Rapidly Estimating Swarm Resource Needs Through Autonomous Simulation

by

Eric Young

B.S. Computer Science, United States Naval Academy (2015)
B.S. Information Technology, United States Naval Academy (2015)

Submitted to the Department of Mechanical Engineering and the
System Design & Management Program
in partial fulfillment of the requirements for the degrees of

Naval Engineer

and

Master of Science in Engineering Management

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Author: Eric Young
Department of Mechanical Engineering and the System Design & Management Program
May 12, 2023

Certified by: Dr. Michael Benjamin
Principal Research Scientist in Mechanical Engineering, MIT

Approved by: Dr. Nicolas G. Hadjiconstantinou
Chairman, Department Committee on Graduate Theses

Approved by: Dr Joan S. Rubin
Executive Director, System Design & Management

Rapidly Estimating Swarm Resource Needs Through Autonomous Simulation

by

Eric Young

Submitted to the Department of Mechanical Engineering and the System Design & Management Program
on May 12, 2023, in partial fulfillment of the
requirements for the degrees of
Naval Engineer
and
Master of Science in Engineering Management

Abstract

The maritime industry spends significant time and resources accomplishing long lasting collaborative tasks such as search and rescue or ocean surveying. Autonomous swarm ships' ability to scale rapidly and operate with limited resources allows them to outperform conventional crewed ships at these collaborative operations. Despite their incredible potential, perpetually operating productive autonomous swarms creates significant logistic challenges. This thesis aims to solve these problems. Specifically, this thesis aims to maximize collaborative swarm productivity, by predicting and managing robot resource needs, using operations theory, simulation, and machine learning.

Maximizing swarm productivity first requires developing a common scenario to measure productivity. Drawing from multi-robot patrol research, this thesis implements two resource-aware multi-robot patrol missions in MOOS-IvP. In each mission, vehicles perpetually patrol a grid and must periodically break patrol formation to refuel at a depot. Missions measure their performance based on how frequently robots visit each portion of the mission operating area (grid idle time) and how much area each robot controls (average Voronoi polygon area). With a common patrol scenario developed, this thesis then simulates patrol missions using different vehicle and depot parameters to generate a broad performance dataset.

Finally, this thesis develops a method to predict future mission performance from the simulated productivity dataset. Simulated mission data is post processed and used to train XGBoost models. Compared to mission simulations, these models take far less time to produce while still showing planners what performance and vehicle output they can expect from a given mission.

Thesis Supervisor: Dr. Michael Benjamin
Title: Principal Research Scientist in Mechanical Engineering, MIT

Acknowledgments

I could not have done this thesis without the help and support of my friends and colleagues at MIT. I cannot believe how fortunate I was to meet and interact with so many *exceptionally* talented humans during my three years here. I will never forget my time with you all.

To my advisor Dr. Mike Benjamin, thank you so much for guiding me through this research, helping me learn marine autonomy, and helping me push the boundaries of collaborative vehicle autonomy. Despite teaching a class, running a lab, and maintaining a major open source software project, you accepted me into the Pavlab and helped me grow into a better human. Your work, humility, and breathtaking talent inspire me. From your help, I leave MIT better prepared to usher the Navy into a new autonomous era.

To Dr. Sean Willems, thank you for being a truly unbelievable mentor and operations management resource. From my simple email titled "Autonomous Ship Supply Chain Questions" you helped me focus my thesis topic, develop analytical techniques to analyze autonomous ship performance, and repeatedly took time to meet and offer constructive feedback on my [often crazy] ideas. From your help, I leave MIT better prepared to define and solve complex operational tasks.

To Randy, Sterling, Brian, and Nate, thank you all for recommending me for admission at MIT. Your recommendations kicked off a truly unbelievable opportunity that changed my life for the better. I intend to pay these recommendations forward in the future.

