

Decentralized Control of an Energy Constrained
Heterogeneous Swarm for Persistent Surveillance

by

NIKHIL KAMALKUMAR ADVANI

A Thesis

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

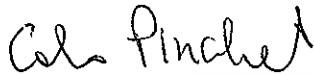
Degree of Master of Science

in

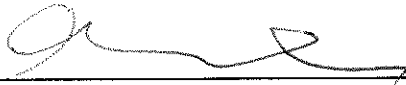
Robotics Engineering

May 2017

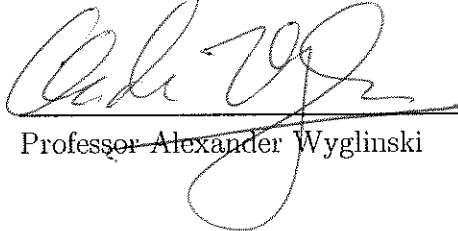
APPROVED:



Professor Carlo Pinciroli, Thesis Advisor



Professor Eugene Eberbach



Professor Alexander Wyglinski

Abstract

Robot swarms are envisioned in applications such as surveillance, agriculture, search-and-rescue operations, and construction. The decentralized nature of swarm intelligence has three key advantages over traditional multi-robot control algorithms: it is scalable, it is fault tolerant, and it is not susceptible to a single point of failure. These advantages are critical to the task of persistent surveillance - where a number of target locations need to be visited as frequently as possible.

Unfortunately, in the real world, the autonomous robots that can be used for persistent surveillance have a limited battery life (or fuel capacity). Thus, they need to abandon their surveillance duties to visit a battery swapping station (or refueling depot) a.k.a. depots. This down time reduces the frequency of visitation. This problem can be eliminated if the depots themselves were autonomous vehicles that could meet the (surveillance) robots at some point along their path from one target to another. Thus, the robots would spend less time on the 'charging' (or refueling) task.

In this thesis we present decentralized control algorithms, and their results, for three stages of the persistent surveillance problem. First, we consider the case where the robots have no energy constraints, and use a decentralized approach to allow the robots choose the best target that they should visit next. While the selection process is decentralized, the robots can communicate with all the other robots in the swarm, and let them know which is their chosen target. We then consider the energy constraints of the robots, and slightly modify the algorithm, so that the robots visit a depot before they run out of energy. Lastly, we consider the case where the depots themselves can move, and communicate with the robots to pick a location and time to meet, to be able to swap the empty battery of a robot, with a fresh one. The goal of persistent surveillance is to visit target locations as frequently as possible, and thus, the performance measurement parameter is chosen to be the median frequency of visitation for all target locations. We evaluate the performance of the three algorithms in an extensive set of simulated experiments.

Acknowledgements

I would like to express my deepest gratitude to my adviser, Dr. Carlo Pinciroli for giving me an opportunity to work on this thesis in the Novel Engineering for Swarm Technologies (NEST) Lab. His guidance, support and approachability have been invaluable to me. I would also like to thank Professor Alex Wyglinski, Professor Eugene Eberbach, and Professor Michael Ciaraldi for being a part of my thesis committee and providing insights into different aspects of the thesis.

The NEST Lab has been a great place to collaborate and learn. Thanks to everyone in the lab for our lively brainstorming sessions. Thanks to Ari Goodman for our discussions about energy aware swarms and MILP.

I would also like to thank my parents and brother for the support and motivation they have offered me, without which I could never have reached here.

Contents

Abstract	i
Acknowledgements	ii
List of Figures	v
1 Introduction	1
1.1 Persistent Surveillance	3
1.2 Contributions	3
1.3 Thesis Structure	4
2 Literature Review	5
2.1 Persistent Surveillance With No Energy Constraints	5
2.2 Mechanisms for Automated Battery Recharging and Replacement	7
2.3 Ideal Depot Location	8
2.4 Persistent Surveillance Considering Energy Constraints	8
2.4.1 Considering Static Depots	8
2.4.2 Considering Moving Depots	10
3 Methodology	11
3.1 Problem Formulation	11
3.2 Persistent Surveillance Without Energy Constraints	12
3.3 Persistent Surveillance Considering Energy Constraints - With Static Depots	15
3.4 Persistent Surveillance Considering Energy Constraints - With Moving Depots	18
3.4.1 Moving Depots with Reactive Help Requests	18
3.4.2 Preemptive Help Requests	21
4 Results	26
4.1 Performance Evaluation Metric	26
4.2 Experimental Setup	27
4.3 Results	28
5 Conclusions And Future Work	31
5.1 Summary and Conclusion	31
5.2 Future Work	32

5.2.1 Diversity Metric 33

Bibliography **36**

List of Figures

1.1	Honey bees are one of the most common sources of inspiration for swarm intelligence	2
1.2	Relation between Multi Robot and Robot Swarms	2
3.1	The Persistent Surveillance Problem	11
3.2	Decentralized Control for Persistent Surveillance with No Energy Constraints	14
3.3	Robot Behaviour - No Energy Constraints	14
3.4	Ideal location of Depots	15
3.5	Flowchart for our decentralized control algorithm for persistent surveillance with energy constraints and static depots	17
3.6	Robot Behaviour - Energy Constraints - Static Depots	18
3.7	Flowchart for our decentralized control algorithm for persistent surveillance with energy constraints and moving depots responding to Reactive requests	20
3.8	Robot and Depot behaviour considering Reactive Help Requests	21
3.9	Flowchart for our decentralized control algorithm for persistent surveillance with energy constraints and moving depots responding to Preemptive and Reactive Help requests	24
3.10	Robot and Depot behaviour considering Preemptive and Reactive Help Requests	25
4.1	Results for [8,16,32,0.1]	28
4.2	Results for [4,16,64,0.1]	29
4.3	Results for [4,8,64,0.1]	29
4.4	Results for [4,8,32,0.02]	30
5.1	Graph Idleness vs Diversity	34

Chapter 1

Introduction

In the past few years there has been an increase in the number of commercial applications that employ Multi-Robot Systems. For some specific robotic tasks, such as exploring an unknown planet, pushing objects or cleaning up toxic waste, it has been suggested that rather than sending one very complex robot to perform the task it would be more effective to send a number of smaller, simpler robots (Dudek et al., 1996). A Multi-Robot Systems is composed of multiple interacting intelligent agents within an environment, working in a cooperative manner to achieve a common goal.

The advantage of Multi-Robot Systems is that they are more robust and that damaged robots are much more economical to replace compared to the case where a single complex robot is damaged. When these systems use a centralized controller to ensure coordination between all the intelligent agents in the system they have a single point of failure (the central controller). Moreover, computing the optimal behavior for each robot is often NP-hard so these systems cannot operate efficiently in an online manner. In most cases, as the scale of operations and the number of robots in the system increases, the computation time required increases to such an extent that a suboptimal solution has to be used.

Swarm robotics systems [1] are a subset of Multi-Robot Systems that take inspiration from biological examples, especially from social insects like ants and bees. These animals show complex collective behaviours even though they use very little direct communication. Only locally available information is exploited and indirect communication is obtained by modifications of the environment using stigmergy [2]. Thus, swarm intelligence, which is decentralized-distributed, circumvents the aforementioned problems faced by traditional Multi-Robot Systems.



FIGURE 1.1: Honey bees are one of the most common sources of inspiration for swarm intelligence

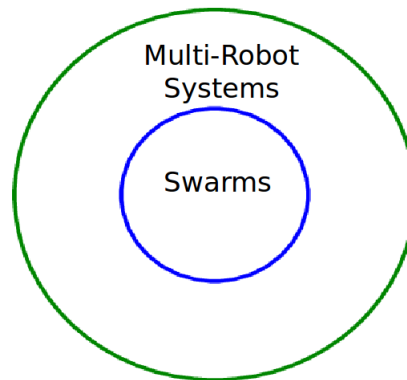


FIGURE 1.2: Relation between Multi Robot and Robot Swarms

While discussing Multi-Robot Systems, it is important to understand the difference between centralized, decentralized and distributed algorithms. The terms ‘centralized’ and ‘decentralized’ describe the decision making process of the algorithm. A centralized algorithm is one where a single robot makes the decisions regarding the behavior of every robot in the system. A decentralized algorithm is one where every robot makes decisions about its own behavior. On the other hand, the term ‘distributed’ describes whether or not the agents communicate with each other before a decision is made. Thus, it is possible to have a ‘centralized-distributed’ system or a ‘decentralized-distributed’ system. However, if every robot is being given instructions by a single robot, it is unlikely that the robots would have to communicate with each other. On the other hand, in a decentralized system, like swarm robot systems, the robots need to communicate with other robots to effectively complete the task.

1.1 Persistent Surveillance

The problem being addressed in this thesis is that of Persistent Surveillance. This problem states that there are numerous points of interest (a.k.a. ‘targets’) in the environment that need to be visited as frequently as possible for surveillance. Some applications of persistent surveillance are air quality sampling [3], border security [4], visual inspections of power plants and pipelines [5], traffic surveillance [6] or disaster management [7]. The security applications of persistent surveillance are highlighted by an article [8] published in 2016 which states that a company named ‘Persistent Surveillance Systems’ was hired by the Baltimore Police Department for more than 9 months to persistently survey 64 square kilometers of the city of Baltimore to ensure the safety of the citizens.

In the past decade there has been a significant amount of research on quadcopters. They are ideal to perform the persistent surveillance task. However, quadcopters have a limited battery life, and would thus need to recharge or replace their batteries periodically while executing the task. Most of the literature on persistent surveillance does not account for this fact. To the best of our knowledge, there has been no decentralized solution proposed to the persistent surveillance problem that considers the energy constraints of the surveillance robots.

1.2 Contributions

In this thesis we present algorithmic solutions to 3 stages of the persistent surveillance problem. In the first stage, we consider the case where the robots do not have energy constraints, and must decide which target to visit next in a decentralized manner. In the second stage, we present the first decentralized solution for the persistent surveillance problem while considering the energy constraints of the robots. Since the position of the targets is known beforehand, we use integer linear programming to find the optimal location of the automated battery charging/battery replacing stations (a.k.a. ‘depots’). In the third stage we present two algorithms that consider moving depots instead of the static depots presented in the previous stage. We end by comparing the results of the algorithms from the second and third stage. Since the swarm of robots performing this task now contains the ‘surveillance’ robots and the ‘depots’, the swarm is heterogeneous.

