D**.

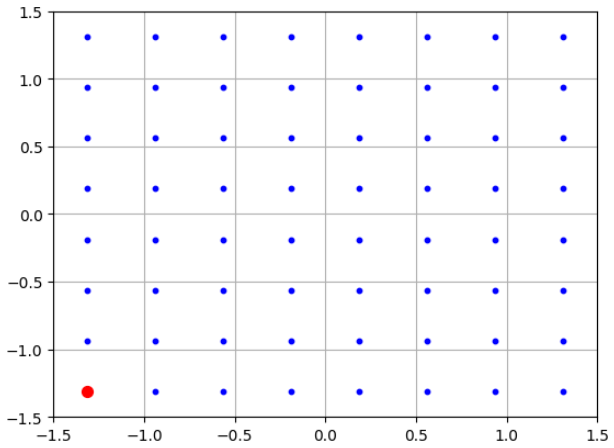

Figure 1: Example map of the field (Field side distance length, $S_{\text{side-length}}$, of 3 meters, and a robot surveillance radius, $R_{\text{surveillance}}$, of 0.265m, resulting in 8 nodes per axis, or 64 nodes in total) Depot node is in red, and all the other target nodes are in blue.

In this project, we focus on a situation with N number of UGV (Unmanned Ground Vehicles) agents. More specifically, we use the Khepera IV robots. Each robot has a surveillance radius of $R_{\text{surveillance}}$, which means that any measurement of an agricultural property is valid within the surveillance radius. Given this, the relationship between the number of nodes a field is divided into is as follows. This gives us the

minimum number of nodes needed to cover all of the square field:

$$\text{Nodes} = \lceil \frac{S_{\text{side-length}}}{\sqrt{2}R_{\text{surveillance}}} \rceil^2 \tag{1}$$

Additionally, enclose the field, the depot, and the starting region region by adding walls. In covering the area, robots are expected to avoid each other and the walls as can be seen in Figure 2.

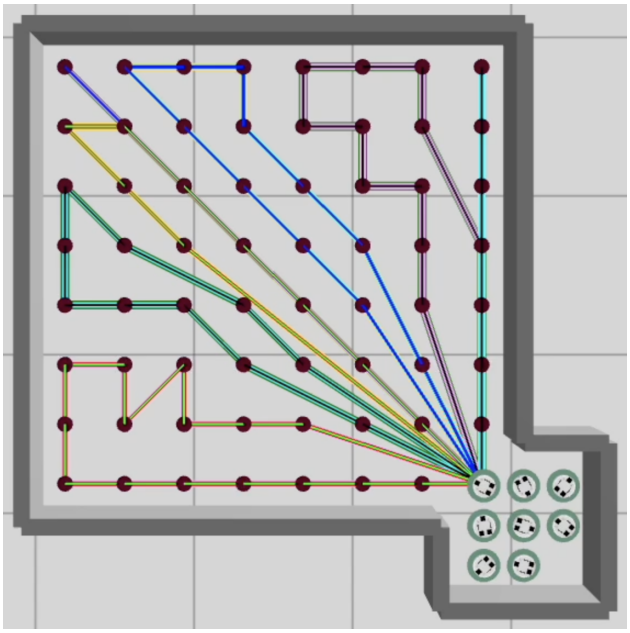

Figure 2: Example field setup (as mentioned in Figure 1) visualized in ARGoS [12]. All the robots start from the lower-right part of the field. Red dots indicate the nodes to visit. Lines indicate the paths assigned to robots.

Differential Drive Kinematics

We use the Khepera IV robots as seen in Figure 3. These robots are differential drive robots where we can set the right and left wheel velocities for moving around and rotating.

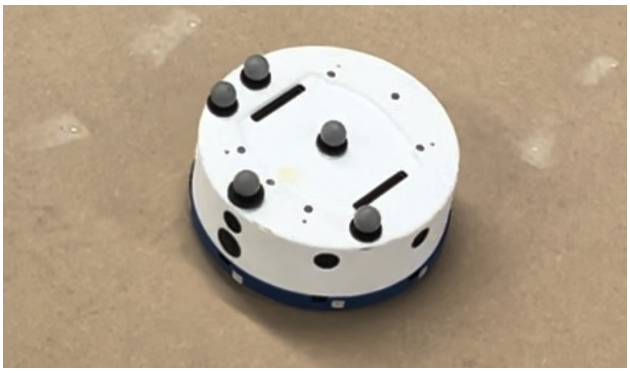

Figure 3: Khepera IV robot in action

Differential drive setups allow for longitudinal translation (forward/backward movement) and in-place rotation, but they do not allow for lateral translation (right/left movement). Hence, we cannot apply any desired velocity vector directly. In the following subsections, we will show the controller design to follow the given velocity vector. Following are a set of equations to mathematically represent the constraints of a differential drive robot as seen in Figure 4.

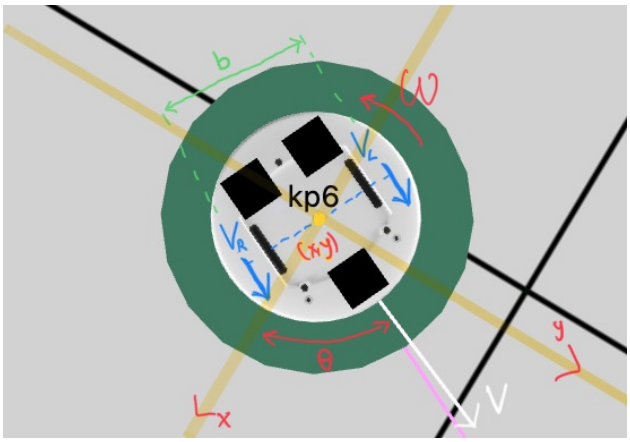

Figure 4: Khepera IV annotated with differential drive variables

$$\dot{x} = V\cos(\theta) \tag{2}$$

$$\dot{y} = V\sin(\theta) \tag{3}$$

$$\dot{\theta} = \omega = \frac{(V_r - V_l)}{b} \tag{4}$$

Additionally, each of the agents are under the following constraints.

- Fuel capacity $L_{\text{max-capacity}}$, which is the maximum distance a robot can travel before needing to refuel at a depot
- Constant probability rate (stochastic) of F_{rate}, which is evaluated at each control step of robots. Our controller operates at a frequency of 10Hz, allowing to evaluate failures in periods of 100ms.

Fuel Capacity

The fuel capacity is expressed in units of distance. Each robot can travel a maximum of $L_{\text{max-capacity}}$ units of distance before it runs out of fuel. Hence, we express fuel capacity in terms of distance/range. Each robot's fuel loss has a linear relationship with the distance travelled, and there is one charging station in one corner of the field for robots to refuel.

Stochastic Failures

Stochastic failures are considered situations where an agent fails with a failure probability rate of F_{rate}. When an agent fails, it is considered to disappear from the map for a period $F_{\text{replacement-time}}$, then reappear in the starting region. If enough robots fail, other robots would need to fill in for the lost functionality of that robot. When all the robots are healthy again, robots switch from recalculated paths to their initial paths.

Objective and Metrics

Given all the setup and constraints, our goal is to maximize the coverage of the field. There are multiple ways to measure coverage. The objective function used by Mitchell et al. [4] maximizes the frequency of visitation of the least frequently visited node/tour. In this case, the objective function is the cost of the maximum costing tour assigned to any robot, and the solvers proposed try to minimize this cost. This is useful in distributing the visitation load between robots and allow the map to be visited more uniformly. Our MILP (which is an extension of the work of Mitchell et al. [4]) and Heuristic 1 solvers uses this same objective. However, one of the problems we faced with this approach is that, when the density of the robots are high enough, there becomes a high traffic region near the depot, and the collision avoidance algorithms we deploy skew the actual visitation frequency distribution.

To better compare solutions from different solvers and environments, we developed another metric called "percent coverage". In this metric, we record the timestamped world positions of each robot, and

map each of the robots to the discrete nodes based on closest distance. Using a look-back period of B, we present the percentage of nodes that were visited within the B period at each time step. Usually, "percent coverage" converges to a scalar value after some time. We use both the coverage over time and the scalar coverage value to compare solutions.

Notional Values

We evaluate the performance of baseline and heuristic algorithms for the given notional values. The original set of notional values was given by our sponsor, BAE Systems. Through our evaluation, we modified some of the specific notional values, without the changing the idea behind them. For example, instead of testing between 1 and 1000 robots, we used between 8 and 1024, as it scaled more cleanly, and allowed more granularity in the intermediary steps. Additionally, because we wanted to compare to our real world environment setup, we modified our minimal set of values, shown below, to be that of our real world setup. Throughout our experimentation, we also added some values, such as B, the lookback period, as it became clear that for the problem to be realistic and meaningful, robots could not appear back at the depot immediately. Surveillance radii are chosen to allow for 100, 400, 2500 nodes in the field. Fuel capacities are chosen to allow the robots to travel 1.5, 3.0, and 4.5 times the fuel capacity needed to have a round-trip travel between the depot node and the furthest point in the field ($L_{\text{min-required-capacity}}$). In the case presented, this would be calculated as follows.

$$L_{\text{min-required-capacity}} = 2\sqrt{2}D \tag{5}$$

Notional Values Used in Point-Mass Simulation (No Failure or Recalculation)

- N = 8, 16, 32, 64, 128, 256, 512, 1024
- $R_{\text{surveillance}}$ = 42.42m, 106.07m, 212.13m
- V = 10m/s, 15m/s, 20m/s
- $S_{\text{side-length}}$ = 3km
- $L_{\text{max-capacity}}$ = 12.72km, 25.44km, 38.16km

Notional Values Used in ARGoS Physics Simulation and Real World [12]

- $N = 8$
- $R_{\text{surveillance}} = 0.265\text{m}$
- $V = 0.05\text{m/s}$
- $S_{\text{side-length}} = 3\text{m}$
- $L_{\text{max-capacity}} = 12.72\text{m}$
- $F_{\text{rate}} = 0.01\%$ chance to fail at every 0.1s
- $F_{\text{replacement-time}} = 300\text{s}$

3.2 Approach

Our solvers generate solutions with built-in resiliency configurable by parameter, r, which makes the solutions such that no re-planning is needed up to the failure of $(r-1)$ failures, but if there are more failures, we run a problem-specific algorithm (Heuristic 2) such that each target is visited at least once by an active robot.

Our environments allow us to evaluate the performance of the solvers.

3.2.1 Overall Diagram

The overarching architecture for the experiments can be seen in Figure 5.

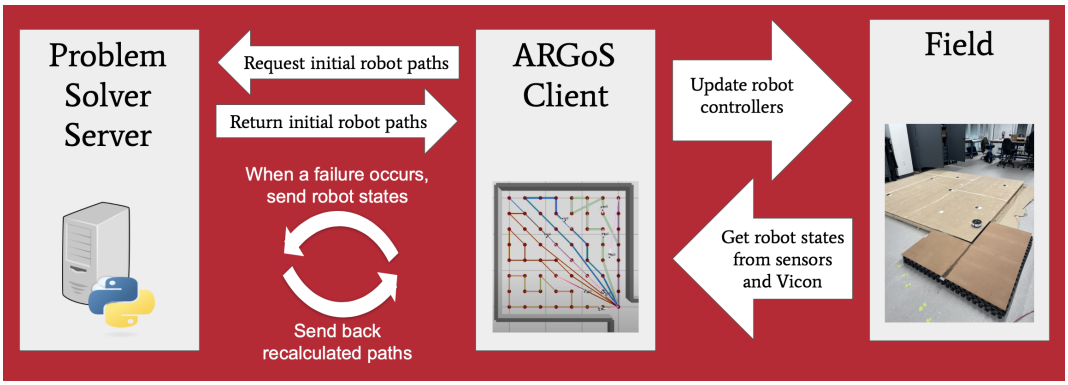

Figure 5: Overall architectural diagram of the setup we used for evaluating the solutions in ARGoS. The problem solver server (left) implements the three solvers: MILP formulation, Heuristic 1, and Heuristic 2/Recalculation. The ARGoS client (center) requests initial paths from the server. Additionally, if there is a need to recalculate paths, more requests are made to the server. Finally, the ARGoS client runs the controllers and commands the robots in the world (sim/real) to run the experiment (right).