To my family, thank you for your continued love, sacrifice and support. I love you all so much.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Collaborative and Perpetual Operations in the Maritime Industry

Maritime operations incentivize collaboration. Specifically, the maritime industry spends significant energy accomplishing collaborative and perpetual operations. In this work, collaborative operations occur when commodity ships work together to achieve a common goal. Perpetual operations occur when a ship must operate longer than its normal operating period to achieve a task. In other words, a ship must return to port at least once during its tasking to address human and supply needs. Examples of collaborative, perpetual operations in the maritime industry include the following:

- Search and Rescue
- Hydrographic Mapping
- Patrol and Adversarial Search
- Convoy Escort
- Area Coverage Operations (e.g Smart Buoys)

These operations require substantial human and maritime resources. For example, in 2013 alone, the US coast guard completed 17,826 search and rescue operations

that saved 3,773 lives. Saving these lives required Coast Guard assets perform 53,911 hours of search and rescue operations[Tra]. While the Coast Guard does not detail their total Search and Rescue costs, they publish how much each of their assets cost to assist federal and state governments. Utilizing a single Fast Response Coast Guard Cutter costs government agencies $4,803 dollars per hour in 2021. Aircraft and special purpose craft can cost double or even triple a cutter's hourly rate[Guaa]. Consequently, Search and rescue likely costs the government hundreds of millions each year. Search and rescue aside, hydrographic surveys also require significant economic investment. For example, in 2008 alone, hydrographic survey organizations employed over 8,400 people around the world to conduct an estimated $1.7 billion in offshore, port, and harbor surveys[Joh]. Improving performance in these operations potentially saves millions of dollars, strengthens our knowledge of the ocean, and saves human lives.

1.2 Performance Benefits Created When Autonomous Ship Swarms Perform Collaborative Maritime Operations

Autonomous ships' unique architectural advantages allow them to outperform crewed ships at collaborative and perpetual tasks. Specifically, autonomous drone ships can scale rapidly and require fewer resources than crewed ships.

1.2.1 Autonomous Scaling Capability and its Benefits

Autonomous ships can easily scale in quantity to tackle collaborative tasks in parallel. Purpose-built autonomous ships require no galley, berthing, pilot house, or any other crew space. Further, they require no waste management, roll control, nor other crew comfort systems. Without a crew, ship builders can design vessels containing only systems required to accomplish a mission. This design shift reduces ship complexity, enhances production capacity, and reduces overall ship cost.

Design choices aside, operating autonomous ships eliminates the need to find, train, and retain capable mariners. Operating large ships in harbors and the open ocean requires capable mariners to follow Rules of the Nautical Road, respond to shipboard casualties, and navigate across large distances. Depending on the size of the ship in operation, obtaining a license to operate these ships can take years of training[Guab]. These significant training requirements and the diversity of vessels restrict the mariner labor pool's size and flexibility. In other words, growing crewed ship capacity requires significant human capital considerations. By contrast, autonomous ship operators can update training, scheduling, and operational guidance to their entire fleet using messaging and over the air software updates. This capability dramatically simplifies the time required to add new autonomous ships to a fleet.

Autonomous ship scaling capability yields tremendous benefit in collaborative, perpetual tasks. Collaborative tasks generally involve multiple ships performing their own separate negotiated tasks to solve a large problem. These individual tasks can also generally be performed in parallel. Further, each problem's perpetual nature guarantees there will be an abundance of tasks available for each ship to do. For example, conducting ocean surveys generally involves multiple ships driving patterns in the ocean to measure ocean floor characteristics with a sensor. Harbor patrol requires ships repeatedly visit various spots to maintain harbor safety. Individual ships can take surveys and visit patrol areas without affecting the quality of other ships' operations, provided they know basic information like previously surveyed areas or patrol visit frequency. Consequently, adding more ships to accomplish these simple, perpetual, and parallel tasks improves global problem performance. In other words, adding ships to an operation means more patrol visits, quicker ocean surveys, and more lives rescued.