To the best of our knowledge, the control algorithm developed in this thesis is the first work that considers a decentralized strategy for persistent coverage while accounting for the energy constraints of the robots. By extension, it is also the first work in using a decentralized control strategy considering mobile depots.

1.3 Thesis Structure

The organization of this thesis is as follows: Chapter 2 introduces the different approaches taken to solve the Persistent Surveillance problem. Chapter 3 describes the formulation and the solution to the 3 stages of the problem described in Section 1.2. Chapter 4 elaborates on the experimental setup, performance metric, and results of the algorithms. Chapter 5 describes our conclusions, and our recommended directions for future work.

Chapter 2

Literature Review

The existing algorithms in literature for persistent surveillance using Multi-Robot systems present many variations in terms of strategy, cooperation scheme, performance evaluation, communication paradigm etc. The structure of this chapter is as follows: we start by presenting approaches to the persistent surveillance problem that do not consider energy constraints of the robots. We then discuss the different autonomous battery charging and replacing mechanisms that have been developed. This is followed by a description of algorithms that determine the ideal depot locations. Finally, we discuss approaches to the persistent surveillance problem that consider energy constraints of the robots.

2.1 Persistent Surveillance With No Energy Constraints

A centralized controller using Voronoi partitions to split the region to be surveyed into subsections is presented in [9]. The centroids of these subsections are determined to be the optimal positions that the robots should take up to maximize a collective reward function. The robots occupy their respective positions and remain static till the end of the experiment. In [10] the authors consider N agents, each with bounded sensing capabilities; and present a dynamic control algorithm that guarantees that every point in the region is visited atleast once. The robots communicate with each other to divide the area among themselves. Each robot then visits every point on the perimeter of the area that it is assigned. While these solutions can be used for inspiration to solve the problem of persistent surveillance, they are solutions to the coverage surveillance problem. In this problem, the exact points of interest are not provided, but an entire area needs to be surveyed. The coverage surveillance problem is a super-set of the persistent surveillance problem.

A solution to the persistent surveillance problem is presented in [11] where the authors describe an algorithm that finds routes for every robot in the system, such that each robot has a fixed set of targets to be visited in a cyclic manner. The authors model the problem as a Travelling Salesman Problem (TSP) [12] for a single robot, and find the optimal path for that robot. This path is then partitioned to find paths for each robot in the system. The algorithm guarantees a lower bound of performance, which is relative to the optimal frequency of visits for a single agent system. Since visiting a target can be considered to be a ‘task’, the persistent surveillance problem can be modelled as a Multi-Robot Task Allocation (MRTA) Problem [13]. Some centralized strategies to solve the MRTA problem are presented in [14], [15] and [16] where the general approach is that the robots bid on the task, and communicate their situational awareness to a central server which decides which robot should perform which task.

Pioneering work in using decentralized algorithms for persistent surveillance was presented in [17] where the authors proposed several architectures for multi-robot surveillance as well as 3 evaluation criteria. This work was evaluated and expanded in [18] where the authors compared 5 persistent surveillance algorithms. Among the 4 online algorithms presented in this paper, the Cyclic algorithm for Generic Graphs (CGG) shows the best performance. In this algorithm the persistent surveillance problem is modelled as a graph where the locations to be surveyed (a.k.a. ‘targets’) are the vertices of the graph. Each robot looks for Hamiltonian cycles in the graph to visit all the vertices. The robots use a fast heuristic algorithm proposed in [19] to calculate the sub-optimal Hamiltonian paths. A shortcoming of these algorithms is that, in all of the decentralized algorithms the robots do not communicate their intended target with the other robots in the system. The same authors overcome this shortcoming in [20]. Another interesting decentralized approach is presented in [21] which is a biologically inspired approach. In this approach, the intelligent agent continuously releases a pheromone which decays with time. As the strength of the pheromone at a particular area reduces, the swarm agents are more attracted towards it.

Other approaches that have been proposed over the years include Markov Decision Processes (MDP) [22], reinforcement learning [23], where the robots’ strategies are automatically adapted to the topology of the environment, and optimal control [24]. A comprehensive survey of persistent surveillance algorithms that do not consider fuel constraints is presented in [25]. The next stage of the persistent surveillance problem is to consider the energy constraints of the robots.

2.2 Mechanisms for Automated Battery Recharging and Replacement

Some of the biggest problems in swarm robotics are energy related. With the increased interest towards research involving quadrotors, and considering their limited flight time, the last decade has seen significant progress in the development of automatic battery charging and battery replacing platforms. By automating the battery swapping or charging process, the overall mission time can be greatly increased.

The first work done to demonstrate the autonomous charging of an Unmanned Aerial Vehicle (UAV) on a mobile charging station was presented in [26]. In this paper the authors attached copper contacts to the quadrotor's base. The charging station is designed such that when the quadrotor lands on it, the inverted pyramid-like internal structure causes the quadrotor to slide down in a manner that the copper contacts come in contact with the charging port. This charging station was then mounted on an RC car, which made it the first mobile UAV charging station. More recently, in [27] the authors have developed a charging station for a Micro Aerial Vehicle (MAV), and modified the MAV to include magnets below the 4 rotors of the quadrotor. When the quadrotor lands on the charging station, the charging ports on the quadrotor come in contact with electrical contacts on the charging station and charge the quadrotor's battery. The magnets on the quadrotor ensure robust and reliable contact with the charging pad. Other examples of autonomous charging include [28], where the charging pad is solar powered which is mounted on the quadrotor itself. Autonomous recharging capabilities are not just being developed for quadrotors. The iRobot Roomba [29] is a well known commercial robotic vacuum cleaner. When the product first launched, the Roomba would run out of battery while vacuuming, and would have to be manually carried to a power outlet and plugged in to charge. In more recent versions of the product, the Roomba detects when it is going to run out of power, finds its charging dock (using an infrared sensor and an infrared beacon which is mounted on top of the dock). This ability to autonomously charge itself has given the Roomba the capability to vacuum large areas that it would not have been able to do when the product was first launched.

In [30] the authors have developed a mechanism to perform a cold swap of the battery of a RC helicopter. The mechanism can not only swap batteries, but can charge them as well. Thus, the helicopter can theoretically run indefinitely without requiring human intervention. Another interesting approach to the battery swapping problem on an RC helicopter is presented in [31] where arms on the landing pad position the RC helicopter before its battery is removed, and a new one is inserted. The problem with cold swapping is that the electronics on board need to be shut down when the battery is removed, and

need to be restarted when the new battery is inserted. A mechanism to enable hot swapping of a quadrotor battery is introduced in [32]. When the quadrotor lands on the swapping mechanism, it is clamped onto a pad and provided with shore power before the battery is removed, and this enable the hot swapping. The mechanism takes an average of 11.8 seconds to perform the hot swap, with a deviation of 3.0 seconds.

A study and provide an analytic answer to the question of whether a battery charging or replacement platform is preferable is presented in [33]. The authors also developed and compared various recharge station designs and proposed a conceptual replacement platform.

2.3 Ideal Depot Location

Now that we know that the battery can be autonomously recharged or replaced, the next question that arises is, ‘what is the ideal location of the depots?’ This is a variant of the Facility Location Problem [34] which is concerned with the optimal placement of facilities to minimize transportation costs. This problem is expanded upon in Section 3.3. It is also aligned with the Charging Station Placement (CSPL) problem as described in [35]. A novel approach to ideal depot placement is described in [36]. Here the depots start at a random location and move towards robots when they are running low on energy. Every time a robot couples with the depot, the robot evaluates the quality of the dock’s position. This evaluation is used to improve the position of the depot iteratively. This is because, in non-stationary tasks such as surveillance, the location of the docking station has a significant impact on the task performance of the team, since the optimal docking location may vary over the mission.

2.4 Persistent Surveillance Considering Energy Constraints

If the robots run out of energy, they will get stranded and won’t be able to move. Thus, the robots need to account for their energy level, and the distance to the nearest depot before choosing a target to visit.

2.4.1 Considering Static Depots

In [37] and [38], the authors propose a solution whereby they can use a team of MAVs to perform persistent surveillance while ensuring that the MAVs return to their base station (or charging station) when their battery level falls below a certain threshold. They start

by exploring the possibility of modelling the problem as a Travelling Salesman Problem (TSP), and spacing agents evenly along the solution. However, this approach does not provide the flexibility that is needed to guarantee the frequency of visitation. They then consider modelling the problem as a Dynamic Vehicle Routing Problem (DVRP) [39]. This allows the algorithm to add targets in a dynamic manner. But this doesn't guarantee the frequency of visitation either, and thus, the authors decided to model it as a Vehicle Routing Problem with Time Windows (VRPTW). The Time Windows extension to the VRP is used to be able to guarantee the frequency of visitation at all targets and also scale the algorithm (in terms of number of robots). This is done by solving the problem optimally within a set time horizon and then executing this path for a small amount of time, shifting the horizon, and resolving. This amounts to a receding horizon framework. The problem is formulated as a Mixed Integer Linear Programming (MILP) [40] problem and solved using combinatorial optimization. When the energy level falls below a threshold, the MAV return to the base station for recharging. However, for this approach to guarantee that the MAV would have enough energy to return to the base station from the furthest target, the threshold would be quite high, and the system would not be very efficient.

Another solution to the problem of Persistent Coverage with Fuel Constrained Robots is described in [41]. This paper considers the position of the depots, and the energy capacity of the robots while finding tours for the robots. The authors refer to the problem as Multi Robot Persistent Coverage Problem (MRPCP). The goal of the paper is to 'find a collection of tours (one for each robot), such that every target is visited by the robots and the minimum frequency with which a target is visited is maximized'. The problem is initially solved using combinatorial optimization without considering fuel constraints. For edges whose cost exceeds the the fuel capacity, a new path is found that connects the 2 vertices but passes through a depot. The new path represents the true cost between the 2 vertices. This updated cost is used to find a Hamiltonian path using Christofides algorithm [42]. This path is then split so that where each robot is assigned a segment. Every segment is then modified to add a detour to a depot wherever necessary. Each robot must now follow this segment in a cyclic manner which results in each target being visited frequently. This approach is inspired by [43] and [44]. The difference is that [43] considers a single robot with Dubin's car motion model, whereas [41] considers multiple robots. [44] is an extension of [43] that considers multi-robot deployments and uses incremental local searches to improve on initial feasible solutions for each robot. The initial solutions are generated by assigning targets to robots and then computing the path using heuristics described in [43]. On the other hand, [41] distributes targets while generating the initial solution.