3.2.2 Solver: MILP

To solve the problem, we first defined a MILP formulation, and solved it using Gurobi. This formulation is modified from the work of Mitchell et al. [4]. We used this solution as a baseline for developing and testing our heuristics. This section includes all the equations we used and modified for our MILP formulation.

T is the set of all targets, and D is the set of all depots. In our case, we only have only one physical location for refueling, but we need two discrete depots to solve the problem. One depot is used to have the robots start and end (D_s), and the other is used for intermediate refuels (D_r). Since the two depots map to the same physical location (overlaid), in practice, it makes no difference which one is visited. N is the union of the two sets ($N = T \cup D$). K is the set of all robots.

$\boldsymbol{C}$ is a two-dimensional cost matrix, where $\boldsymbol{C}_{ij}$ denotes the distance cost to travel from node i to j. $\boldsymbol{X}$ is a binary three-dimensional edges matrix, where $\boldsymbol{X}^k_{ij} = 1$ if there is a connection from node i to j for robot k, and $\boldsymbol{X}^k_{ij} = 0$ otherwise. $\boldsymbol{P}$ is an intermediate matrix used to enforce the capacity and flow constraints, where $\boldsymbol{P}^k_{ij}$ indicates the remaining number of targets to visit for robot k as it travels from node i to j. Lastly, $\boldsymbol{R}$ is the fuel matrix used to enforce fuel constraints, where $\boldsymbol{R}^k_{ij}$ is the fuel left (in distance units) for the robot k as it travels from node i to j.

The objective function of the MILP formulation is as follows. $P_{\max}$ is a scalar variable and is equal

to the cost of the most expensive path assigned to any robot.

$$J^* = \min P_{\max} \tag{6}$$

where

$$\sum_{i \in N} \sum_{j \in N} \boldsymbol{C}_{ij} \boldsymbol{X}_{ij}^k \leq P_{\max} \qquad \forall k \in K \tag{7}$$

In the rest of the MILP formulation, we show the constraints we used to solve our problem. We used the same grouping method as done in the work of Mitchell et al. [4]. In our formulation, we assume robots can recharge instantly, so we do not include other variables that consider the time it takes to recharge the robots. This is motivated by the relatively short time it takes to swap the battery packs, or pump fuel.

A. Integer Constraints

This constraint makes sure that an edge can either exist or not. It is not possible to have more than one edge between two nodes.

$$\boldsymbol{X}_{ij}^k \in \{0, 1\} \qquad \forall i, j \in N, k \in K \tag{8}$$

B. Degree Constraints

These constraints make sure that robots start and end at the depot, and each target is visited r times. Equations (9) and (10) ensure that there is exactly r incoming and outgoing edges for each target. Equations (11) and (12) ensure that, when there is a need for redundancy ($r > 1$), a robot has at most 1 connection going into and coming out of a target to avoid having two incoming/outgoing edges to/from the same node, which would break the flow. Equations (13) and (14) ensure that, if a robot is going to be used in the solution, it starts and ends at a specific depot. Keeping the start/end depots and refueling depots separate allows MILP to keep track of paths (it always starts and ends at D_s). Equations (13) and (14) also show that it is possible to use less than the specified number of robots in the solution. Equation (15) ensures that, if a target has an incoming edge, it needs to have an outgoing edge too. Finally, Equation (16) ensures

that there are no loopholes (i.e., a target cannot be connected to itself).

$$\sum_{k\in K}\sum_{j\in N} \boldsymbol{X}_{ij}^{k} = r \qquad \forall i \in T \tag{9}$$

$$\sum_{k\in K}\sum_{j\in N} \boldsymbol{X}_{ji}^{k} = r \qquad \forall i \in T \tag{10}$$

$$\sum_{j\in N} \boldsymbol{X}_{ij}^{k} \leq 1 \qquad \forall i \in T, \quad \forall k \in K, \quad \text{if } r > 1 \tag{11}$$

$$\sum_{j\in N} \boldsymbol{X}_{ji}^{k} \leq 1 \qquad \forall i \in T, \quad \forall k \in K, \quad \text{if } r > 1 \tag{12}$$

$$\sum_{j\in N} \boldsymbol{X}_{D_s j}^{k} \leq 1 \qquad \forall k \in K \tag{13}$$

$$\sum_{j\in N} \boldsymbol{X}_{j D_s}^{k} \leq 1 \qquad \forall k \in K \tag{14}$$

$$\sum_{i\in N} \left(\boldsymbol{X}_{ij}^{k} - \boldsymbol{X}_{ji}^{k} \right) = 0 \qquad \forall j \in N, \forall k \in K \tag{15}$$

$$\sum_{i\in N} \boldsymbol{X}_{ii}^{k} = 0 \qquad \forall i \in N, \forall k \in K \tag{16}$$

C. Capacity and Flow Constraints

These constraints ensure that the paths assigned to robots consist of subtours, that are single enclosed loops. $\boldsymbol{P}_{ij}^{k}$ represents the number of remaining nodes to visit. Equation (17) sets the number of nodes to visit, to the number of edges in a robot's subtour. More specifically, it ensures that the difference in "units held" between the start and end node equals to the number of edges in a robot's path. Equation ((18) reduces "units held" by one each time a new target is visited. This mimics the dropping off of that "unit" to that target. Equation (19) ensures that depot refueling trips do not count towards the capacity constraints. Lastly, equation (20) ensures that a robot cannot hold less than 0 units, and cannot hold more units than the number of targets in the field.

$$\sum_{i\in N} \left(\boldsymbol{P}_{D_s i}^{k} - \boldsymbol{P}_{i D_s}^{k} \right) = \sum_{i\in T, j\in N} \boldsymbol{X}_{ij}^{k} \qquad \forall k \in K \tag{17}$$

$$\sum_{j\in N} \left(\boldsymbol{P}_{ji}^{k} - \boldsymbol{P}_{ij}^{k} \right) = \sum_{j\in N} \boldsymbol{X}_{ij}^{k} \qquad \forall i \in T, \quad \forall k \in K \tag{18}$$

$$\sum_{j\in N} \boldsymbol{P}_{j D_r}^{k} - \boldsymbol{P}_{D_r j}^{k} = 0 \qquad \forall k \in K \tag{19}$$

$$0 \leq \boldsymbol{P}_{ij}^{k} \leq |T| \boldsymbol{X}_{ij}^{k} \qquad \forall i, j \in N, k \in K \tag{20}$$

D. Fuel Constraints

These constraints ensure that the paths generated satisfy all the fuel constraints (i.e., a robot is always scheduled to refuel at the depot before it is out of fuel). First, we set M to the sum of fuel capacity and the maximum distance between the depot and targets. In equation (5), we calculate the minimum fuel needed to solve this problem. The maximum distance between the depot and targets is, then half of $L_{\text{min-required-capacity}}$. Equations (22) and (23) ensure that, the fuel lost between nodes i and j equal to the cost of going from node i to j. Equations (24) and (25) ensure that, the fuel capacity after leaving a depot on its way to target i equals the fuel capacity minus the cost to travel from the depot to target i. Equation (26) ensures that a robot always has enough fuel to go to the depot. Lastly, Equation (27) ensures that a robot fuel cannot exceed the fuel capacity at any time.

$$M = L + \frac{L_{\text{min-required-capacity}}}{2} \tag{21}$$

$$\boldsymbol{R}_{ji}^{k} - \boldsymbol{R}_{ij}^{k} + \boldsymbol{C}_{ij} \leq M\left(1 - \boldsymbol{X}_{ij}^{k}\right) \qquad \forall i, j \in T, k \in K \tag{22}$$

$$\boldsymbol{R}_{ji}^{k} - \boldsymbol{R}_{ij}^{k} + \boldsymbol{C}_{ij} \geq -M\left(1 - \boldsymbol{X}_{ij}^{k}\right) \qquad \forall i, j \in T, k \in K \tag{23}$$

$$\boldsymbol{R}_{ji}^{k} - L + \boldsymbol{C}_{ij} \geq -M\left(1 - \boldsymbol{X}_{ij}^{k}\right) \qquad \forall i \in D, \quad \forall j \in T, k \in K \tag{24}$$

$$\boldsymbol{R}_{ji}^{k} - L + \boldsymbol{C}_{ij} \leq M\left(1 - \boldsymbol{X}_{ij}^{k}\right) \qquad \forall i \in D, \quad \forall j \in T, k \in K \tag{25}$$

$$\boldsymbol{R}_{ji}^{k} - \boldsymbol{C}_{ji} \geq -M\left(1 - \boldsymbol{X}_{ji}^{k}\right) \qquad \forall i \in D, \quad \forall j \in T, k \in K \tag{26}$$

$$0 \leq \boldsymbol{R}_{ij}^{k} \leq L \qquad \forall i, j \in N, k \in K \tag{27}$$

Post-Processing With a TSP Solver (k-opt)

The objective of our MILP formulation is to minimize the cost of the maximum costing path assigned to a robot. This is increases the frequency in which each node is visited in the discrete world. In our implementation, we save and visualize every better intermediate solution. We observed that the MILP formulation was not able to optimize paths that are not maximum costing. To see an example of this, consider the solution in Figure 6. In the solution, the maximum costing tour is robot 5's path. As can be seen, this and paths of other robots could be improved by a TSP solver without violating any other constraints. When a TSP solver is run for each of the subtours of a given robot, the sum of costs metric could also be minimized. In Figure 7, the k-opt algorithm [13] is used to improve the paths. The paths of robots 2, 4, and 5 are improved, reducing both the cost of the maximum costing tour and the sum of costs

of all paths. Using this approach, we can get slight improvements for each robot path at virtually no cost (in comparison to the time it takes for MILP to find better solutions).