1.2.2 Autonomous Ship Reduced Resource Benefits

Individual autonomous ships require far fewer resources and support than crewed vessels. From a short-term perspective, crews require food, water, medicine, and space

while at sea. Further, ships always need a contingent of humans awake to navigate and maintain the ship. This requirement forces large and small ships to operate several shifts to sustain basic operations at sea. Sustaining several shifts at sea requires significant energy to run crew support systems. Even if ships can replenish these resources at sea, their crews cannot mentally remain at sea indefinitely.

By contrast, autonomous ships consume fuel only to power systems vital to accomplishing their goals. Further, with no crew space requirements, ships can be built smaller and thus require less fuel to move through the water. To summarize, assuming long maintenance windows, only fuel limits the time an autonomous ship can remain at sea.

1.3 New Challenges Caused by Autonomous Ship Swarm Operation

Despite their incredible potential, perpetually operating large-scale autonomous swarms creates significant logistic challenges. As previously discussed, an autonomous ship's fuel range limits their max operational time at sea. This limit means autonomous ships must interrupt their current tasking to restock at a fuel depot when performing perpetual operations. This restocking operation creates two problems:

1.3.1 Problem 1: Nonlinear Fuel Consumption

Sea state, wind, tasking adjustments, and collision avoidance maneuvers make ships consume fuel at nonlinear rates. Consequently, if ships can restock whenever they want, large groups of ships may leave the work area en masse to restock. This mass movement may result in incomplete tasking and significant fuel queueing at fuel depots. Human schedulers may be able to sequence and deconflict several ships restocking manually. However, manually coordinating restocking operations for dozens or hundreds of ships becomes rapidly unsustainable. In other words, organizations

cannot effectively implement autonomous swarms in perpetual operations without managing how drones restock at sea.

1.3.2 Problem 2: Difficult Operational Planning

Operational planners cannot easily predict what infrastructure they need to maintain perpetual swarm operations. Any organization operating a large-scale swarm of ships at sea needs to answer the following questions:

- How many drones do you plan to deploy?
- What geometry does the swarm operate in?
- Where is the farthest place their algorithms will take them?
- How many fuel depots do you have to restock these ships?
- Where are the depots located?
- Can the depots move?
- How does depot location affect mission performance?
- Who controls the scheme dictating when ships refuel?
- How does this refueling scheme affect overall mission performance?
- Can the ships run out of fuel?
- What happens if the ships run out of fuel?
- Can ships communicate their fuel status to each other?
- How do limited or degraded communication capabilities impact mission performance?

Currently, many of these questions, especially those related to mission performance, cannot easily be answered without historical data or time-consuming simulations. Time-sensitive operations like search and rescue missions cannot afford to waste hours estimating the resources needed to maintain sea swarms. Consequently, organizations cannot effectively implement autonomous swarms in perpetual operations without rapid tools to estimate swarm resource needs.

1.4 Thesis Objective

This thesis aims to maximize collaborative swarm productivity, by predicting and managing robot resource needs using operations theory, simulation, and machine learning.

1.5 Thesis Contribution to Marine Robotics

This thesis performs the following:

- Implements and simulates low communication resource-aware patrol algorithms using many ships operating in a constrained marine environment.
- Tests how dynamic robot behavior impacts naïve resource management strategies.
- Develops a novel way to predict future swarm resource needs based on machine learning and past mission simulation.

1.6 Thesis Structure

While many organizations invest significant effort to develop uncrewed or autonomous marine vehicles, no organization publicly deploys and operates marine vehicles in long-term collaborative swarm scenarios. Consequently, this thesis uses simulation functions within the marine autonomy software MOOS-IvP to create and test

collaborative swarm scenarios. Maximizing collaborative swarm productivity requires answering the following questions:

1. How do you develop a common scenario to measure robot productivity?

2. How do you develop general-purpose resource management algorithms to resupply robots in the common scenario, and how well do these algorithms perform?