2.4.2 Considering Moving Depots

Instead of having the robots go out of their way to visit the depots, there has been some work published recently which explores the possibility of having the depots move towards the robot when the robot is running out of energy. In [45] and [46] the authors consider the case where the paths of the aerial vehicles being used for surveillance is known. They present an algorithm to find the optimal trajectory of the ground vehicles (which are the moving depots), such that these vehicles rendezvous with the aerial vehicles to recharge their batteries. This is done using a discrete acyclic graph (DAG) based approach. Finding the optimal trajectory is a variant of the TSP and is an NP-hard problem. Thus, MILP is used to find the optimal trajectory.

Similarly, in [47], the authors consider the case where an energy aware multi UAV system is to be used for mapping a region. It is assumed that this region has some roads that are accessible by UGVs (which are the moving depots). They present an algorithm to find the best rendezvous points for the UGVs and UAVs (to recharge the UAVs), while considering the energy constraints of the UAVs. The path of the UAVs is then calculated while accounting for these rendezvous points.

The limitation of these approaches is that they are offline, and they optimize the path of either the depots or the surveillance robots, while keeping the path of the other constant. This reduces the flexibility of the system.

Chapter 3

Methodology

3.1 Problem Formulation

The persistent surveillance problem can be represented as an fully connected undirected graph $G(V, E)$ where V represents the set of vertices (which are the targets that need to be persistently visited or surveyed), i.e. $v_i \in V$. E represents the set of edges that connect all the vertices, i.e. $e_{i,j} \in E$. Since it is assumed that there are no constraints on the motion of the depots, the cost c of traversing an edge is the euclidean distance between the vertices being connected by that edge. Thus, G corresponds to the topological map for the surveillance operation and it is assumed to be known a priori. This can be seen below in Figure 3.1. The image on the left is a Google Earth view of the Worcester Polytechnic Institute campus. The red circles indicate the targets that need to be surveyed. The representation of this problem as a graph is shown in the image on the right. The edges between vertices are shown in blue

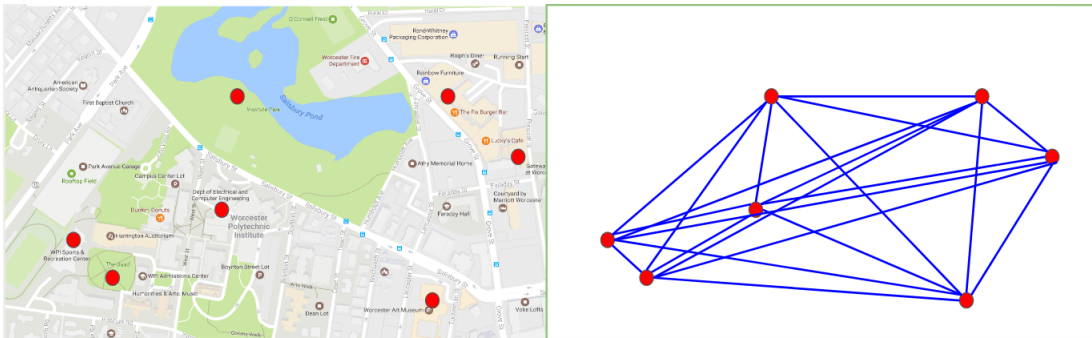


FIGURE 3.1: The Persistent Surveillance Problem

As mentioned in Chapter 1, in this thesis we present decentralized control algorithms for 3 stages of the persistent surveillance problem. They are described in Sections 3.2, 3.3 and 3.4.

3.2 Persistent Surveillance Without Energy Constraints

In the first case, we build on a solution provided in [20] to solve the persistent surveillance problem in a decentralized manner without considering energy constraints on the robots. While the authors of [20] describe a probabilistic approach for the robot to decide which target to visit next, we have used a deterministic algorithm to do the same.

In this algorithm, the robots are incentivised to visit a target by assigning a ‘reward’ value at each of the targets. The reward at a target is equal to 5 times the time since the target was last visited. For the robot to decide which target to visit next, the robot evaluates the gain at each target. The gain of a target for a robot is the ratio of the reward at that target to the distance that the robot would have to travel to get to that target. Thus, the gain of a robot r at target t is given by:

$$Gain_{r,t} = \frac{t_{reward}}{dist(r,t)}$$

Algorithm 1 Individual robot - No energy constraints

```

closed_list = [ ]
while true do
  list_of_gains = [ ]
  for target in targets do
    | list_of_gains.append(target.reward/dist(robot,target))
  end
  target = find_best_admissible_target(list_of_gains)
  robot.move(target)
  closed_list.remove(target)
end

```

The algorithm followed by each robot is shown in Algorithm 1. The robot starts by evaluating its gain at every target. Once this is done, it picks the target with the maximum gain value. It is possible that numerous robots in the system pick the same target, and move there. This would drastically reduce the performance of the algorithm. To ensure that this does not happen, the robots maintains a list of targets that are being visited

at that point in time. This is known as the ‘*closed list*’. Any target that is not in the closed list is considered to be an ‘*admissible target*’. Thus, when a robot picks a target to visit, it checks if the target is admissible before it starts moving there. If the target is admissible the robot adds the target to the closed list and moves there. Alternatively, if the target is not admissible, the robot sets the gain value for that target to 0. This cycle is repeated till the robot finds the best admissible target. The algorithm to find the best admissible target is shown in Algorithm 2, where the value returned is the index of the best admissible target.

Algorithm 2 Find best admissible target

potential_target = index(max[list_of_gains])

if *potential_target not in closed_list* **then**

 closed_list.append(potential_target)

 return potential_target

else

 list_of_gains[potential_target] = 0

 return find_next_target(list_of_gains)

end

When a robot reaches the target it had chosen, it deletes the target from the closed list, and looks for the next, ‘best admissible target’. The flowchart for this algorithm is shown in Figure 3.2.

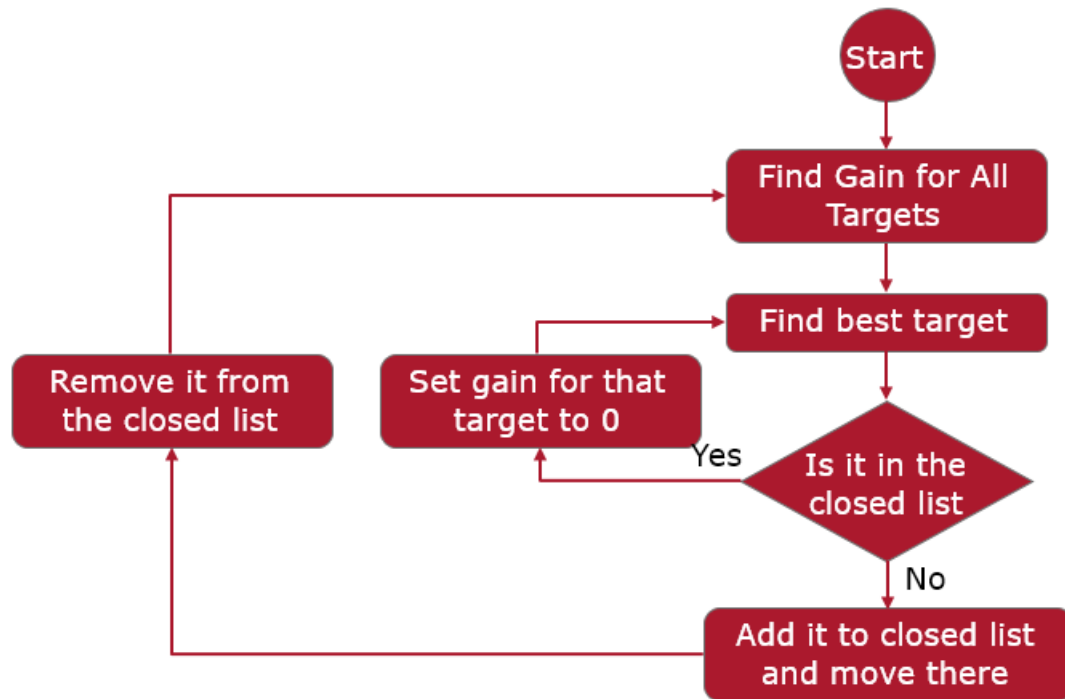


FIGURE 3.2: Decentralized Control for Persistent Surveillance with No Energy Constraints

The behaviour of the robots can be observed in Figure 3.3 below. The robot is represented by the white circle and the targets that must be visited are the white squares. The value of the reward at each target is shown near the top-right corner of each target. When the algorithm starts, all the targets have the same reward value. Since the robot is the closest to the upper left target, the gain at that target would be the highest. Thus, the robot moves towards the upper left target. Once it visits this target, the reward here falls to 0, and the robot moves to the target with the best gain.

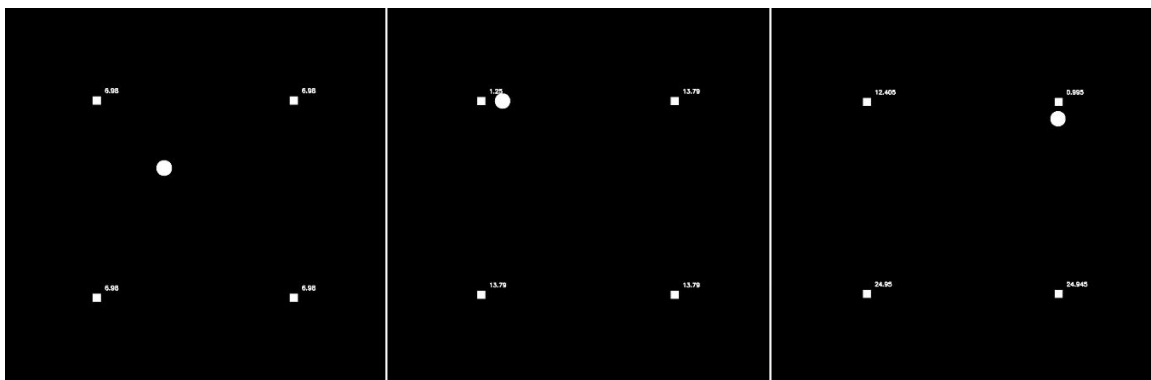


FIGURE 3.3: Robot Behaviour - No Energy Constraints

3.3 Persistent Surveillance Considering Energy Constraints - With Static Depots

Robots have a limited battery life, and thus, they need to visit depots to replace their batteries. As mentioned in Section 2.2, mechanisms that demonstrate automatic battery swapping capabilities have been developed over that past few years. These mechanisms can be used at the depots.