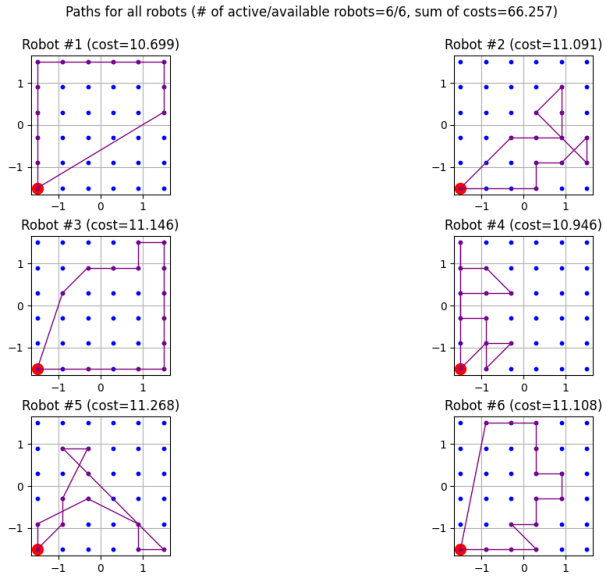

Figure 6: An example intermediate solution found by MILP. ($N = 6, |\text{Nodes}| = 36, r = 2, S_{\text{side-length}} = 3, L_{\text{max-capacity}} = 8.485$)

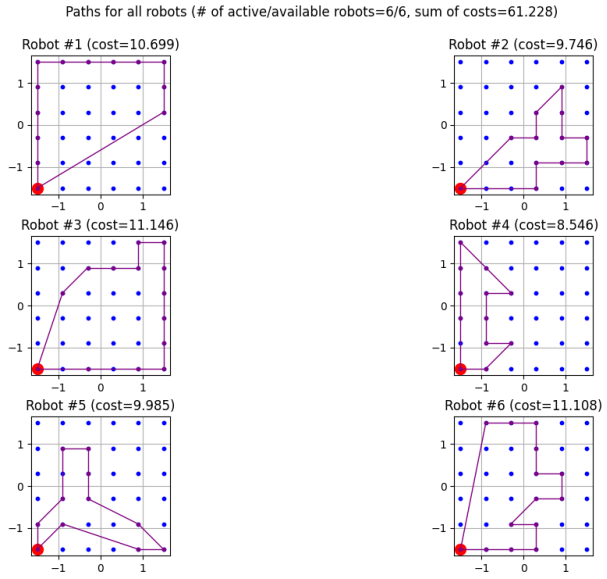

Figure 7: The TSP-optimized version of the intermediate solution in Figure 6

Another benefit of this post-processing trick is that it can be run in parallel, utilizing the multiple cores/CPUs in the solver host. This is run each time MILP finds a better solution. This trick usually helps

to get better solutions in the intermediate solutions early on. Usually, as MILP finds better solutions, this step becomes redundant.

Post-Processing to Account for Failures

Prior to implementing the recalculation heuristic (Heuristic 2), we also used to rely on this post-processing trick to rewire paths when robots fail. While this did not make it to the final code we demonstrate, we thought it would be more complete if we share some of our early attempts at dealing with failures.

In the event of a failure in the online case, we used to rewire the edges such that each target in the map is visited at least once. A robot was assumed to fail at some random position during its path. When that happened, the remaining nodes on its path were added to the other robots' paths based on distance. When the nodes could not be added to the other robots' paths without going over the fuel capacity, paths from the depot, to the target node, and back to the depot were added.

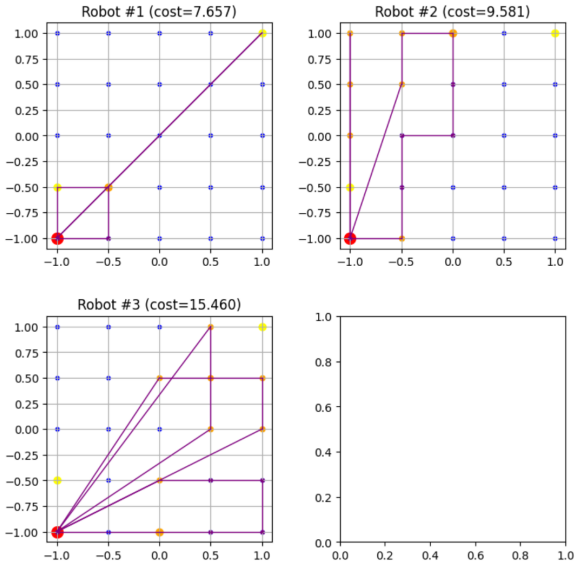

Figure 8: An example intermediate MILP solution

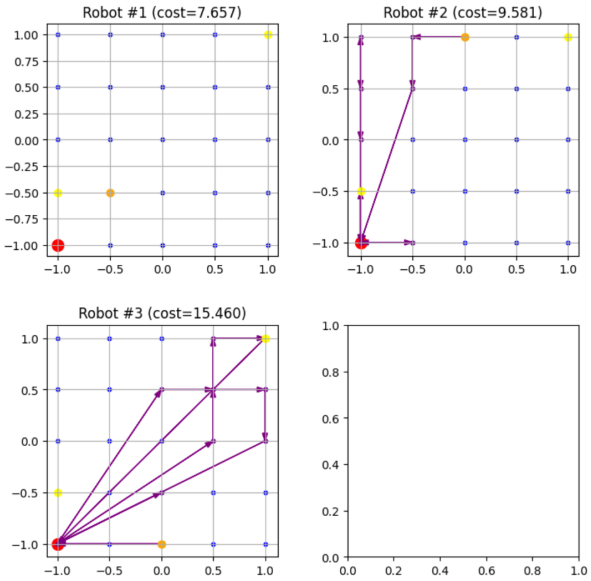

Figure 9: An example intermediate MILP solution with this post-processing trick, after robot 1 has failed

3.2.3 Solver: Heuristic 1

The computational complexity of the MILP formulation to find "good enough" paths pushed us to devise heuristic algorithms that had no optimality guarantees, but could generate paths much faster. In total, we developed two heuristic algorithms.

The pseudo-code can be found in Heuristic 1. This algorithm has objective similar to MILP, where it minimizes the cost of the maximum costing path. This algorithm can scale up to 20,000 nodes and 10,000 robots. If the TSP optimization (step 3) is turned off, it solves most of the problems in a matter of seconds.

This algorithm consists of five steps. First, in step one, the algorithm sorts the nodes based on angle to the depot, and where the nodes are co-linear, sorts based on distance (first step as depicted in Figure 10). Next, in step two, the splitting algorithm bounces between fuel capacities $L_{\text{min-required-capacity}}$ and $L_{\text{max-capacity}}$ to split the nodes into at most N groups using the cheapest insertion heuristic for TSP. This simple heuristic sequentially gets a new node, and places it between two other nodes where the tour cost increase is minimum. The splitting algorithm works by trying out different maximum fuel capacities to divide the nodes into groups. If creating N groups conflicts with the fuel capacity constraints, N is incremented by one, until fuel capacity is no longer violating the constraint. This gives the fuel constraint satisfying "novel" subtours, which, if each is executed once, all the nodes would be visited exactly once (second step as depicted in Figure 11). Given the groups of nodes, a TSP solver can be optionally run to further optimize the subtours from step two (step three as depicted in Figure 12). This step (step three) usually lowers the

cost of the paths by up to 15%. So, if there are 50,000+ nodes, it might save compute time and memory resources to skip this step. Next, the optimized subtours are sorted and at least r copies are made (step four as depicted in Figure 13). In cases where there are a lot more robots, than the subtours created, r increased so that every robot is assigned at least one subtour. When all these subtours are assigned, each node is visited at least r times. Finally, in step five, the same splitting algorithm is used once again to bounce between the maximum costing subtour, and the sum of subtours to split the subtours into at most N groups. In this case, the splitting algorithm tries bouncing maximum costs for each robot's path. It stops where the maximum cost of the paths assigned can not be further minimized. An example of how these paths are distributed can be seen in Figure 14.

Heuristic 1

Input: N, R, D, L, r

Output: RobotPaths: Collection of paths assigned to each robot

$Nodes, NodeIndices, TargetIndices, DepotIndices \leftarrow ConstructMap(S_{\text{side-length}}, R_{\text{surveillance}})$

`/* Step One: Sort targets based on angle and distance to depot */`

$SortedTargetIndices \leftarrow SortTargetIndices(TargetIndices, DepotIndices)$

`/* Step Two: Group nodes using the cheapest insertion heuristic */`

$PreTSPSubtours \leftarrow divideTargetsByP(SortedTargetIndices, P = N, low = L_{\text{min-required-capacity}}, high = L_{\text{max-capacity}})$

`/* Step Three: (Optional) Run the k-opt algorithm to improve subtours from step 2 */`

for $i \leftarrow 0$ *to* $|PreTSPSubtours|$ **do**

| $NewCPUCore \leftarrow TSPSubtours[i] \leftarrow kopt(PreTSPSubtours[i])$

end

$WaitUntilTSPComplete()$

`/* Step Four: Sort subtours based on cost, then make r copies */`

$TSPSubtours = SortSubtours(TSPSubtours)$

$Subtours = MakeNCopies(TSPSubtours, N = r)$

`/* Step Five: Distribute subtours between the robots */`

$RobotPaths \leftarrow divideTargetsByP(Subtours, P = N, low = max(Subtours), high = sum(Subtours))$

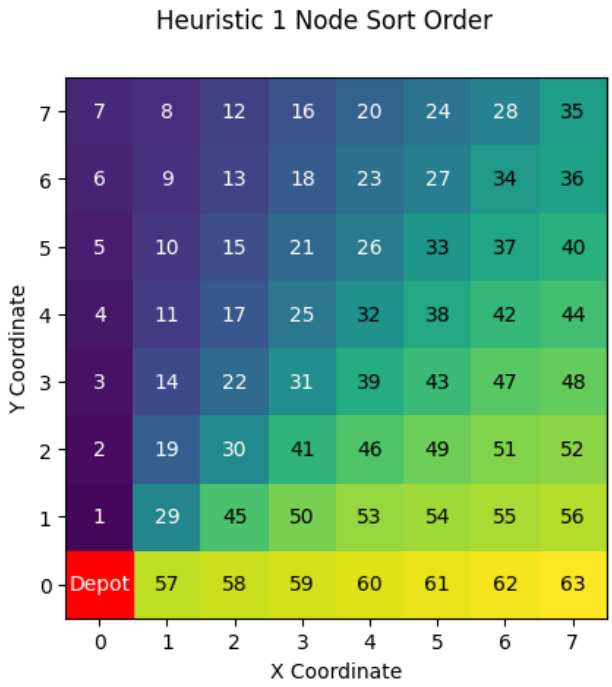

Figure 10: Heuristic 1 Step 1: Sort the nodes based on direction and distance from depot

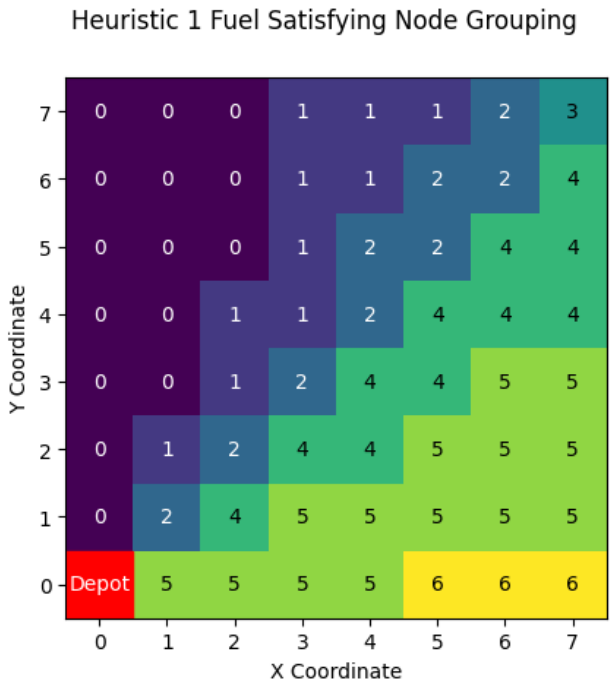

Figure 11: Heuristic 1 Step 2: Group the nodes based on fuel capacity

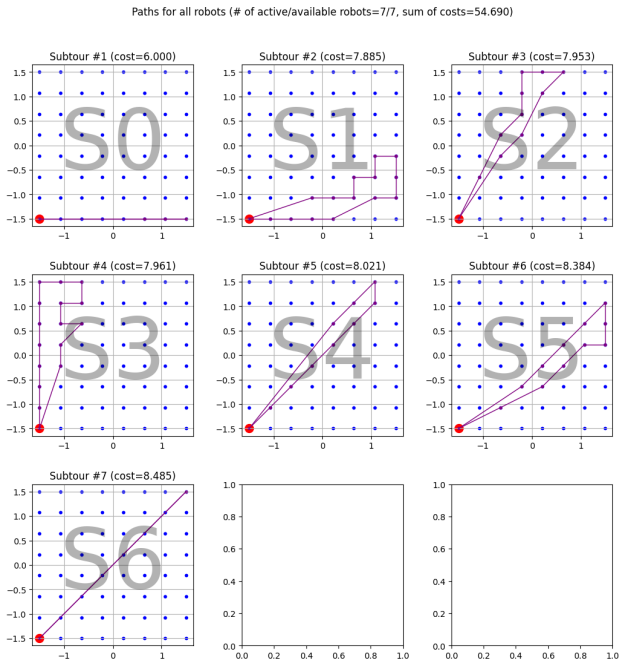

Figure 12: Heuristic 1 Step 3 (Optional): Run the TSP solver on the nodes in the groups, and create "novel" subtours