3. Given a common productivity scenario and appropriate resource management algorithms, how can you predict what supply resources a future mission planner might need?

Sections two through four will answer these questions respectively. These sections will first discuss how others have viewed and analyzed each question. The sections will then briefly discuss novel methods to answer the question. Finally, these sections will discuss the selected method, its implementation, and how that implementation contributed to the goals of the thesis. Section five concludes the thesis and recommends areas for further study.

Chapter 2

Developing a Common Scenario to Measure Swarm Productivity

Developing a common scenario to measure swarm productivity requires analyzing what common traits maximize robot collaborative task performance. These traits can then illustrate how to measure productivity in our scenario. Finally, a common robot task can be created to cause robot movements that amplify each measurable trait.

2.1 Common Trait 1: Spreading

In most collaborative tasks, robots maximize job performance when they spread themselves across an operating area. Specifically, robot spread enhances task performance in the following ways:

- Bathymetric Surveys: Robot spread minimizes the overlapping map data captured by each robots sensor and thus minimizes redundant survey work.
- Sonar Search: Ocean characteristics distort acoustic propagation. Spreading sensors around an area maximizes the likelihood a nearby sound will be heard.
- Weather Observation: Spreading sensors around an area maximizes the likelihood of identifying changing weather gradients.

- Search And Rescue: Many search and rescue tools (e.g visual search) cannot search beyond the distance to the horizon caused by the curvature of the earth. Spreading out sensors minimizes this blind spot. Further, robots that can automatically spread themselves out eliminate the need for a mother ship to deliver search and rescue sensors to prescribed locations.

2.2 Measuring Robot Spreading

Scientists frequently measure relative object placement and use these measurements to answer questions across many scientific disciplines. The following object observation methods were considered to analyze robot spreading across an operating area.

2.2.1 Nearest Neighbor Distance Anomaly Detection

Data scientists use nearest neighbor distances (k-distances) to identify patterns and detect anomalies in groups of data. Other researchers use similar k-distance metrics to identify clustering and dispersion patterns in geographic planning. While several methods exist to conduct k-distance anomaly testing, identifying anomalous clusters of robots would generally work as follows:

Algorithm 1 Nearest Neighbor Anomaly Detection

INPUT: Test Point P, Threshold Value anomaly_thresh
OUTPUT: Density Anomaly Likelihood

```
Select P's k nearest neighbors you want to analyze
for each neighbor ∈P's Nearest Neighbors do
    Calculate the Distance between P and neighbor
end for
Average all of P's neighbor distances as P_d
for all Pneighbor ∈P's Nearest Neighbors do
    Get the kNeighbors of Pneighbor
    for all kNeighbor ∈pNeighbors' Nearest Neighbors do
        Calculate the Distance between kNeighbor and pNeighbor
    end for
end for
Average all distances between kNeighbors and pNeighbors as K_d
if diff(K_d,P_d) > anomaly_thresh then
    return Density Anomaly Found
else
    return No Density Anomaly Found
end if
```

While k-distance based anomaly detection algorithms can be completed faster than polynomial time, they hold several properties that disqualify them from robot spreading analysis. First, while [Qu08] and [Sof] successfully implemented efficient k-distance density anomaly detection, [Sof] found that the algorithm routinely failed when their area exhibited collections of both extremely sparse and extremely dense data points. If implemented to identify poorly spread robots, this algorithm may falsely consider a large group of fueling robots to be not anomalous because a few well spread robots remain to patrol in the operating area. False positives aside, anomaly detection algorithms require significant environmental based tuning to output good results making them unfavorable to use in a robot system with dozens of possible scenario variables.