Since the locations of the targets are known, the optimal location of the depots can be calculated. This is known as the Facility Location Problem [34]. If the world has n targets and m depots, and $dist_{ij}$ is the distance between a target i and a depot j , the problem can be formulated as:

$$\text{minimize } \sum_{i=1}^m \sum_{j=1}^n dist_{ij} x_{ij}$$

$$\text{subject to } \sum_{j=1}^n x_{ij} = 1, \quad i = 1, \dots, m.$$

$$x_{ij} \leq y_j, \quad i = 1, \dots, m. \quad j = 1, \dots, n.$$

where $y_j = 1$ if location j is selected as position of depot $y_j = 0$ otherwise

$$x_j = 1 \text{ if location } j \text{ is closest to target } i \quad x_j = 0 \text{ otherwise}$$

Consider the case where 10 targets are distributed as shown in the left image of Figure 3.4. If 3 depots had to be placed, the optimal location of the depots would be as shown by the red circles in the right image.

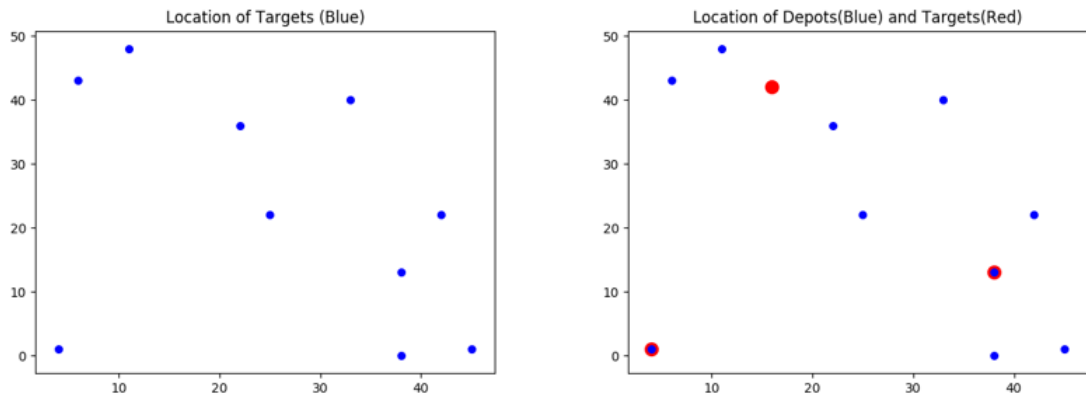


FIGURE 3.4: Ideal location of Depots

The algorithm followed by each robot is shown in Algorithm 3. It is essential to ensure that a robot does not become stranded. Thus, before it starts moving towards a target it checks to ensure that it has enough energy to first move to that target, and then move to the depot closest to that target. If it does, this is called a ‘feasible target’.

Algorithm 3 Individual robot - Static Depots

```

while true do
  closed_list = [ ]
  list_of_gains = [ ]
  robot.moving = False
  for target in targets do
    | list_of_gains.append(target.reward/dist(robot,target))
  end
  while robot.moving = False do
    target = find_best_admissible_target(list_of_gains)
    if robot.energy_level >= dist(robot,target) + dist(target,nearest_depot) then
      | robot.moving = True
      | robot.move(target)
    else
      | list_of_gains[target] = 0
    end
  end
end

```

Similar to the algorithm described in Section 3.2, the robot starts by finding the list of gains for all targets. It then finds the best admissible target and confirms that it is feasible before it starts moving towards that target. If the target is not feasible, the gain value of that target (for that robot) is set to 0. When the robot’s energy level is so low that none of the targets are ‘feasible’, the gain of all the targets would be 0. In this case, the robot looks for the closest depot and heads there to replace its battery. Once the battery is replaced, the robot evaluates the gain of all the targets and finds the best admissible and feasible target to move to. The flowchart for this algorithm is shown below in Figure 3.5.

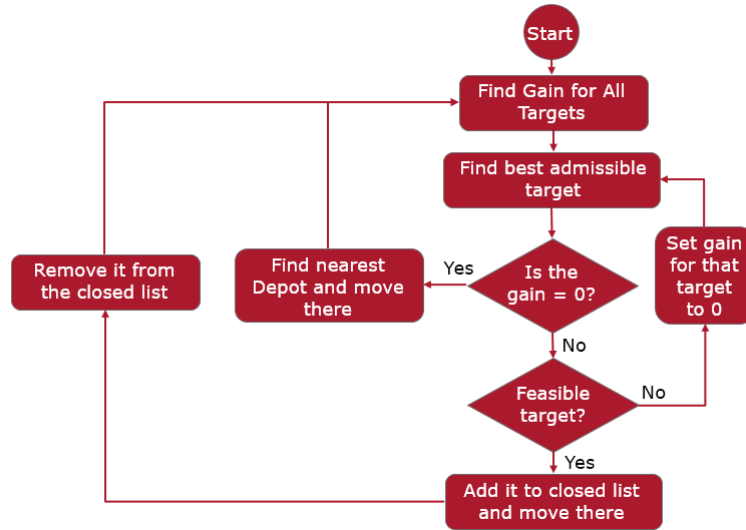


FIGURE 3.5: Flowchart for our decentralized control algorithm for persistent surveillance with energy constraints and static depots

The behaviour of the robots can be observed in Figure 3.6. The robot is the green circle. Targets are white squares that turn green after they are visited by the robot. The hollow squares are the depots. The images are arranged in a clockwise manner starting from the top-left image. In the second image, after the robot visits the target in the lower left corner, it is clear that the reward would be highest at the unvisited (white) target. However, since the robot does not have enough energy to visit that target and then reach the closest depot, it decides to visit a target with a lower reward instead. In the third image (bottom-right) the robot energy level is so low that none of the targets are feasible, and thus, it heads to the closest depot. After its batteries are replaced, it heads towards the unvisited target.

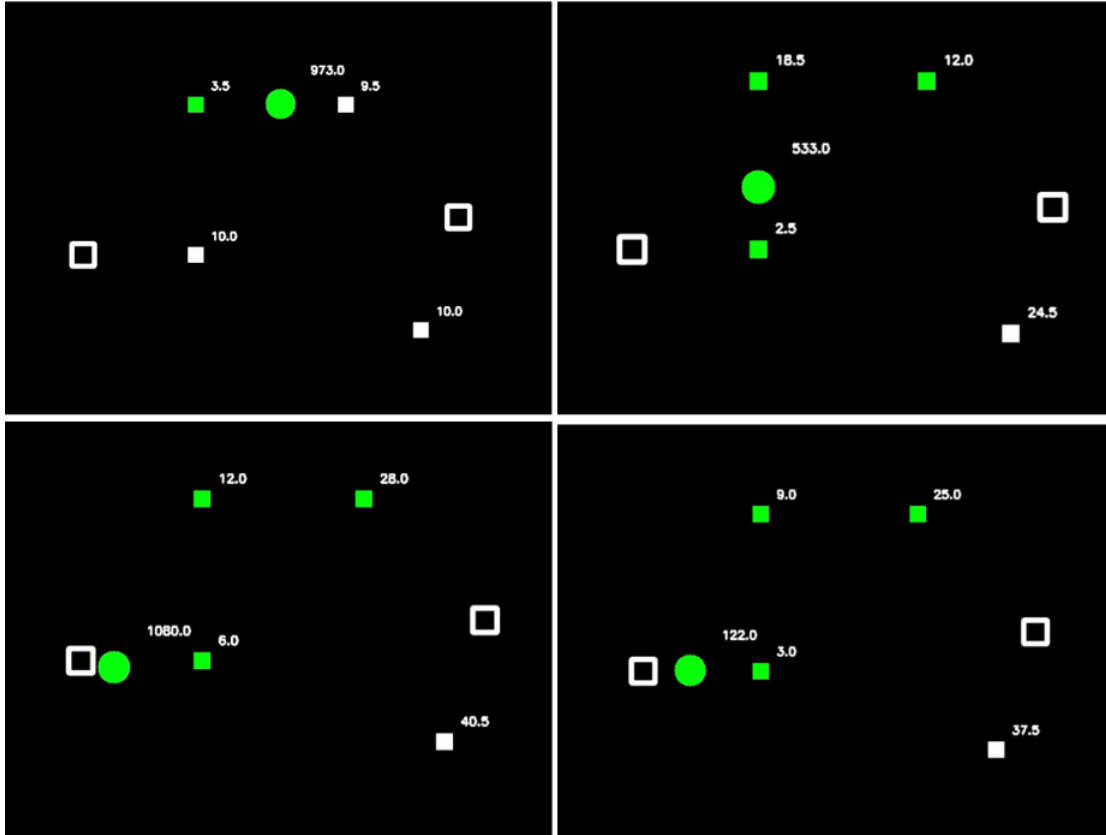


FIGURE 3.6: Robot Behaviour - Energy Constraints - Static Depots

3.4 Persistent Surveillance Considering Energy Constraints - With Moving Depots

Each robot spends a significant amount of time visiting depots to replace its batteries. In order to reduce this down time, we explore the possibility of having the depot be a mobile robot that can rendezvous with the robot at a point on the robot's path. We have developed two algorithms that would ensure that depots move towards robots to replace their batteries when the robots ask for help i.e. when their batteries need to be replaced. The difference between the two algorithms is the set of circumstances under which a robot would ask the depots for help.