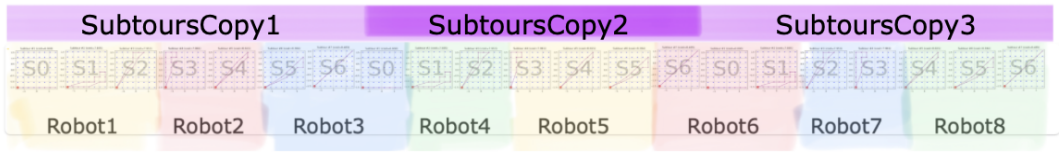

Figure 13: Heuristic 1 Step 4: Create r copies of each of the "novel" subtours.

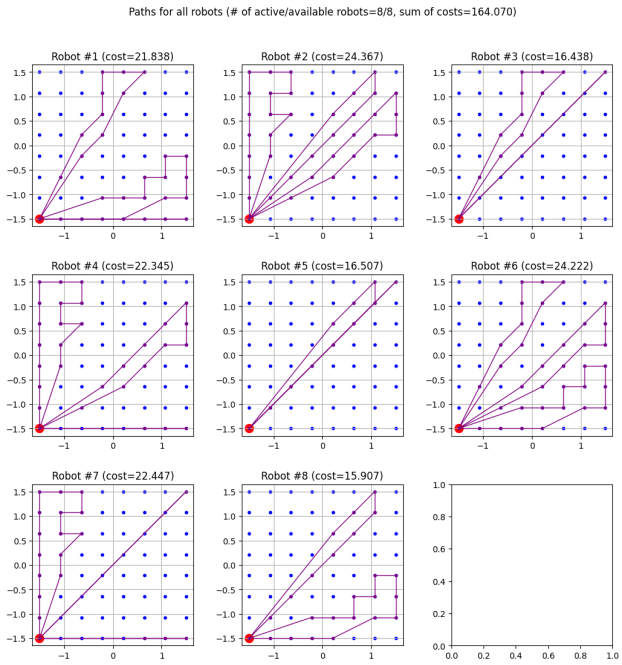

Figure 14: Heuristic 1 Step 5: Assign to robots based while minimizing the maximum costing path of each robot.

3.2.4 Solver: Heuristic 2

Heuristic 2 has two goals in our setup. One, it is focused around being resilient to failures. Instead of building a set of paths that run until enough robots fail that they must be recalculated, it instead sends robots paths each time they come to the depot. Second, it serves as a recalculation algorithm for MILP and Heuristic 1, when they must be recalculated because of robot failures.

Heuristic 2

while more nodes to be covered **do**

  for ki in num_of_robots **do**

    while goal in nodes already covered AND fuel for return journey **do**

      $max\_dist \leftarrow 0$

      $goal \leftarrow (0, 0)$

      for n in uncovered nodes **do**

        if distance((0,0), n) ¿ max_dist **then**

          $max\_dist \leftarrow distance((0, 0), n)$

          $goal \leftarrow n$

        end

      end

    end

    $path, distance\_travelled \leftarrow Modified\_A^*(last\_node(ki), goal)$

    $robot\_paths(ki) \leftarrow robot\_paths(ki) + path$

    UpdateNodesCovered()

    $robot\_fuel(ki) \leftarrow robot\_fuel(ki) - distance\_travelled$

    $last\_node(ki) \leftarrow robot\_paths(ki)(-1)$

    if (0,0) == last_node(ki) **then**

      $robot\_fuel(ki) \leftarrow$ FUEL_CAPACITY

    end

  end

end

3.2.5 Environment: Point-Mass Simulation

We have three environments to test our solvers. The first one is a simple point-mass simulator where we assume no collisions. Every robot perfectly follows the assigned trajectory at a velocity V. Given a time step dt, this simulation advances each robot by a distance $ds = V \cdot dt$ in its path. An example of the sampled world points can be seen in Figure 15. The experiments stops after t units of time (second).

The main advantage of this approach is in its speed. We can very quickly sample points, and used them to calculate metrics such as percent-coverage. However, this simulation fails to mimic a real physics simulator when there is a high traffic region in the map.

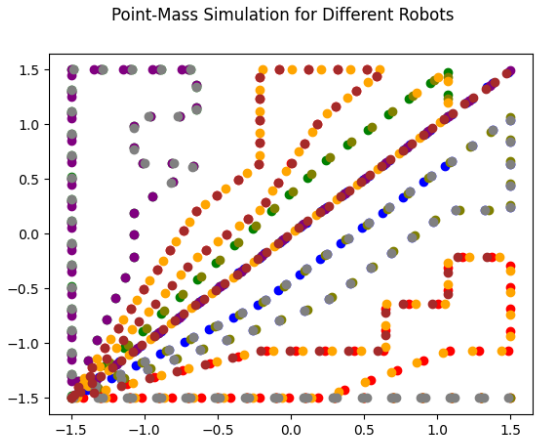

Figure 15: Example sampled points in the point-mass simulation. Each different color represents a different robot. ($V = 0.2m/s$, $dt = 1s$, $t = 100s$)

3.2.6 Environment: ARGoS Physics Simulation

ARGoS is a multi-agent simulation. In contrast to Gazebo, a simulator commonly used for simulation of robots, it allows large numbers of robots to be tested. This functionality is important for our testing because our goal was to create algorithms that could work with large numbers (> 100) of robots. Additionally, it allows for quick switching between simulation and real world testing. In our case, we use the *dyn2d* engine for simulating the Khepera IVs, and the Vicon tracker engine when we test in the lab.

In these environments, we need to execute trajectories using a controller, handle collisions by using ORCA and employing deadlock resolution algorithms, and introduce failures for testing the recalculation pipeline.

Differential Drive Controller Design

Given the trivial nature of the differential drive systems, the controller design is a very simple one. The controller takes in desired goal and velocity commands from the ORCA algorithm. Khepera IV robots allow for their wheel velocities to be set directly. Hence, manipulating the set of equations (2), (3), and (4), we get the following relationship:

$$V_l = V_{eff} - \frac{b}{2}\omega_{eff} \tag{28}$$

$$V_r = V_{eff} + \frac{b}{2}\omega_{eff} \tag{29}$$

Equations (28) and (29) allow us to control the robot velocity magnitude and direction through the V_{eff} variable, and control the heading through the ω_{eff} variable. We devise the following PID loop to have the robot go in the direction of the velocity vector. $Dist_{err}$ is the distance left to reaching the goal node. $Angle_{err}$ is the difference between the current robot heading, and the ORCA preferred heading (V_{ORCA}). We simply tuned the PID controllers in simulation, and verified with the real robots. The PID controllers for linear velocity and angular velocity are separate.

$$V_{eff} = PID.computeEffort(Dist_{err}) \tag{30}$$

$$\omega_{\text{eff}} = PID.computeEffort(Angle_{err}) \tag{31}$$

where

$$Dist_{\text{err}} = \sqrt{(goal.x - curr\_pos.x)^2 + (goal.y - curr\_pos.y)^2} \tag{32}$$

$$Angle_{\text{err}} = atan2(V_{ORCA}) \tag{33}$$

ORCA

ORCA is a collision avoidance algorithm we use to have robots progress towards their goal positions without colliding with each other. In essence, it relies on the concept of velocity obstacles, and time horizons to check for future collisions, and course adjustments. This collision avoidance algorithm is able to respect both static (walls) and moving (robots) obstacles. We load each of the walls placed in ARGoS as static obstacles in ORCA. To supply ORCA with the velocity vectors of neighboring agents, we have each robot broadcast its own velocity to its neighbors, and listen other agents. We do not implement ORCA from scratch, instead, we use the official C++ implementation from the RVO2 library [14]. In Figures 16 and 17, a simple demo of ORCA is presented.

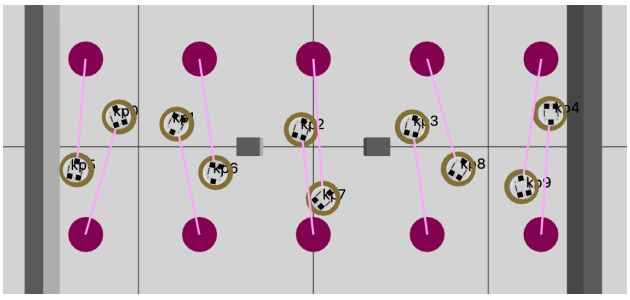

Figure 16: ORCA algorithm in action. Red dots are the goal positions, and the pink lines connect the robots to their goal positions. White line is the ORCA preferred velocity vector over time horizon.

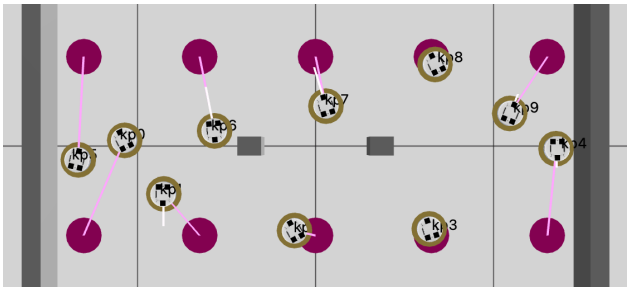

Figure 17: ORCA algorithm in action (some time after Figure 16). Red dots are the goal positions, and the pink lines connect the robots to their goal positions. White line is the ORCA preferred velocity vector over time horizon.

Deadlock Resolution

ORCA only allows to avoid collisions, but in high traffic scenarios, ORCA might get robots in a deadlock. There are couple tricks we employ to avoid this. First, we allow the distance threshold for visiting a node dynamic, increasing based on the number of robots in that region. This allows robots to not be blocked in visiting nodes it other neighboring robots are blocking the path. If there are no nearby robots, however, robot has a low distance threshold for visiting the node. Second, after getting the ORCA preferred velocity vector, we add a small random perturbation of around 5%. This helps break symmetries in the ORCA vectors for robots to go around each other. Third and last case is the most advanced measure we had to take. In this measure, robots constantly monitor their progress towards their goal position. If not enough progress has been made in a given time period, a robot puts itself on a "deadlock" mode. In this mode, the ego robot gets the centroid of its neighbors, and goes in the opposite direction for a period of time. This allows the outermost robots to back off, allowing the inner robots to move and resolve conflicts. After some period of time, robots go back to ORCA mode. In Figures 18, 19, 20, and 21, we present four stages of deadlock resolution.