2.2.2 Voronoi Entropy

Scientists use Voronoi entropy in chemical structural analysis to determine how ordered given groups of objects are compared to their spatially random counterparts[Bor+18]. Unlike the k-distance evaluation, Voronoi entropy yields a single value and requires no k-variable selection. Voronoi entropy, outlined by [Bor+18], can be calculated for vehicles as follows:

Algorithm 2 Vehicle Voronoi Entropy Calculation

INPUT: Operating Area, Vehicle positions
OUTPUT: Operating Area Voronoi Entropy S

1: Create a Map(edgeNum,count) to record frequency of polygon edge counts
2: **for each** *vehicle* ∈Operating Area **do**
3: Compute vehicle's Voronoi Polygon V_p
4: Count the edges of V_p as C_e
5: Increment count in Map(C_e,count) by 1
6: **end for**
7: Create total Voronoi Polygon Count P_c equal to number of vehicles in operating area
8: Create total entropy count S
9: **for each** C_e ∈Map **do**
10: $S{=}S{+}\ (Map[C_e] \div P_c) \cdot \ln(Map[C_e] \div P_c)$
11: **end for**
12: **return** -S

In theory, robots patrolling in an orderly manner would display lower entropy than a chaotic swarm disrupted by frequent fueling. However, Voronoi entropy's edge metrics make it unsuitable to measure robot spreading. When robots leave the operating area to fuel, few operating robot Voronoi tessellations exist. Fewer tessellations lower the likelihood of diverse edge counts. Consequently, Voronoi entropy can improve during periods of heavy fueling even when most robots are not accomplishing their tasks.

2.2.3 Probabilistic Sensor Coverage

Scientists use probabilistic sensor coverage analysis to estimate how wireless signals cover an area, identify holes in mobile sensor networks, and predict how well

sensor array detect moving targets. These algorithms generally involve evaluating how well objects, whose coverage abilities decay probabilistically with distance, cover an area. While these algorithms provide concrete ways to measure how vehicle spread impacts productivity, they have specific characteristics making them unsuitable for robot spreading measurements.

For example, [AKJ05] Probabilistic Coverage Algorithm used probabilistic sensor detection models to estimate coverage of a wireless network with overlapping nodes. Specifically, the algorithm creates distance-based wireless probability gradients around nodes, sums overlapping nodal probabilities using the likelihood no overlapped gradients cover an object, and combines these overlapped probabilities with coverage probabilities from non-overlapped nodes to estimate full area coverage. While this algorithm provides excellent real time coverage estimation, it requires significant geometric computation and code development to work effectively within open source marine autonomy software like MOOS-IvP. Further, this algorithm requires users upload probabilistic sensor performance models. Many collaborative tasks (e.g search and rescue) cannot accurately measure how well their sensors perform over time. These poor sensor models would likely yield inaccurate or false coverage results.

Alternatively, [Eva22] implemented probabilistic sonar searching in MOOS-IvP. This experiment spawned multiple vehicles carrying simulated sonar sensors. These vehicles patrolled in assigned area and tried to acoustically detect an adversarial vessel driving nearby. While these algorithms run efficiently and indicate how robot position and movement affect detection performance, they cannot provide concrete spreading information without simulating multiple different adversarial runs. For example, a sparse group of robots may always detect an adversary when moving through the center of an operating area but never detect an adversary driving through a corner. These additional simulations cost valuable computation time which could be used to simulate scenario responses to changing vehicle quantity or restocking strategy.

Also, like [AKJ05] this algorithm requires that user estimate how well their sensors perform.

2.2.4 *Selected Metric*: Average Voronoi Cell Area

Scientists utilize Voronoi diagrams to solve problems across mathematics and computer science. Voronoi diagrams partition a region based on the objects or points located inside that region. Specifically, Voronoi diagrams partition regions into cells containing one object. The algorithm then cleaves the cells so that every point inside each cell lives closer to that cell's object than any other object in the region. In other words, in a region containing objects P1-P3, any location in cell P1 will be closer to P1 than points P2 or P3.

Figure 2-1: An operating area partitioned by the Voronoi algorithm. In this case, every point lying in P1's polygon is closer to P1 than P2 or P3.