3.4.1 Moving Depots with Reactive Help Requests

The help requests are known as 'reactive' help requests because a robot sends out such a request when it does not have fuel to reach its next desired target. Thus, in a way, it reacts to having low energy by sending out a 'reactive help request'. The reason for the term 'reactive' will become more evident in Section 3.4.2. In the current section, the

terms ‘reactive help request’, and ‘help request’ mean the same thing. The algorithm followed by each robot is shown in Algorithm 4.

Algorithm 4 Individual robot - Moving Depots - Reactive Requests

```

while true do
  closed_list = []
  list_of_gains = []
  robot.moving = False
  for target in targets do
    | list_of_gains.append(target.reward/dist(robot,target))
  end
  while robot.moving = False do
    target = find_best_admissible_target(list_of_gains)
    if robot.energy_level >= dist(robot,target) then
      | robot.moving = True
      | robot.move(target)
    else
      | publish reactive help request
      if depot reply with Rendezvous Point (RP) within time limit then
        | robot.moving = True
        | robot.move(RP)
        | robot.move(target)
      else
        | list_of_gains[target] = 0
      end
    end
  end
end

```

In this algorithm the robot finds its best admissible target and checks if it has enough energy to get there. If not, it publishes a request for help and starts a timer. This help request consists of the robot’s information, the target’s information, and the gain that the robot calculated when it decided to visit this target.

$$\textit{Help Request} = [\textit{Robot}, \textit{Target}, \textit{Gain}_{\textit{Robot}, \textit{Target}}]$$

The robot then waits for a depot to reply and confirm that it can meet the robot. If it does not receive a conformation before the timer runs out, the gain value of that target is set to 0, and the robot looks for its best admissible target.

On the depot side of things, all depots are constantly on the lookout for help requests. If there is a help request, the depot considers the robot's position and fuel level; as well as the target's position to find the most suitable rendezvous point. The depot then responds to the help request and informs it about the rendezvous point. To ensure that many depots don't respond to the same help request, the depots maintain a list of help requests that are being responded to, at that point in time. This is known as the *reactive closed list*. If the help request is not the reactive closed list, it is called an *admissible reactive request*. If there a number of help requests, the depot finds its 'depot_gain' for all help requests and chooses to respond to the one with the highest depot_gain. The depot_gain for a depot d responding to a help request h can be defined as the ratio of the gain of the help request to the distance between the depot and the rendezvous point. Thus,

$$depot_gain_{d,h} = \frac{h_{gain}}{dist(d, rendezvous\ point)}$$

The flowchart describing the algorithms for the robot and depot algorithm is shown in Figure 3.7. The robot's algorithm is in red, and the depot's algorithm is in blue.

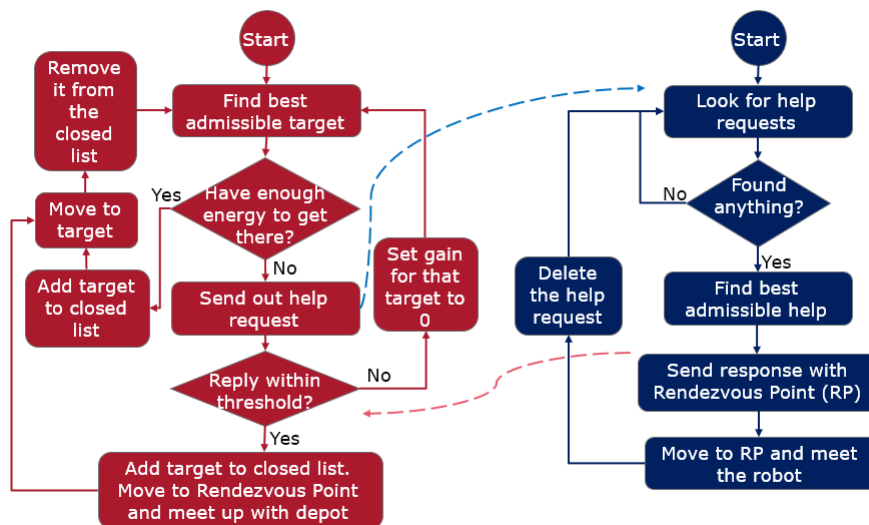


FIGURE 3.7: Flowchart for our decentralized control algorithm for persistent surveillance with energy constraints and moving depots responding to Reactive requests

The behaviour of the robots can be observed in Figure 3.8. The robot is represented by the yellow circle. Targets are white squares that become yellow after they are visited by the robot. The hollow squares are the depots. The red arrow indicated the direction of motion of the robot, and the blue arrow indicates the direction of motion of the depot. The images are arranged in a clockwise manner starting from the top-left image. In the second image the robot realizes that it does not have enough fuel to reach the target,

and it sends out a help request. The depot responds to the help request and starts moving towards the Rendezvous Point (RP). The robot and depot meet at this RP and the robot's batteries are replaced.

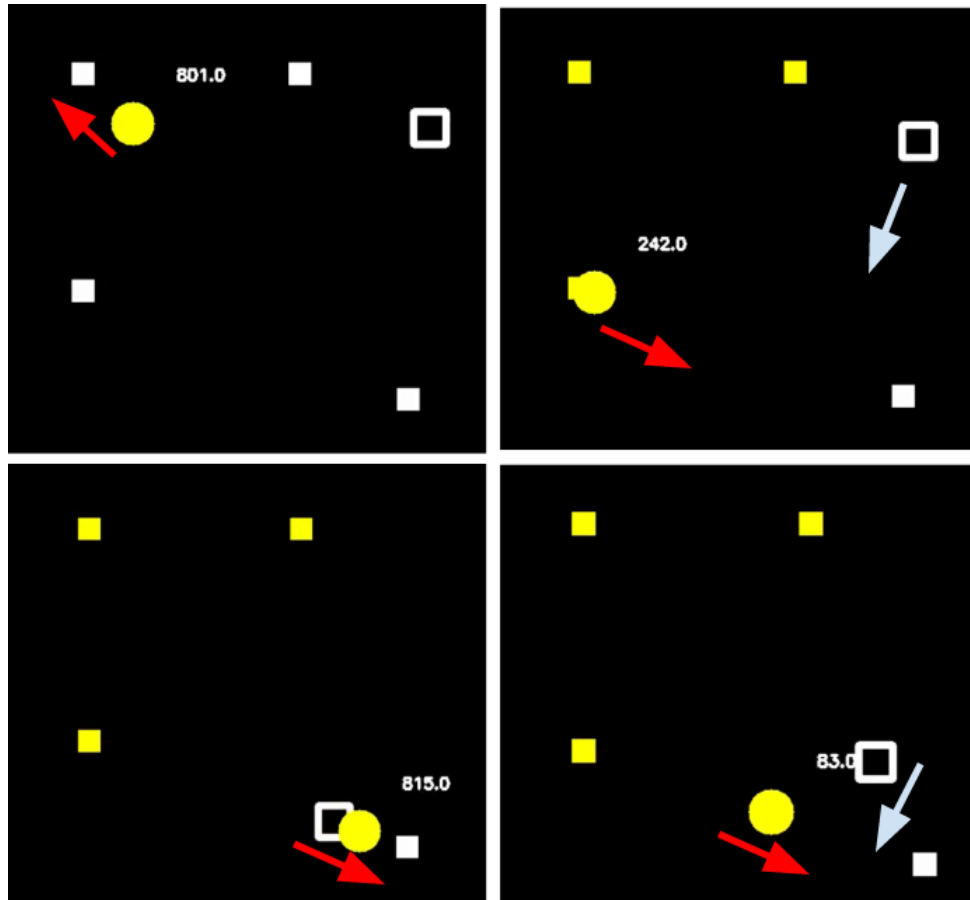


FIGURE 3.8: Robot and Depot behaviour considering Reactive Help Requests

3.4.2 Preemptive Help Requests

It is clear that the system would be more efficient if the depot starts moving towards the robot, or even replaces the robot's dying battery preemptively before the energy level falls to a point where the robot does not have enough energy to reach its next desired target. Thus, we introduced a new type of help request called 'preemptive requests'. The algorithm followed by each robot is shown in Algorithm 5.

Algorithm 5 Individual robot - Moving Depots - Preemptive Requests

```

while true do
  closed_list = []
  list_of_gains = []
  robot.moving = False
  for target in targets do
    | list_of_gains.append(target.reward/dist(robot,target))
  end
  while robot.moving = False do
    target = find_best_admissible_target(list_of_gains)
    if robot.energy_level >= dist(robot,target) then
      | if robot.energy_level >= dist(robot,target) + fuel_threshold then
        | robot.moving = True
        | robot.move(target)
      | else
        | publish preemptive help request
        | robot.moving = True
        | robot.move(target)
      | end
    | else
      | publish reactive help request
      | if depot reply with Rendezvous Point (RP) within time limit then
        | robot.moving = True
        | robot.move(RP)
        | robot.move(target)
      | else
        | list_of_gains[target] = 0
      | end
    | end
    robot.moving = True
    robot.move(target)
  end
end

```

When the robot starts moving from one target to another, it calculates what its energy level is going to be once it gets there. If this value is below a predefined threshold (which is a function of the robot's energy level at full charge), the robot sends out a preemptive request. One of the major differences between the 2 types of help requests is that, when the robot sends out a preemptive request, it does not have to wait to receive conformation that a depot is coming to its aid. It just sends out the request and moves

towards its target. A depot might meet up with the robot while it is on its way to the target. If this happens, the depot deletes the preemptive help request. On the other hand, if this does not happen, the robot deletes the preemptive request when it reaches the target. If the robot does not have enough energy to reach its next desired target, it then continues in the same manner as described in the Section 3.4.1.