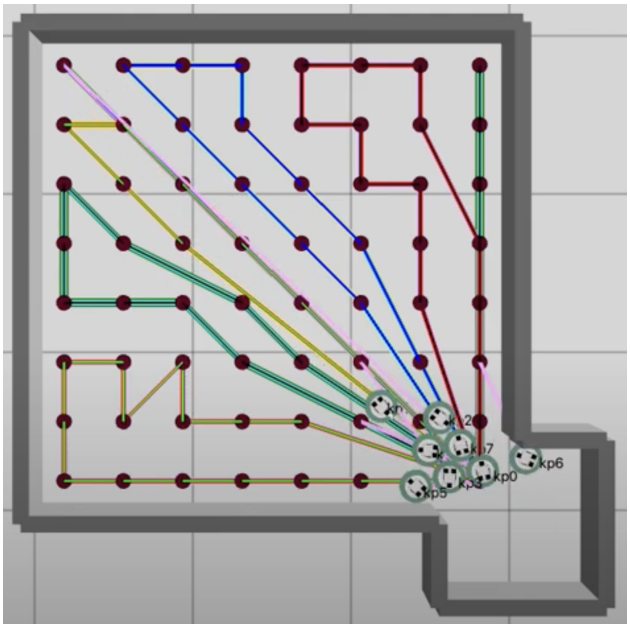

Figure 18: Deadlock resolution demo at $t = 2188.8s$. Robots are about to get into a deadlock situation.

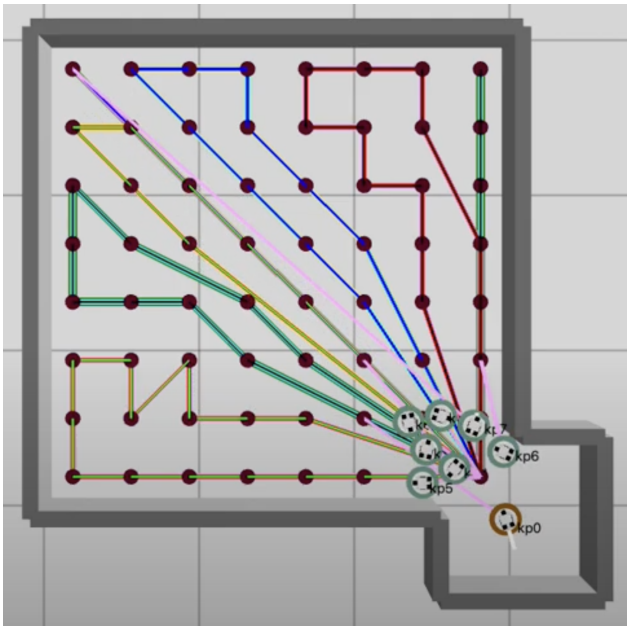

Figure 19: Deadlock resolution demo at $t = 2195.3s$. First robot backs off after detecting a deadlock. Deadlock mode is signified by the brown circle around the robot.

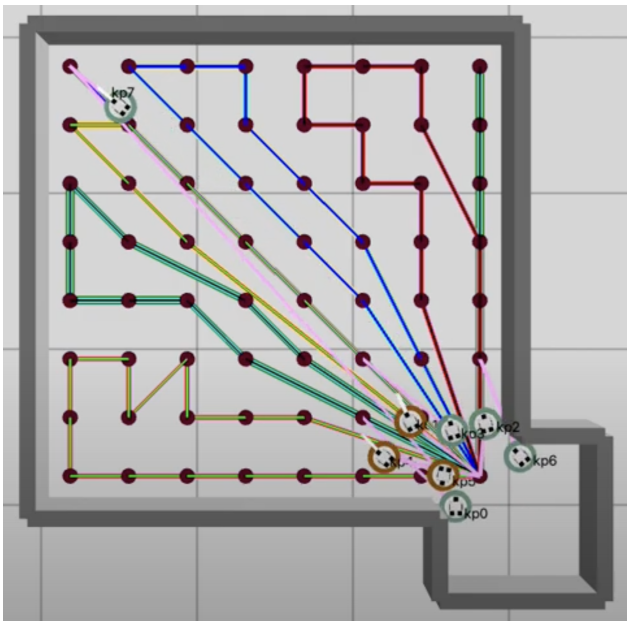

Figure 20: Deadlock resolution demo at $t = 2213.4s$. One robot was able to escape already. More robots back off after detecting a deadlock. Deadlock mode is signified by the brown circle around the robot.

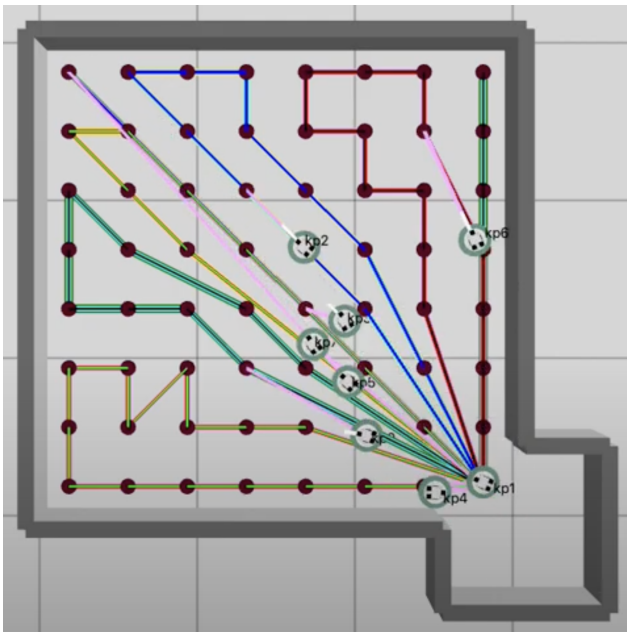

Figure 21: Deadlock resolution demo at $t = 2230.9s$. Deadlock is solved.

Robot failure and replacements

When a robot fails, we return it to the starting region and activate it after a set period of time passes. This way, robots that have failed 1) take time to be replaced, as it is realistic, and 2) do not block

other robots from moving, creating more complex deadlocking and blocking scenarios, to keep everything simple enough to be feasible.

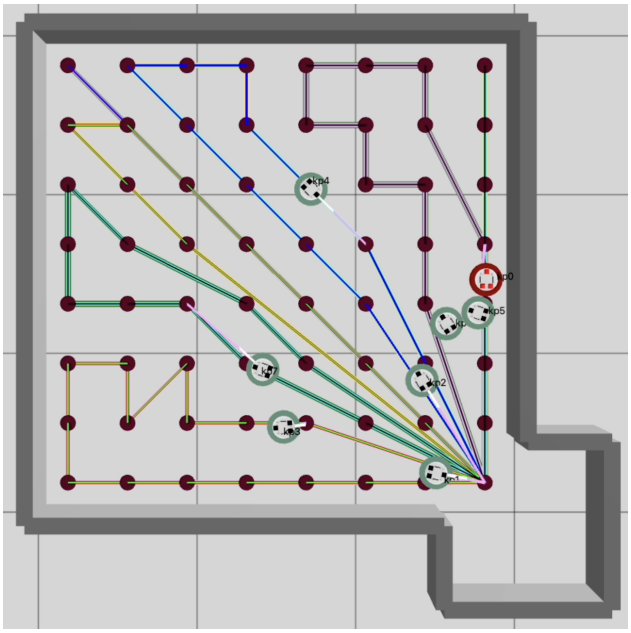

Figure 22: ARGoS visualization immediately after a robot fails.

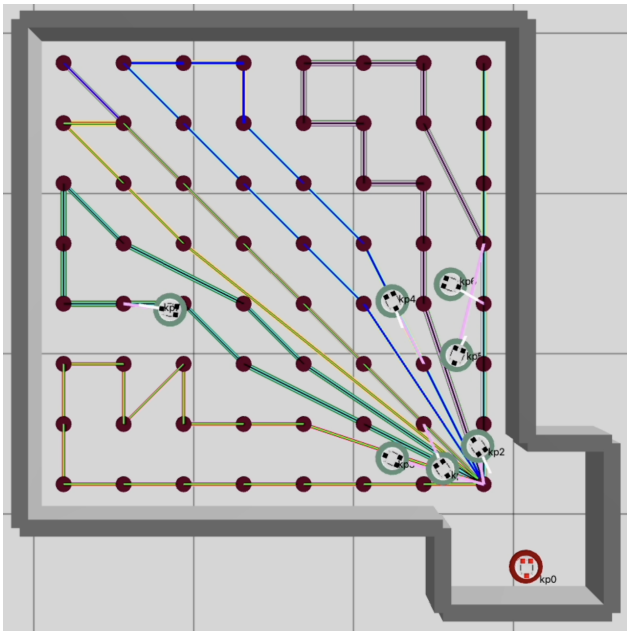

Figure 23: ARGoS visualization after the failed robot is relocated to the starting region.

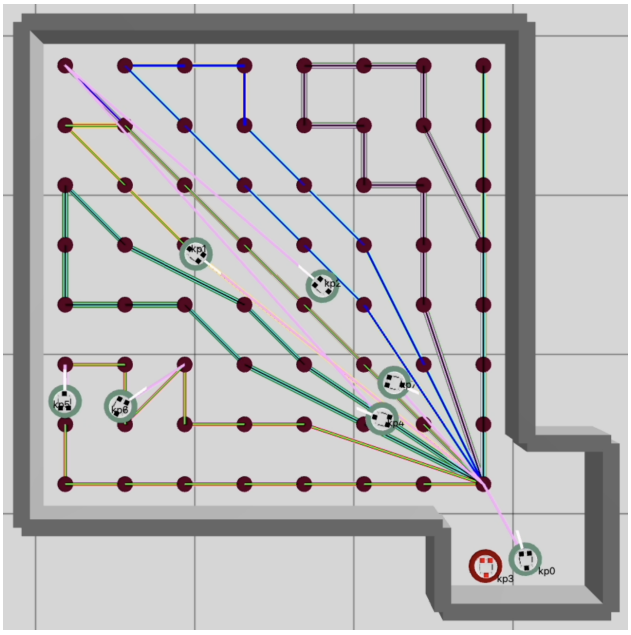

Figure 24: ARGoS visualization after the failed robot is back healthy. In the meantime, another robot has failed.

3.2.7 Environment: ARGoS Vicon Setup

We used the Vicon setup in the lab to get the robot positions. As can be seen in Figure 3, each of the Khepera IV robots have markers which allow for positioning using the cameras overlooking the field. Each robot is calibrated in the same coordinate frame as the ARGoS simulator to allow for easy switching between simulation and real world testing. In the final state of our code, we simply created a different .argos configuration file to run the experiments. All the code between ARGoS simulation and real world stays the same. Only changes are that the physics engine is replaced by the Vicon tracker, and the robot velocity commands are sent to the robots over the network.