Modern algorithms can compute Voronoi diagrams in $\mathcal{O}(n \log n)$ time making cell area statistics easy to compute[For86]. The following method uses Voronoi cell area statistics to analyze vehicle spreading and task performance:

Algorithm 3 Voronoi Cell Area Performance Analysis

INPUT: Vehicle operating states
OUTPUT: Simulation Voronoi mean cell area

1: **while** Simulation$\neq$ *Complete* **do**
2: Identify all vehicle states. Label vehicles transiting to, receiving, or returning from fueling FUELING.
3: Identify all other vehicles in operating area PATROLING
4: Compute Voronoi cells for all PATROLING vehicles
5: Compute instantaneous Voronoi cell areas for all PATROLING vehicles.
6: Average instantaneous cell areas as instantaneous average grid area
7: Log instantaneous average grid area every minute.
8: **end while**
9: **for all** *InstantaneousGridAreas* $\in$Mission Logs **do**
10: Average instantaneous areas as mean cell area
11: **end for**
12: **return** mean cell area

Average Voronoi cell area doesn't explicitly identify robot spreading in an operating area because average area does not indicate vehicle location in a cell. However, when robots pull off station to fuel, they leave unoccupied holes in the operating area which cannot immediately be filled by other patrolling robots and require robots to reposition to obtain optimal spreading. These sustained holes in the operating area drive up instantaneous Voronoi cell areas resulting in a higher simulation mean cell area. Consequently, when comparing two simulations containing the same number of robots, the simulations with higher overall mean cell area experienced more robot disruption to fueling, more operational holes, and therefor worse spreading.

2.3 Common Trait 2: Robot Movement

Besides spreading, robots maximize performance in collaborative tasks when they consistently move around the operating area. Specifically, robot movement enhances

task performance in the following ways:

- Bathymetric Surveys: Ships must tow mapping sensors through an entire operating area to map all bottom features.
- Weather Observation: Moving around large operating areas improves weather sample diversity.
- Sonar Search: [Eva22] Showed that when conducting a multi-vehicle sonar search, continuous vehicle movement alone raises the likelihood of detecting an adversary.
- Search and Rescue: Ships may need to move through sites multiple times to identify drifting debris and small objects obscured by high waves. Further, ships cannot rescue or recover objects they cannot drive to.

2.4 Measuring Robot Movement

Each collaborative task rewards movement in different ways. Consequently, robot movements cannot be simply inferred through task completion. However, robotics engineers developed metrics to measure robot movement.

2.4.1 Problems With Inferring Movement from Task Performance

Since all robots must move to complete their tasks, any common productivity scenario could simply implement an existing movement heavy task (e.g sonar search, bathymetric survey etc), score how well the robots perform the task, and use that performance to conclude how robots would perform on other tasks. However, robots move in different ways during each task. For example, sonar searches may move robots exclusively towards high interest areas whereas weather searching may require robots visit defined areas in a certain order.

Besides diverse movement patterns, many tasks contain probabilistic elements that preclude comparing empirical robot performance against an analytical standard. For example, during sonar search, sensor performance, sea state, environmental conditions, and ocean bottom contouring all effect detection performance. These factors eliminate the ability to calculate what the "best" possible run looks like without making environment assumptions not relevant to the goals of the thesis. Consequently, robot movement cannot be inferred through task performance.

2.4.2 *Selected Metric:* Time Sampled Average Node Idle Time

Robotics engineers frequently monitor the time since robots last visited a location (node idle time) to measure how much robots move during a patrol. Most robot patrol problems generally involve robots moving to a point, marking that point as visited, and then employing situational metrics to guide them to a different point. Patrol algorithms that employ idle time normally assign higher priority to locations not recently visited. For example, while [MTR06] and [Chu+07] employ different patrol decision and coordination techniques, all their robots change plans in part because of changing node idle time.