On the depot side of things, the depots are now constantly on the lookout for both, reactive, and preemptive requests. The priority is given to reactive requests i.e. the depots look for preemptive requests only if there are no reactive requests. If there are reactive requests, the depots ignore the preemptive requests and continue in the same manner as described in the Section 3.4.1. If there are no reactive requests, and the depot responds to the preemptive request. It evaluates whether or not it can meet up with the the robot without disturbing the robot's trajectory (i.e. without having the robot wait). If so, the depot intercepts the robot and replaces the robot's battery with a new one. The robot does not need to be told about the rendezvous point since does not have to make any changes to its trajectory. If the depot cannot meet up with the the robot without disturbing the robot's trajectory, it just moves towards the target that the robot is heading towards. This way the depot is closer to the robot once it sends out a reactive request. If there are numerous preemptive requests the depot finds its own gain ratio before choosing which robot to help. The depot gain ratio is calculated by taking a ratio of the gain that the robot calculated, to the distance that the depot would have to travel to help out the robot. As described in the previous section, once a depot decides to help a robot, the associated help request is not visible to other depots anymore. This ensures that 2 depots don't respond to the same help request. If a depot is responding to a preemptive request, but the robot that sent out the preemptive request reaches the target, the depot is stopped, and the request is deleted. The flowchart describing the algorithms for the robot and depot algorithm is shown in Figure 3.9. The robot's algorithm is in red, and the depot's algorithm is in blue.

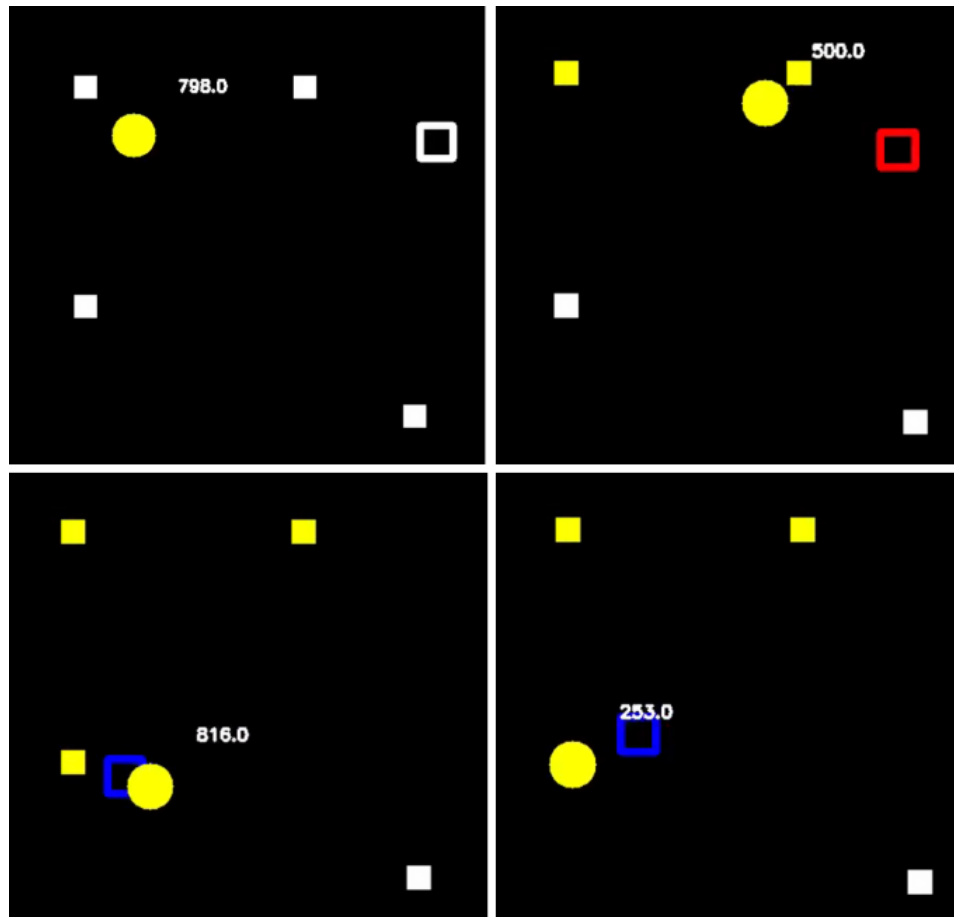


FIGURE 3.10: Robot and Depot behaviour considering Preemptive and Reactive Help Requests

Chapter 4

Results

4.1 Performance Evaluation Metric

In order to evaluate the effectiveness of different persistent surveillance algorithms, it is important to establish an evaluation metric. The ‘idleness’ of a target is the average time between successive visits at that target. Thus, if a target t has been visited at times t_1, t_2, t_3 and t_4 , the idleness of t is given by i_t where:

$$i_t = \frac{(t_2 - t_1) + (t_3 - t_2) + (t_4 - t_3)}{3}$$

‘Graph idleness’ is the average value of the idleness of all targets, and this is selected as the performance metric. Thus, the graph idleness for a system for 4 targets that have an idleness of i_1, i_2, i_3 and i_4 , the graph idleness g is given by:

$$g = \frac{i_1 + i_2 + i_3 + i_4}{4}$$

Intuitively, the lower the value of the graph idleness, the better is the performance of the algorithm.

It is important to note that, since the robots and (moving) depots start at random locations, the system requires some time to ‘stabilize’, before it can be evaluated. The system is considered to be stable when the graph idleness changes by less than 5% between successive evaluations. Once the system is stable we wait for each target to be visited at least 4 times before the algorithm is terminated. The time between stabilization and termination is used for the performance evaluation.

4.2 Experimental Setup

We have considered a number of scenarios while evaluating the performance of the algorithms. The factors that can be varied are: number of targets, number of robots, number of depots, and the density of targets in the world. Since the world is assumed to be square shaped, the values of the number of targets and the density of targets determine the size of the world. It is assumed that the energy level of the robot at full charge is enough to let it travel a distance equal to the length of the diagonal of the world. The location of the robots, targets, and depots is assumed to be uniformly distributed at the start. We decided to have the number of robots and targets be a multiple of the number of depots. We chose the options for the number of depots to be 4 and 8. Thus, the options for the number of robots was chosen to be 8, 16 and 32; and the options for the number of targets was chosen to be 32 and 64. The density values were chosen to be 0.1 and 0.02. A *setup* is a selection of number of depots, number of robots, number of targets and density.

$$Setup = [number\ of\ depots, number\ of\ robots, number\ of\ targets, density]$$

It is intuitively understood that the graph idleness of the system reduces if the number of robots and depots, and the density is increased, and the number of targets is the decreased. The ‘best case setup’ that we have considered is:

$$Best_Case_Setup = [8, 16, 32, 0.1]$$

Similarly, the graph idleness of the system increases if the number of robots and depots, and the density is decreased, and the number of targets is the increased. The ‘worst case setup’ that we have considered is:

$$Worst_Case_Setup = [4, 8, 32, 0.02]$$

The best and worst case setups are used for algorithm analysis. We have also considered two ‘moderate’ cases viz. [4,16,64,0.1] and [4,8,64,0.1].

The algorithms developed were tested in a simulator that was developed using the OpenCV libraries [48] in Python. To be able to study any patterns that emerge, each robot is given a unique color. Each target is white to start with, but when it is visited

by a robot, its color changes to match the color of the robot that visited it most recently. As described in Chapter 3, for the third stage of the problem, the depots respond to preemptive requests and reactive requests. In that case, the depot's color is red when it is responding to preemptive requests and blue when it is responding to reactive requests.

4.3 Results

This section presents the results for different setups. The Y axis of the graphs represent the 'Graph Idleness'. On the X axis, '1' represents the results for the static depot case. '2' represents results for moving depots without preemptive requests and '3' represents results for moving depots with preemptive requests. Each of the algorithms, has been tested with 10 random seeds.

For the analysis we start by considering the best case setup i.e. $[8,16,32,0.1]$, and we look at incrementally worse cases till we reach the worst case setup. The results for the best case setup are shown in Figure 4.1. The graph idleness appears to be similar for all 3 cases. This is because the world size is so small, and the number of robots, and number of depots is so large that the effect of the moving depots cannot be seen. The mean appears to be lower for the static depot case because the outliers are not accounted for in the calculation of the mean.

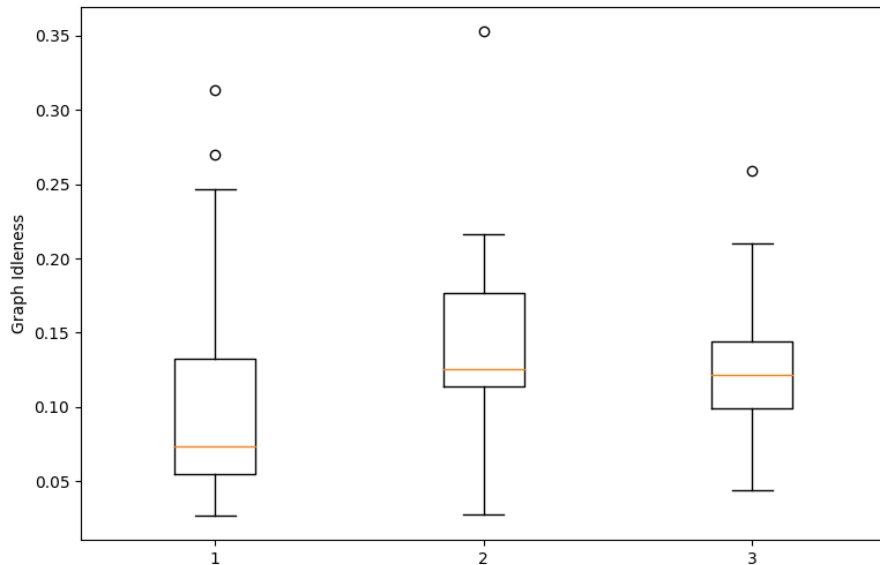


FIGURE 4.1: Results for $[8,16,32,0.1]$

We then look at the setups $[4,16,64,0.1]$ and $[4,8,64,0.1]$, for which the results are shown in Figures 4.2 and 4.3 respectively. There are numerous outliers for the static depot case. If these were taken into account while plotting the box plots, the mean for the static

depot case would be higher. Unfortunately, the difference between the 3 algorithms is not evident for these setups. We believe this may be due to the fact that 10 random seeds is not enough to see the results emerge.