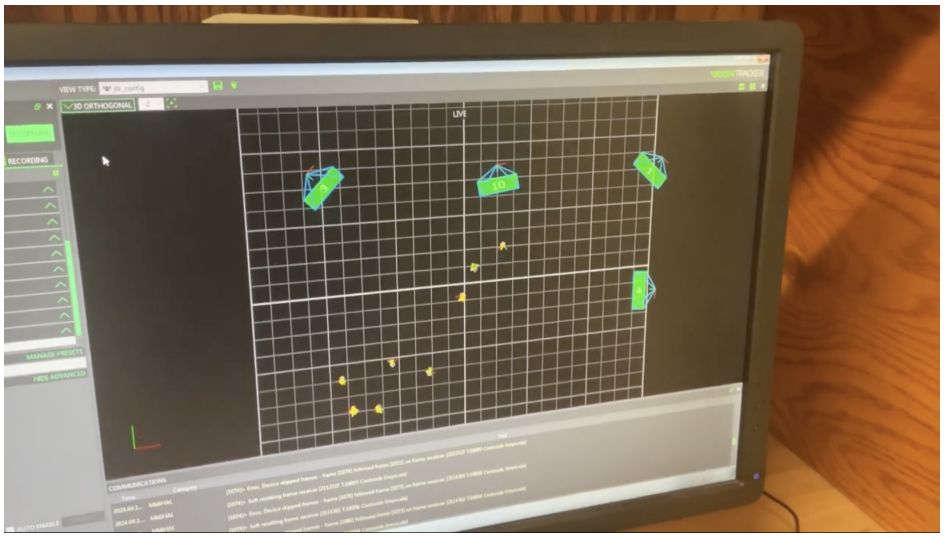

Figure 25: Vicon screen

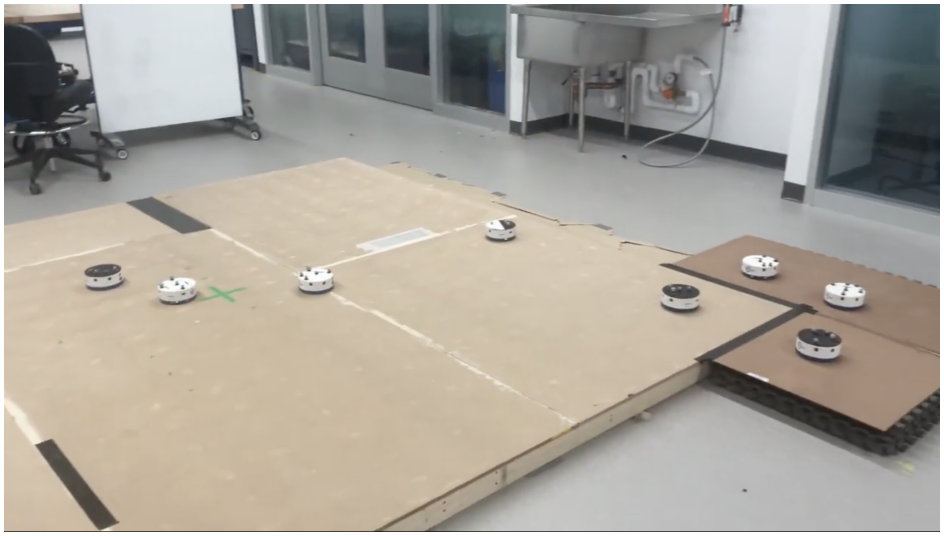

Figure 26: Lab field

4 Results

Here, we present our findings on the three solvers are run using the three environments as described in section 4. First, we compare solvers based on runtime. Second, we compare the three solvers using the same problem parameters and the ARGoS physics simulation. Then, we use the same problem parameters to compare the three environments we have. Lastly, we use the point-mass simulation to test the different problem parameters in solvers.

4.1 Comparison of Runtime for Different Solvers

Here we provide two examples of runtime characteristics of the proposed solvers. All the reported runtimes are executed on an Apple MacBook Pro with M3 Max chip (14-core variant). These tables also show that Heuristic 1 is constrained not as much on the computation, but on the memory. In Table 4, the memory usage went as high as 36GB. Upon inspection, we realized that the cost matrix alone accounted for around 29GB of memory consumption. The function used (`scipy.spatial.distance.cdist`) to calculate the cost between nodes does all of its computation in 64-bit float. This creates a matrix of shape $62,500 \times 62,500$, where each element is a 64-bit float. If this resolution can be reduced to 32-bit or 16-bit, Heuristic 1 can easily solve problems with 100,000+ nodes.

Solver	Runtime
MILP	*60 Hours
Heuristic 1 (With Step 3/TSP Optimization On)	1.318 Seconds
Heuristic 1 (With Step 3/TSP Optimization Skipped)	0.00151 Seconds
Heuristic 2	0.0225 Seconds

Table 1: Table showing the runtimes of each solver for the following problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$. *MILP formulation was early stopped after reaching an optimality gap of less than 8%.

Solver	Runtime
Heuristic 1 (With Step 3/TSP Optimization On)	5.367 Seconds
Heuristic 1 (With Step 3/TSP Optimization Skipped)	0.288 Seconds
Heuristic 2	1055.630 Seconds

Table 2: Table showing the runtimes of each solver for the following problem parameters: $N = 1000$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 42.4m$ (Equivalent to 2,500 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 100$. MILP is not included as it takes a long time to setup the problem before starting to solve.

Solver	Runtime
Heuristic 1 (With Step 3/TSP Optimization On)	117.557 Seconds
Heuristic 1 (With Step 3/TSP Optimization Skipped)	1.478 Seconds

Table 3: Table showing the runtimes of each solver for the following problem parameters: $N = 1000$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 21.2m$ (Equivalent to 10,000 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 100$. MILP is not included as it takes a long time to setup the problem before starting to solve. Likewise, Heuristic 2 was not able to complete this in an hour.

Solver	Runtime
Heuristic 1 (With Step 3/TSP Optimization Skipped)	72.195 Seconds

Table 4: Table showing the limits of Heuristic 1 solver for the following problem parameters: $N = 1000$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 8.49m$ (Equivalent to 62,500 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 100$.

4.2 Comparison of Paths for Different Solvers

The following paths (see Figures 27, 28, and 29) are the initial robot paths for MILP, Heuristic 1 (H1), and Heuristic 2 (H2) using established methodologies tailored to each approach's optimization principles. Following the initial path generation phase, recalculation mechanisms (see Figure 30) brought a novel dimension to our analysis. With online recalculation, robots can regenerate their paths based on what robots are on the field and what their current state is. This dynamic recalculation allowed for the creation of new paths that could adapt to changing circumstances, thereby enhancing the resilience and responsiveness of the overall system. By integrating recalculation into our solvers, we not only explored the performance of initial path-generation methods but also assessed the impact of system robustness to failures.

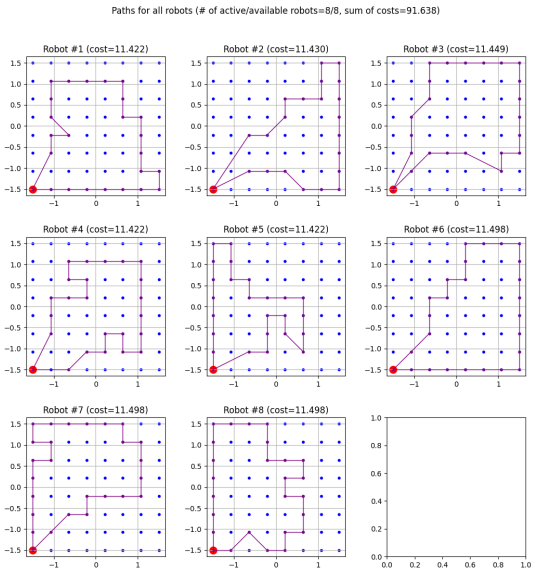

Figure 27: Robot Paths for MILP. The formulation is stopped after 60 hours. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$. The solution presented here is guaranteed to have an optimality gap of less than 8%.

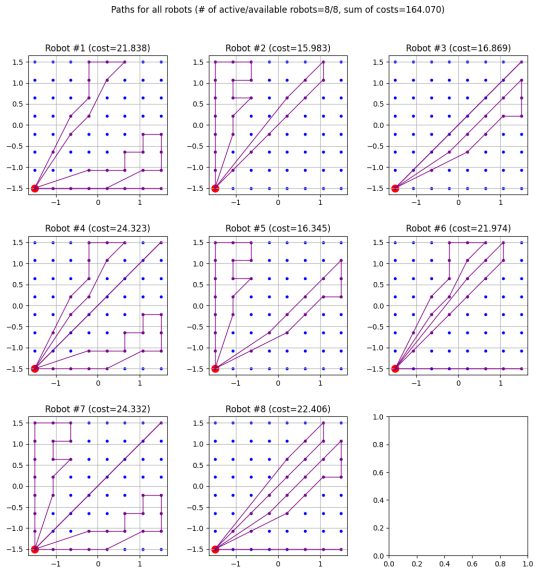

Figure 28: Robot Paths for Heuristic 1. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

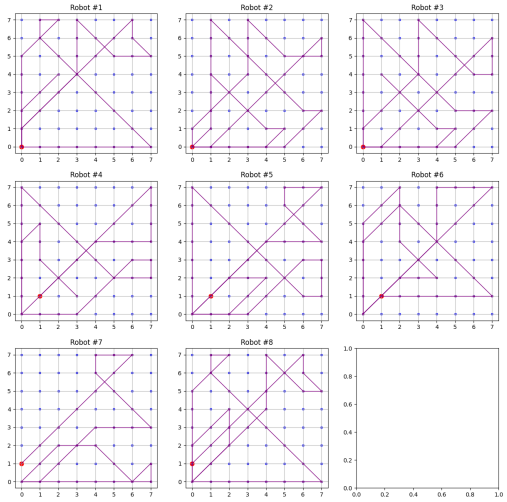

Figure 29: Robot Paths for Heuristic 2. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

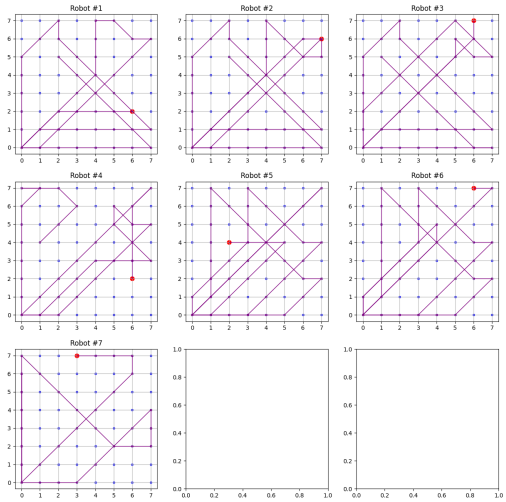

Figure 30: Robot Paths for Recalculation. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

4.3 Comparison of Visitation Frequency for Different Solvers

The visitation frequency heatmap representations shown below offer a normalized view of the paths traversed by the robots. The heat map visualizations were created using point-mass data and ARGoS simulation data, where yellow represents a higher visitation and purple represents a lower visitation. The heat map allows us to easily discern how the robots' movement patterns change with different solvers.

4.3.1 Point-Mass Visualizations

In comparing the visualization of the MILP simulation (Figure 31) against heuristic methods (Figure 32 and Figure 33), a notable aspect lies in the uniformity of the representation. The MILP approach shows balance in the paths, however, the heuristic methods exhibit biases toward traveling diagonally to the furthest area. The MILP+Recalculation visualization in Figure 34 shows a similar pattern to the heuristic methods because the recalculation uses Heuristic 2 on MILP-generated paths.