[Mac+03]'s instantaneous graph idleness measured the "average instantaneous idleness of all nodes in a given cycle". They then averaged instantaneous idleness over an n-cycle simulation to create their "graph idleness". While these metrics provide excellent graph insights for small numbers of nodes and short cycles, long cycles can mask idle time spikes due to erratic robot behavior. For example, in large operating areas, robots may fuel several times before completing a cycle. These robot interruptions may boost uncaptured short term graph idle time prior to robots finishing their cycle. However, graph idle time fidelity can be increased by simply sampling instantaneous idle time at a specific frequency instead of at cycle end. The following method uses time sampled node idle time to measure robot movement:

Algorithm 4 Time Sampled Average Node Idle Time

INPUT: Scenario Operating Area, Vehicle operating states/locations
OUTPUT: Simulation Mean Node Idle Time

```
Divide operating area into a 2D grid made of cells
for each cell ∈grid do
    Store cell's time since last vehicle visit
end for
Identify all vehicle states. Label vehicles transiting to, receiving, or returning from fueling FUELING.
Identify all other vehicles in operating area PATROLING
for each ElapsedSecond ∈Simulation do
    for each cell ∈grid do
        Increment cell's time since last visit by 1 second
        for each vehicle ∈mission do
            if Vehicle located in cell and Vehicle state is PATROL then
                Set cell's time since last visit to zero
            end if
        end for
        if ElapsedSecondsmod60 equals 0 then
            Average all cell idle times
            Log average as an instantaneous average idle time
        end if
    end for
end for
for all InstantaneousAverageIdleTimes ∈Mission Logs do
    Add instantaneous idle time to overall simulation average node idle time
end for
return Simulation Average Node Idle Time
```

Time sampled average node idle time provides immediate indication's about how vehicle operation changes impact productive operational movements. For example, when a vehicle slows down to improve spacing between other vehicles, it visits fewer nodes for a given time causing average idle time to rise. Further, when vehicles must leave the operating area to fuel, they lower the total number of patrolling robots capable of visiting nodes. Consequently, when comparing two simulations containing the same number of robots, robots in simulations with higher average node idle time experienced more fueling disruptions and remained less productive.

2.5 Developing a Common Robot Productivity Task

Any task considered for a common productivity scenario must satisfy the following characteristics:

- Since these experiments will eventually implement strategies to predict and manage resource needs, the task must force the vehicles to constantly consume abundant amounts of fuel. Further, to add realism, the task must force vehicles to consume fuel at variables rates.

- Since these experiments will use vehicle spreading and movement to infer scenario performance, the task must force vehicles to move constantly and adjust their behavior to obtain balanced positions throughout the operating area.

- As previously stated, existing collaborative tasks (e.g sonar search, rescue, etc) require vehicles change their behavior to adapt to exogenous mission circumstances like changing target behavior or updated rescue data. These unpredictable changes require any mission performance analysis to be completed using only empirical simulation. The task chosen for a common scenario must be structured such that it's performance can be estimated analytically.

- Since these experiments focus on vehicle swarms working together, the task must require collaboration, must work under decentralized control schemes, and must work in low bandwidth environments.

2.5.1 Selected Task: Modified Multirobot Patrol Problem (MRPP)

Related Work

Multi Robot Patrol problems plan and optimize how groups of robots periodically visit sets of points. Many concepts that influence MRPP work come from past work analyzing individual robot patrol and path planning. This individual robot patrol work draws heavily on work relating to cellular decomposition-based path planning. Cellular decomposition partitions an imperfectly shaped operating area into a set of polygons. Robots can then use these polygons to create survey paths around each polygon. [Lat91] used a trapezoidal decomposition method to partition operating areas into navigable graphs of trapezoids and triangles. [CP98] then modified the trapezoidal decomposition to segment an operating area using fewer polygons and simpler survey patterns. While these works illustrate methods for robots to partition and patrol an operating area, they do not examine how multiple robots can coordinate on these patrols.