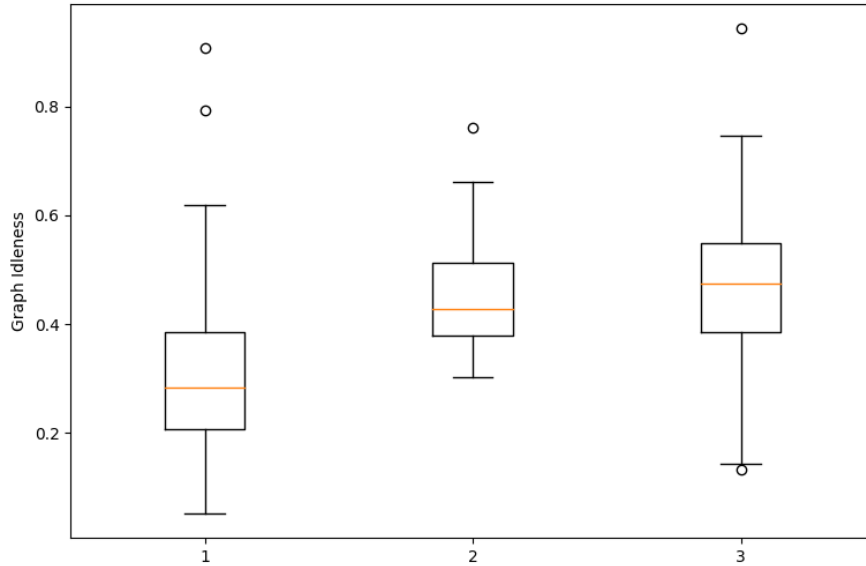


FIGURE 4.2: Results for [4,16,64,0.1]

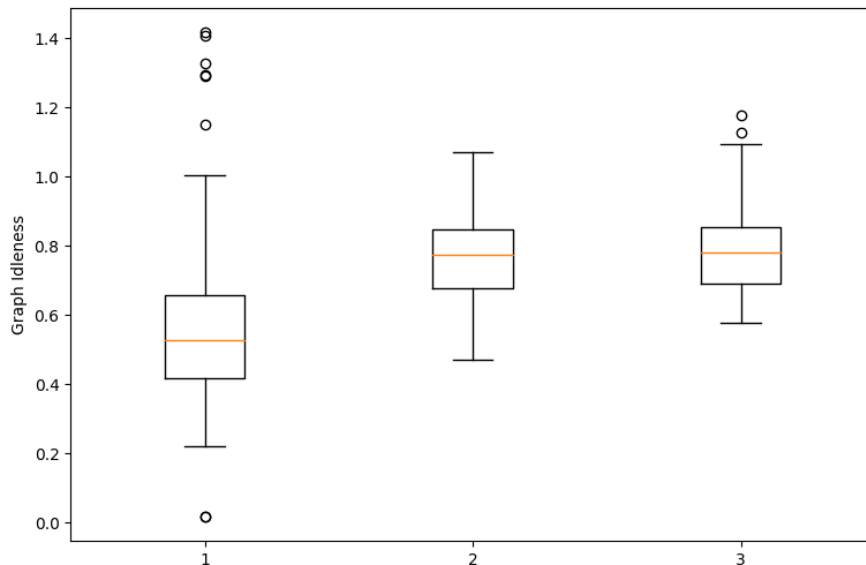


FIGURE 4.3: Results for [4,8,64,0.1]

Lastly, we look at the worst case setup i.e. [4,8,32,0.02]. Unfortunately for this case, the optimizer took too long to solve for optimal location of the depots. Considering suboptimal locations would skew the results, and thus, for this setup we do not consider the algorithm with the static depots. The results for this setup are shown below in Figure 4.4. On the X axis, '1' represents the results for moving depots without preemptive

requests and ‘2’ represents results for moving depots with preemptive requests. It is very clear from the image that the algorithm with the preemptive requests has a smaller value of graph idleness.

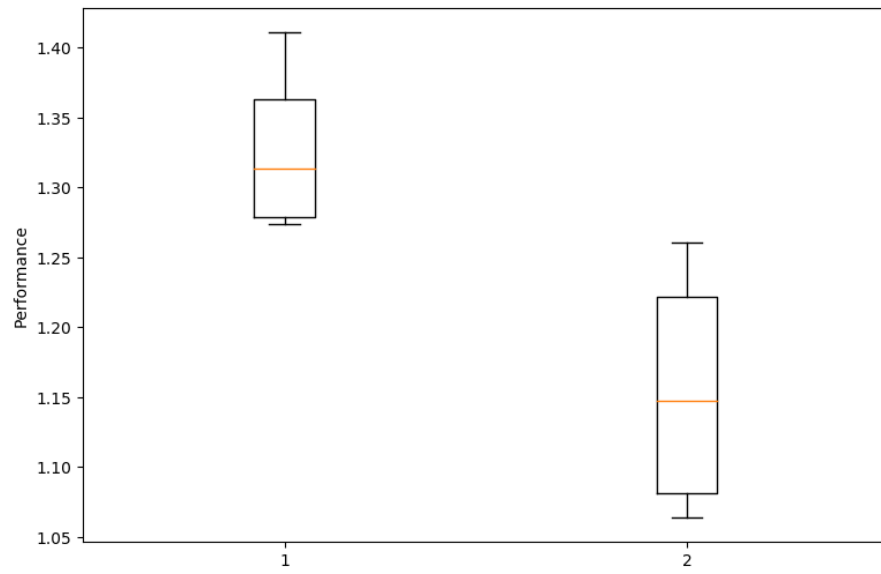


FIGURE 4.4: Results for $[4,8,32,0.02]$

Chapter 5

Conclusions And Future Work

5.1 Summary and Conclusion

This thesis presents decentralized solutions to 3 stages of the persistent surveillance problem. The persistent surveillance problem states that there are numerous points of interest (a.k.a. targets) in the environment that need to be visited as frequently as possible (for surveillance).

In the first stage, we consider the case where the robots being used for surveillance do not have energy constraints i.e. they have an unlimited battery life. The goal of the algorithm is to help the robots decide which target to visit next. This is done by incentivizing the robots to visit the targets by assigning a reward to each target which is a function of the amount of time since the target was last visited. The robots evaluate their gain at each target, which is a ratio of the reward at the target to the distance that the robot would have to travel to reach the target. Before moving to the target with the best gain, a robot makes sure that none of the other robots in the swarm are heading to the same target.

In the second stage, we present a decentralized solution to the persistent surveillance problem while considering the energy constraints of the robots. To the best of our knowledge, there is no work done in this field, and our solution is the first. Since the robots have a limited battery life, they must frequently visit depots which are automated battery replacement stations. Since the position of the targets and number of depots is known beforehand, we use integer linear programming to find the optimal location of the depots. Our algorithm then helps the robots decide which target to visit next, and when it needs to visit a depot. To ensure that the robot does not become stranded, before it starts moving towards a target, it checks to ensure that it has enough energy to first move to that target, and then move to the depot closest to that target.

In the third stage we present 2 algorithms that consider moving depots instead of the static depots presented in the previous stage. The advantage of having moving depots is that the robot does not spend time travelling to a depot, and eliminating the need to take this detour improves the performance. We found few algorithms that use moving depots and none of these algorithms are decentralized or online. Our work presents the first decentralized algorithms that considers moving depots for replenishing/recharging batteries of robots being used for surveillance. We have presented 2 algorithms, and the difference between them is the set of circumstances under which a robot contacts the depots for help. In the first approach a robot sends out a help request when it does not have enough energy to reach its intended target. In the second approach the robot sends out a help request before it finds itself in a position where it does not have enough energy to reach its intended target.

In order to study the behaviors that emerge due to these algorithms, we developed a simulator using the OpenCV libraries in Python.

The performance metric used to evaluate the algorithms is graph idleness, which is the average of idleness at all targets. The idleness at a target is the average time between visits at each target.

We have compared the performance of the algorithm presented in the second stage and the two algorithms presented in the third stage. We ran a number of experiments that prove that having moving depots result in a graph idleness that is less than or equal to the graph idleness obtained for the same setup with optimally placed static depots. The results also prove that, if the robots send out a preemptive help request; the graph idleness is less than or equal to the case where they send out only reactive help request. The results also prove that the difference in the results becomes more profound as the setup becomes increasingly ‘worse’.

5.2 Future Work

This is a very exciting and unexplored area in the field of swarm robotics, and there are a number of directions in which this project could be taken forward. To start with, the algorithms presented in this paper should be implemented in the ARGoS simulator [49]. The simulator we developed was useful to study the behavior of the swarm, but since we have a thread for each robot and depot, the simulator does not perform well if the number of robots exceeds 250. ARGoS has been designed specifically to deal with swarms, and it will be able to handle a large number of robots. Once the algorithms are implemented in ARGoS, they should be implemented on real robots.

Another interesting approach could be to change the manner in which the robots choose the target to visit next. In our algorithm, when a robot chooses a target to visit, the target is added to the closed list. However, there could be a other robots that were better placed to visit that target. Thus, the robots should be able to communicate with each other, to find the best robot that should visit a target. Similarly, in the case of moving depots, the depots should be able to communicate with each other to find the best depot to responds to a help request.

An extension of this problem is to consider the case where the depots themselves have energy constraints, and can carry a limited number of ‘replacement batteries’ for the robots.

Another interesting avenue that we explored, but needs more work, is the ‘Diversity Metric’

5.2.1 Diversity Metric

In most centralized approaches that solve the persistent surveillance problem, the optimal solution is calculated by modelling the problem as an extension of the Travelling Salesman Problem (TSP) and solving it using MILP. The solution shows that it is optimal to assign a set of targets to a robot, and have the robots visit these targets in a cyclic manner.

It can be assumed that the performance of our algorithm would be closer to optimal if the system evolved in such a way that each robot goes exclusively to a few targets, and every target is visited exclusively by a single robots. In other words we want the algorithm to result in a self organizing behavior. However, to the best of our knowledge, there exists no metric to measure the level of self organization.

Thus, we tried to introduce a measure of self organization for the persistent surveillance problem. We started by evaluating the ratio of the number of different robots that have visited a target to the total number of robots i.e. if the total number of robots is 10, and the number of different robots that visited the target is 5, the robot diversity ratio for that target is 0.5. For a target T :

$$T_{robot_diversity_ratio} = \frac{\text{number of robots that visited } T}{\text{total number of robots}} \quad (5.1)$$

Thus, the best possible robot diversity ratio for a target in a 10 robot system is 0.1, but the best possible robot diversity ratio for a target in a 5 robot system is 0.2. Thus, this

metric is not ideal since the value of the best performance changes with the number of robots in the system.