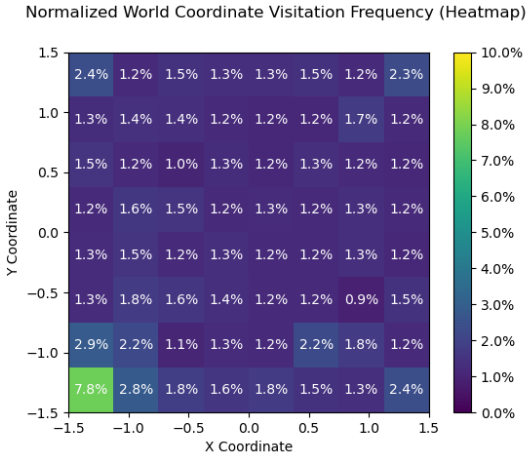

Figure 31: Visitation Frequency Heat Map for MILP. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

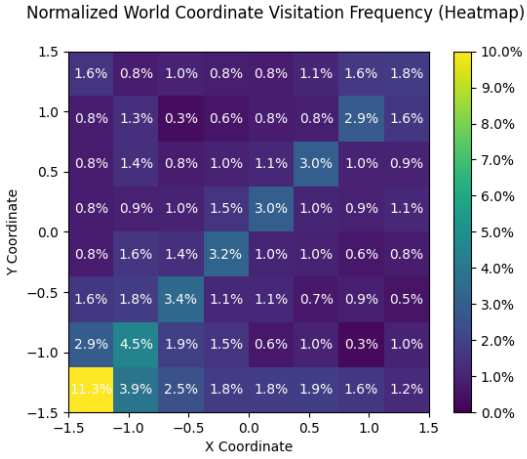

Figure 32: Visitation Frequency Heat Map for Heuristic 1. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

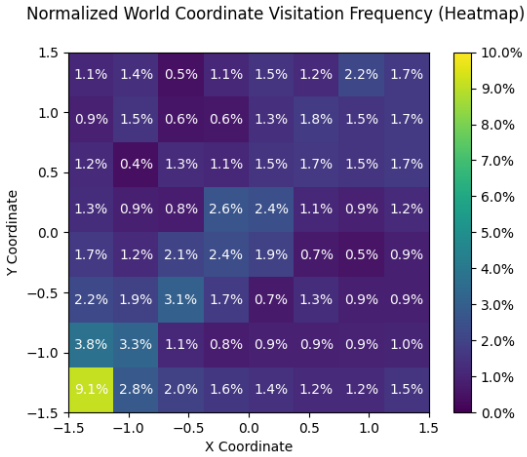

Figure 33: Visitation Frequency Heat Map for Heuristic 2. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

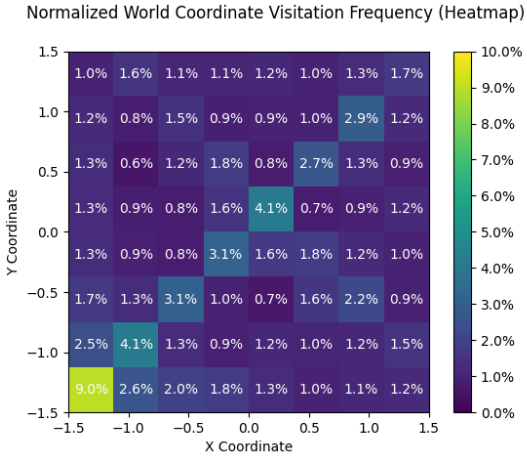

Figure 34: Visitation Frequency Heat Map for Recalculation. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

4.3.2 ARGoS Visualizations

In contrast to the point-mass heat maps, ARGoS provides a different perspective, characterized by its physics-based approach. While MILP prioritizes uniformity in the point-mass simulation, it can be seen in Figure 35, that there is less uniform behavior due to the dynamic interactions in the robots' paths. The following heuristic heat maps (see Figures 36, 37, 38) generate similar visualizations, displaying the higher visitation being across the diagonal axis towards the furthest node.

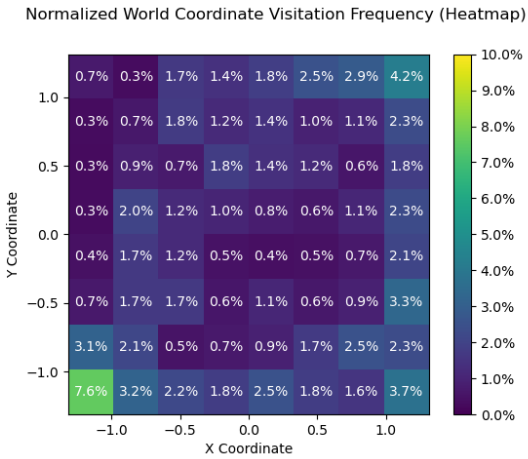

Figure 35: Visitation frequency heat map for MILP. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

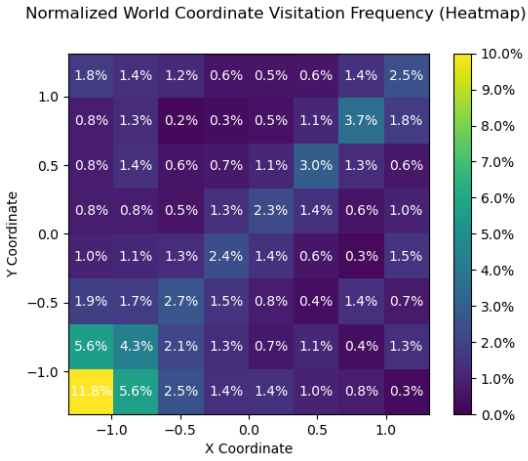

Figure 36: Visitation frequency heat map for Heuristic 1. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

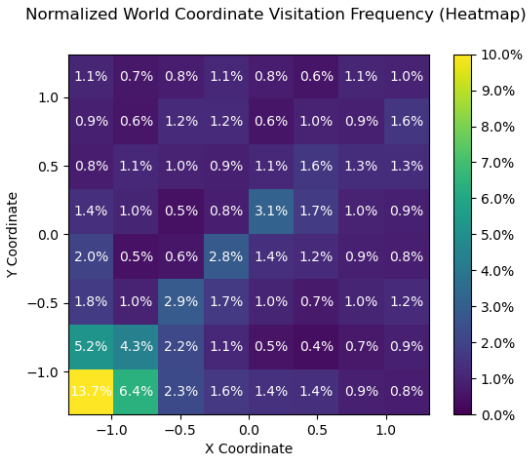

Figure 37: Visitation frequency heat map for Heuristic 2. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

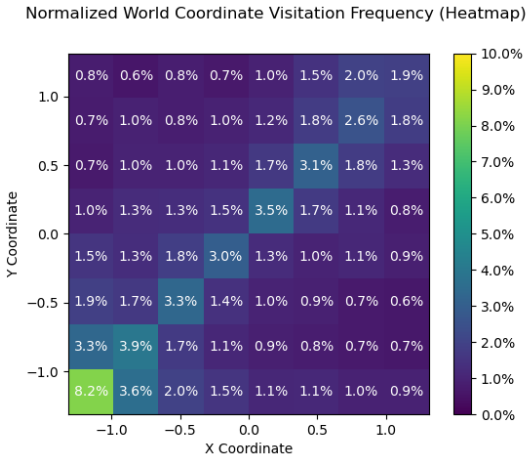

Figure 38: Visitation frequency heat map for Recalculation. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

4.4 Comparison of Solvers and Environments

Analyzing the results depicted in Figures 39 and 40 comparing the percent area coverage over time for Heuristic 1, Heuristic 2, MILP, and MILP with recalculation (MILP+Recalculation), it becomes evident that these approaches exhibit similar behaviors. MILP and Heuristic 2 showcase higher peaks in coverage over time. In contrast, MILP+Recalculation demonstrates lower coverage overall, despite MILP alone providing greater values. It is also interesting to see that the same solutions perform around 50% worse when tested in the ARGoS physics simulation (see Figures 39 and 40).

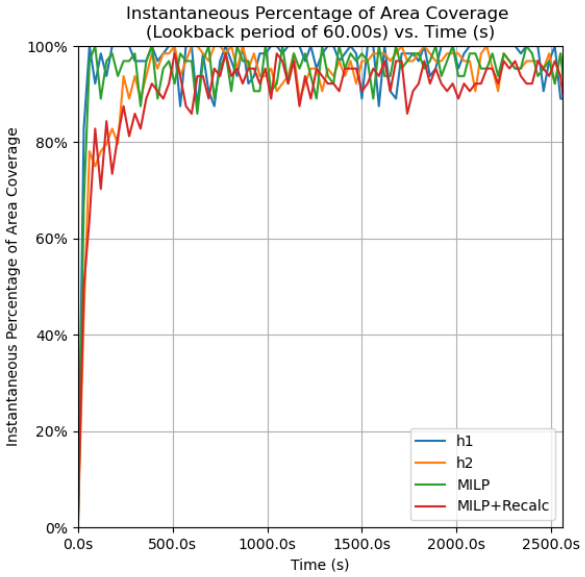

Figure 39: Solver Coverage Comparison using the Point-Mass Simulation. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

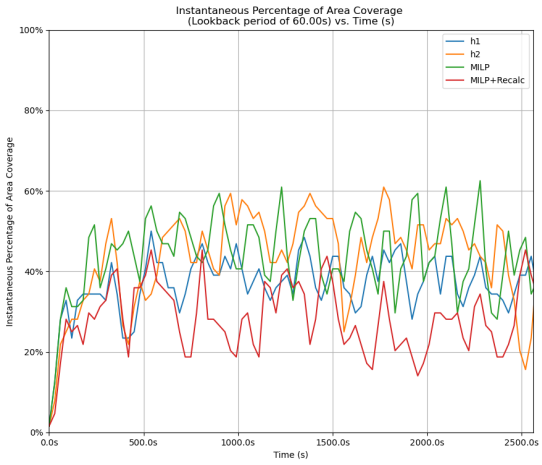

Figure 40: Solver Coverage Comparison using the ARGoS Simulation. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

In both ARGoS simulations and real-world scenarios using Vicon motion capture technology, the robots achieved notably higher percent area coverage compared to the point-mass simulations, which do not factor in collisions.

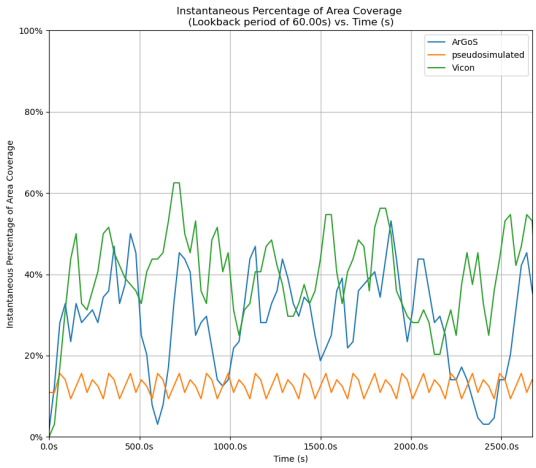

Figure 41: Environment Coverage Comparison. Problem parameters: $N = 8$, $S_{\text{side-length}} = 3km$, $R_{\text{surveillance}} = 265m$ (Equivalent to 64 Nodes), $L_{\text{max-capacity}} = 12.72$, $r = 3$.

4.5 Comparison of Heuristic 1 and Heuristic 2 with Point-Mass Simulations Under Varying Parameters

In our tests to improve how robots cover fields autonomously, we used simulations without collisions to focus solely on key factors. We looked at four main things: how much fuel the robots had, how many robots there were, how far they could see, and how fast they moved. We only compared Heuristic 1 (H1) and Heuristic 2 (H2) for these graphs.

First, the simulations showed that robots with a greater field of view could cover more ground (see Figure 42). This resulted in a greater percentage of coverage as the surveillance radius increased for all variations of speed. H1 generally performed better. This was an expected behaviour.

It was also observed that faster robots allowed for greater coverage (see Figure 43). With this graph, the number of robots was varied, so it can also be observed that H2 performed better with a greater number of robots. This was an expected behaviour.

Next, we found that having more robots meant better coverage overall (see Figure 44). As previously mentioned, H2 performs better with a greater number of robots, and we can clearly see that here. But H1 performs better than H2 with lower numbers of robots. Nevertheless, we see an expected outcome of the coverage being very high when there are a lot of robots on the field.

Lastly, we saw that giving robots more fuel meant they could cover more ground and get better field coverage (see Figure 45). As the number of robots increased, the coverage increased, but also, as the

fuel levels increased, the coverage increased. However, both H1 and H2 converge their own coverages with varied fuel capacities at 32 and 256 robots respectively. This suggests that the fuel capacities do not need to be increased (to achieve a high coverage) after a certain amount of robots is reached for each heuristic.

These tests help us understand how these factors work together to make robots better at covering fields on their own, which is important for many tasks in the real world.

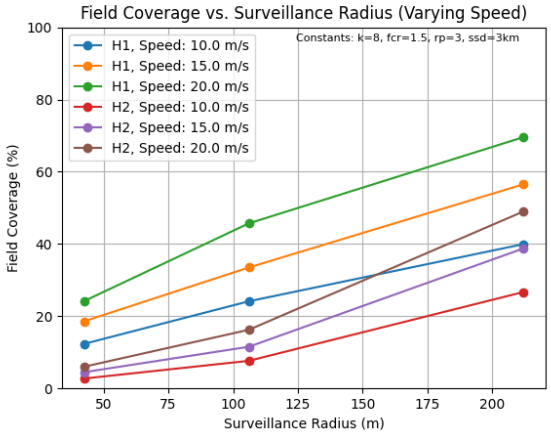

Figure 42: Field Coverage Vs. Surveillance Radius

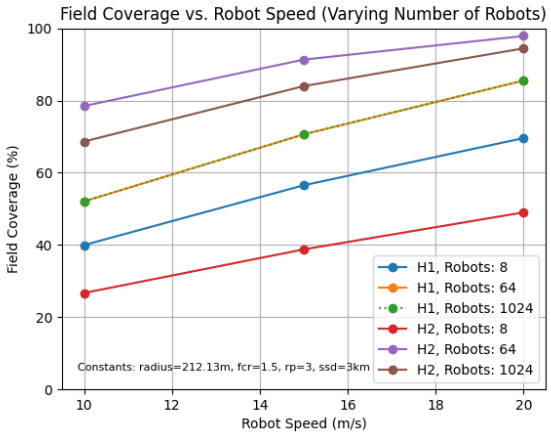

Figure 43: Field Coverage Vs. Robot Speed

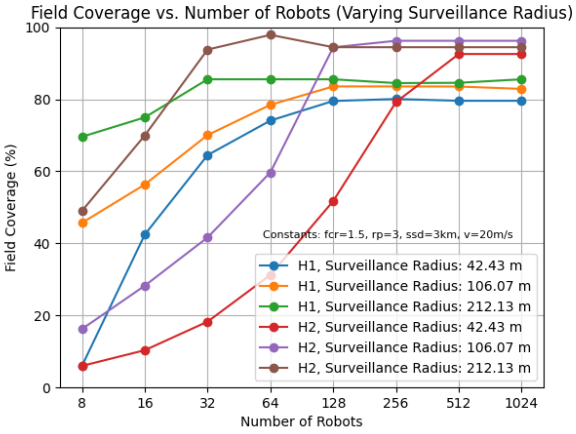

Figure 44: Field Coverage Vs. Number of Robots

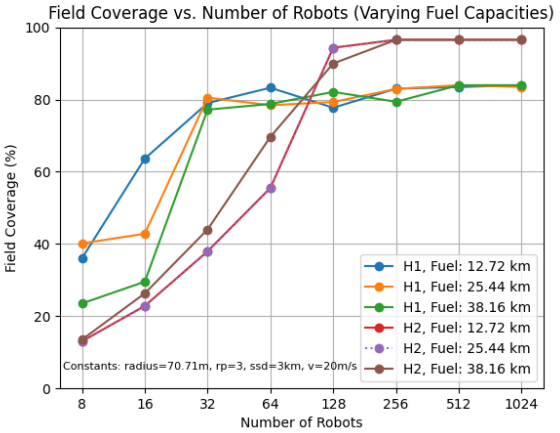

Figure 45: Field Coverage Vs. Fuel Capacity

4.6 Discussion

The comparison of different solvers, environments, and parameters is crucial in evaluating the performance and applicability of robotic systems in various scenarios. In this discussion section, we analyze the implications of these factors on the effectiveness of robotic paths and navigation. It is important to note here the effect of traffic in the results.

Firstly, the choice of solvers does not significantly affect the efficiency and optimality of robotic behavior. Comparing solvers such as heuristics, a Mixed Integer Linear Programming (MILP) formulation, and their variants, we observed trade-offs between computational complexity and solution quality. While heuristics offer speed and practicality, MILP guarantees optimality under certain conditions. Additionally,

MILP with recalculation introduces adaptability, and is not computationally heavy, once the solution has been generated, because it has a heuristic recalculation. Understanding these trade-offs enables us to select the most suitable solver based on the specific requirements of the task at hand.

Secondly, the environment plays a crucial role in shaping robotic behavior and performance. Simulated environments such as point-mass simulations provide us flexibility in generating solutions with varying parameters and fast run times. ARGoS simulations provide controlled settings for testing and optimizing with a physics-based system. Conversely, real-world deployments present challenges such as deadlocks and collisions. Comparing performance across simulated and real-world environments highlights the need for algorithms that can adapt to unpredictable conditions.

Furthermore, parameters such as the number of robots, robot speed, surveillance radius, and fuel capacity significantly impact exploration efficiency and coverage. Increasing the number of robots can enhance coverage but also introduces coordination challenges and resource constraints. High robot speeds expedite exploration and increase coverage, but can lead to increased energy consumption. Similarly, adjusting surveillance radius and fuel capacity influences the trade-offs between exploration range, energy efficiency, and path duration. Understanding the costs of these varying parameters is essential for optimizing the algorithms in the future.

In conclusion, the comparison of different solvers, environments, and parameters provides valuable insights into the strengths, limitations, and trade-offs of the multi-robot coverage problem. By systematically evaluating performance across diverse conditions, we can develop more robust and adaptable robotic systems capable of effectively operating in real-world environments. Future research directions may focus on addressing challenges such as scalability, robustness to uncertainty, and autonomy to further advance the capabilities of robotic exploration and navigation.

5 Conclusion

5.1 Summary

In this work, we add three novel elements to a previously posed problem. We create a problem formulation with both failures and battery constraints. Additionally, we use real world testing on our solutions.

We provide one baseline Mixed Integer Linear Programming (MILP) based solution, and two heuristic solutions to the problem. One heuristic solution is aimed at being a drop in replacement for MILP, and one is made to be responsive to failures as they happen, such that paths do not need to be recalculated. We show that our solutions are generated significantly faster than MILP, however, they are also less optimal than MILP. We also give an intuition of our process for generating these heuristics.

5.2 Future Work

In the future, we would like to add to this project by developing our heuristics with a full testing setup. At the time we were in development, this setup was not complete. We also hope to add additional tests of robots in ArGoS, in particular on large numbers of robots. While our point mass simulation was shown to be fairly accurate, the addition of simulated tests in a physics simulator would make our validation stronger. Additionally, we would like to add more mathematical reasoning to our heuristics.

In terms of significant expansion of the work, we feel that while resiliency to failures is important for swarm systems in real-world settings, there is a gap in the research for the Multi-Robot Persistent Coverage Problem. We also feel that testing these solutions for attack by an agent would also increase their usefulness.

Lastly, the solvers and environments we present could be tested on a more comprehensive set of metrics. In this work, we mainly used "percent coverage". We could have included its continuous counterpart, where we draw circles on the field where the robots are, take a union of the areas, and calculate the coverage as a percentage of the field (in units of area instead of nodes). Additionally, a metric that tracking the longest time a node has not been visited could be useful.

5.3 Lessons Learned

Discrete solutions for the problem of persistent coverage perform usually very well on the point-mass simulations, and this is expected because our discrete planners are designed to cover as many different nodes as possible in a given time. However, this approach has a very strong but implicit assumption: unplanned traffic. When nodes are allowed to be fully connected, the connection from one node to another could go over other nodes, and this creates more unplanned traffic. In the ARGoS simulations and real-world experiments we ran, we observed almost no difference in the coverage of nodes within a given look-back period. In this case, it feels like there is almost an upper bound of percent coverage (in simulation) we can reach given the problem parameters.

Additionally, we underestimated the time it would take for us to set up the ARGoS physics simulation and the Vicon setup in the lab. This was more so influenced by the time spent on solving and trying new approaches to avoid collisions and handling deadlocks. An earlier start on dealing with the issues around traffic would have been beneficial.

References

[1] S. Buchanan, "The future of agriculture: Robotic farming technology," March 2021. https://agamerica.com/blog/the-future-of-agricultural-robotics/.

[2] J. Wilder, T. Sane, and V. Chowdhar, "Reusable software components for multi-robot foraging," *(Undergraduate Interactive Qualifying Project No. E-project-042318-002942)*, 2018. Retrieved from Worcester Polytechnic Institute Electronic Projects Collection: https://digital.wpi.edu/concern/student_works/cc08hh22z.

[3] M. Kegeleirs, G. Grisetti, and M. Birattari, "Swarm slam: Challenges and perspectives," *Frontiers in Robotics and AI*, vol. 8, 2021.

[4] D. Mitchell, M. Corah, N. Chakraborty, K. Sycara, and N. Michael, "Multi-robot long-term persistent coverage with fuel constrained robots," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1093–1099, 2015.

[5] X. Meng, A. Houshmand, and C. G. Cassandras, "Multi-agent coverage control with energy depletion and repletion," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 2101–2106, 2018.

[6] L. Zhou, V. Tzoumas, G. J. Pappas, and P. Tokekar, "Resilient active target tracking with multiple robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 129–136, 2019.

[7] M. Ishat-E-Rabban and P. Tokekar, "Failure-resilient coverage maximization with multiple robots," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3894–3901, 2021.

[8] L. Zhou and P. Tokekar, "Multi-robot coordination and planning in uncertain and adversarial environments," *Current Robotics Reports*, vol. 2, pp. 147–157, Jun 2021.

[9] J. Song and S. Gupta, "Care: Cooperative autonomy for resilience and efficiency of robot teams for complete coverage of unknown environments under robot failures," *Autonomous Robots*, vol. 44, pp. 647–671, Mar 2020.

[10] T. Cabreira, L. Brisolara, and P. Ferreira, "Survey on coverage path planning with unmanned aerial vehicles,"

[11] R. Stern, *Multi-Agent Path Finding – An Overview*, pp. 96–115. 10 2019.

[12] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[13] D. S. Johnson and L. A. McGeoch, *8. The traveling salesman problem: a case study*, pp. 215–310. Princeton: Princeton University Press, 2003.

[14] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*, pp. 1928–1935, 2008.