This problem can be solved by subtracting the best possible robot diversity ratio from the robot diversity ratio for a target. We called the measure of ‘*diversity*’ at a target. For a target T :

$$T_{diversity} = \frac{(\text{number of robots that visited } T) - 1}{\text{total number of robots}} \quad (5.2)$$

Thus, the best possible diversity for a system with any number of robots is always 0. The ‘*diversity measure*’ of the algorithm is the average of the diversities at all the targets. Thus, for a system with N targets, the diversity measure is:

$$\text{diversity_measure} = \frac{1}{N} \sum_{i=1}^N T_{i_diversity} \quad (5.3)$$

Intuitively it can be expected that the graph idleness would reduce as the diversity reduces. We experimented with 300 random seeds for the case where there are 16 robots and 64 targets; and the robots have no energy constraints. We got the graph shown in Figure 5.1.

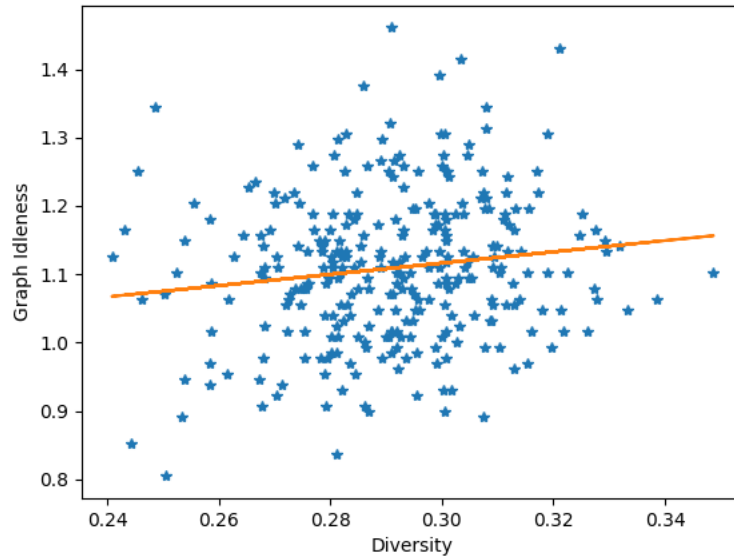


FIGURE 5.1: Graph Idleness vs Diversity

Although the slope of the best fit line is not very steep, it does show a positive slope. And this aligns with our prediction. However, we believe that this must be investigated for different setups before a conclusion can be drawn. The *diversity measure* is not the

performance parameter that is used to evaluate the algorithm, but it is a parameter that reflects how much the algorithm can be improved. If the graph idleness of the algorithm has to be reduced, one can try to reduce the diversity_measure.

Bibliography

- [1] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.
- [2] Francis Heylighen. Stigmergy as a universal coordination mechanism i: Definition and components. *Cognitive Systems Research*, 38:4–13, 2016.
- [3] Tommaso Francesco Villa, et al. An overview of small unmanned aerial vehicles for air quality measurements: Present applications and future perspectives. *Sensors* 16.7, 2016.
- [4] Randal W. Beard Kingston, Derek and Ryan S. Holt. Decentralized perimeter surveillance using a team of uavs. *IEEE Transactions on Robotics* 24.6, pages 1394–1404, 2008.
- [5] Michael Burri, et al. Aerial service robots for visual inspection of thermal power plant boiler systems. *Applied Robotics for the Power Industry (CARPI)*, 2012.
- [6] Anuj Puri. A survey of unmanned aerial vehicles (uav) for traffic surveillance. *Department of computer science and engineering, University of South Florida (2005)*.
- [7] Ivn Maza, et al. Experimental results in multi-uav coordination for disaster management and civil security applications. *Journal of intelligent robotic systems*, 2011.
- [8] Cyrus Farivar. Persistent surveillance systems has been watching baltimore for months. *Arstechnica*, 2016.
- [9] Jorge Cortes, et al. Coverage control for mobile sensing networks. *IEEE Transactions on robotics and Automation*, pages 243–255, 2004.
- [10] Dusan Stipanovic Hokayem, Peter F. and Mark W. Spong. On persistent coverage control. *IEEE Conference on Decision and Control*, pages 243–255, 2007.

- [11] Stephen L. Smith and Daniela Rus. Multi-robot monitoring in dynamic environments with guaranteed currency of observations. *IEEE Conference on Decision and Control*, 2010.
- [12] Gregory Gutin and Abraham P. Punnen. The traveling salesman problem and its variations. *Springer Science & Business Media*, 12, 2006.
- [13] Brian Paul Gerkey. On multi-robot task allocation. *Diss. University of Southern California*, 2003.
- [14] Tillerson M. Richards A. Bellingham, J. and J. How. Multi-task allocation and path planning for cooperating uavs. *Proceedings of Conference of Cooperative Control and Optimization*, 2001.
- [15] Minai A. Jin, Y. and M. Polycarpou. Cooperative real-time search and task allocation in uav teams. *Proceedings of the IEEE Conference on Decision and Control*, 2003.
- [16] Pollini L. Turra, D. and M. Innocenti. Fast unmanned vehicles task allocation with moving targets. *Proceedings of the IEEE Conference on Decision and Control*, 2004.
- [17] Aydano Machado, et al. Multi-agent patrolling: An empirical analysis of alternative architectures. *Multi-Agent-Based Simulation II*, 61:81–97, 2003.
- [18] David Portugal and Rui P Rocha. Multi-robot patrolling algorithms: examining performance and scalability. *Advanced Robotics*, 27(5):325–336, 2013.
- [19] Dana Angluin and Leslie G Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and system Sciences*, 18(2):155–193, 1979.
- [20] David Portugal and Rui Rocha. Distributed multi-robot patrol : A scalable fault-tolerant framework. *Robotics and Autonomous Systems*, 61:1572–1587, 2013.
- [21] Hoang Nam Chu, et al. Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation. *International Conference Tools with Artificial Intelligence*, 2007.
- [22] Camille Besse Marier, Jean-Samuel and Brahim Chaib-draa. Solving the continuous time multiagent patrol problem. *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.
- [23] Hugo Santana, et al. Multi-agent patrolling with reinforcement learning. *International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004.

- [24] Xuchao Lin, Christos G. Cassandras, and Xuchu Ding. An optimal control approach to the multi-agent persistent monitoring problem. *IEEE Transactions on Automatic Control*, pages 947–961, 2013.
- [25] David Portugal and Rui Rocha. A survey on multi-robot patrolling algorithms. *Doctoral Conference on Computing, Electrical and Industrial Systems, Berlin*, 2011.
- [26] How J.P. Dale, D. Automated ground maintenance and health management for autonomous unmanned aerial vehicles. *Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, - Master's Thesis*, 2007.
- [27] Yash Mulgaonkar and Vijay Kumar. Autonomous charging to enable long-endurance missions for small aerial robots. *SPIE Defense+ Security*, pages 90831S–90831S–15, 2014.
- [28] Björn Sjdahl. Autonomous recharging for swarms of flying robots. *Luleå University of Technology - Master's Thesis*, 2014.
- [29] Roomba vacuum robot from irobot corporation. <http://www.irobot.com/>.
- [30] Johnhenri R. Richardson, Jonathan D. White, Zahid Hasan, Elizabeth Qian, Kurt A. Swieringa, Clarence B. Hanson, and Anouck Girard. Autonomous battery swapping system for small-scale helicopters. *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.
- [31] Paulo Kemper Filho, Suzuki, Koji AO, and James R. Morrison. Automatic battery replacement system for uavs: Analysis and design. *Journal of Intelligent & Robotic Systems*, 65.1:563–586, 2012.
- [32] Matthew Michini, Bernard Michini, Jonathan P. How, Tuna Toksoz, Joshua Redding. Automated battery swap and recharge to enable persistent uav missions. *AIAA Infotech@ Aerospace Conference*, 2011.
- [33] Suzuki K.A.O. & Morrison J.R. J Kemper, F.P. Uav consumable replenishment: Design concepts for automated service stations. *Intell Robot Syst*, 65.1:563–586, 2011.
- [34] Zvi Drezner and Horst W. Hamacher. Facility location. *Springer-Verlag*, 1995.
- [35] Yanhai Xiong, et al. Optimal electric vehicle charging station placement. *IJCAI*, 2015.
- [36] Alex Couture-Beil and Richard T. Vaughan. Adaptive mobile charging stations for multi-robot systems. *Intelligent Robots and Systems*, 2009.

- [37] Ethan Stump and Nathan Michael. Multi-robot persistent surveillance planning as a vehicle routing problem. *IEEE Conference on Automation Science and Engineering (CASE)*, 2011.
- [38] Ethan Stump Michael, Nathan and Kartik Mohta. Persistent surveillance with a team of mavs. *IEEE Intelligent Robots and Systems (IROS)*, 2011.
- [39] Victor Pillac, et al. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, pages 1-11, 2013.
- [40] Mixed integer programming basics. <http://www.gurobi.com/resources/getting-started/mip-basics>.
- [41] Derek Mitchell, et al. Multirobot long-term persistent coverage with fuel constrained robots. *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.
- [42] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. 1976.
- [43] Kaarthik Sundar and Sivakumar Rathinam. Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots. *IEEE Transactions on Automation Science and Engineering*, 11.1, 2014.
- [44] Kaarthik Sundar David Levy and Sivakumar Rathinam. Heuristics for routing heterogeneous unmanned vehicles with fuel constraints. *Mathematical Problems in Engineering*, 2014.
- [45] Neil Mathew, Stephen L Smith, and Steven L Waslander. A graph-based approach to multi-robot rendezvous for recharging in persistent tasks. pages 3497–3502, 2013.
- [46] Neil Mathew, Stephen L Smith, and Steven L Waslander. Multirobot rendezvous planning for recharging in persistent tasks. *IEEE Transactions on Robotics*, 31: 128–142, 2015.
- [47] Parikshit Maini and P. B. Sujit. On cooperation between a fuel constrained uav and a refueling ugv for large scale mapping applications. *Unmanned Aircraft Systems (ICUAS)*, 2015.
- [48] G. Bradski. *Dr. Dobb's Journal of Software Tools*, 2000.
- [49] Carlo Pinciroli, Vito Trianni, Rehan OGrady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, et al. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 6(4):271–295, 2012.