Multi robot patrol problems combine individual robot path planning with novel coordination techniques to create emergent patrol performance benefits. While researchers use MRPP to describe diverse types of multi robot problems, the most relevant patrol experiments test patrol algorithms containing frequency constraints. Like multi-agent persistent monitoring [Son+14], robot patrolling with frequency constraints generally requires robots visit all points in a way that minimizes node idle time. [Che04] developed an $\mathcal{O}(n^3)$ algorithm that created a multi agent patrol cycle whose idle time performance rivaled an ideal cycle created by a far more computationally expensive algorithm. Further, they also analyzed how individual vehicle graph partitioning affects graph idle time. To further improve performance, [EAK07] used Hamiltonian cycles created from minimum spanning trees to find an efficient optimal patrol path for objects set in a grid. They then used information from these cycles to control starting robot locations and guarantee a minimum patrol frequency. While these works guide how to implement and measure multi robot patrol, their

methods do not analyze how long-term multi-robot restocking affects idle time and patrol performance.

Benefits of Using a Multi Robot Patrol Problem to Analyze Robot Productivity

Multi Robot Patrol Problem's simple implementation, predictable tasking, and relevance to many collaborative tasks make it an excellent task to measure robot productivity. At a basic level, the MRPP simply requires vehicles to move around, visit points, track the points they visited, and combine this data with other robot information to plan their next visit. MOOS-IvP's mission planning and simulation capabilities contain all the tools required to easily develop and measure this type of scenario. Implementation aside, MRPP's simple scenario also allows for basic analytical productivity estimation. Simply adding basic assumptions on how robots move and restock in an MRPP allows easy estimation of an the best case idle time and Voronoi polygon area with and without restocking. Section 3.3.2 further describes these methods. Finally, MRPP's make great productivity scenarios because they resemble real world collaborative tasking. To resemble rescue or survey problems, MRPP's can be structured to require robots to cover an entire operating area. To resemble sonar search problems, MRPP's can be structured to require infrequent turns. This structural flexibility allows operational planners to estimate real world swarm capabilities without needing to develop and simulate their real-world swarm scenario inside MOOS-IvP.

2.5.2 Chosen Productivity Scenario Description

This section describes the environmental, resource, and control dynamics used to create a MRPP scenario for measuring productivity. Specific details describing how this scenario was implemented in MOOS-IvP can be found in section 3.3.1.

Environment

The chosen scenario takes place in the Pocomoke Sound off the coast of Saxis Virginia. While MOOS-IvP's simulation capabilities do not change based on a scenario's operating area size, the Saxis operating area provides enough space to simulate multi vehicle patrolling without crowding. Further, the Saxis operating area's size means that even high-speed vehicles create measurable productivity disruptions when they transit to restock at a depot. Finally, the Saxis operating area's coastal geometry allow this restocking scenario to be easily expanded in the future productivity testing. Specifically, future tests can easily measure how marine traffic and supplies leaving the coast affects swarm productivity.

Robots will patrol within a square shaped operating area outlined within the Pocomoke Sound. A square shaped operating area simplifies productivity estimation by allowing depots to spawn symmetrically around an operating area. Further, square shaped operating areas reduce the complexity required to create full coverage survey patterns.

Multi-Robot Patrol Problem Task Description

To ensure productivity measurements can be easily simulated and measured, robots will drive simple, long, reciprocating survey patterns designed to cover an entire square shaped operating area. These survey patterns will be adapted from existing capabilities within MOOS-IvP. MOOS-IvP already contains reciprocal survey generation capability currently used by single robots to conduct hydrographic mapping and single robot patrol. These experiments will adapt this survey generator to allow multiple robots to share and patrol from the same survey.

Figure 2-2: Drone photo showing the WAM-V USV operating at the MIT Sailing Pavilion [Bena]. The vehicle uses MOOS-IvP's survey generation capability to patrol an area of the Charles River for demonstration.

Surveys are generated in the operating area based on [Che04]'s two major patrol strategies. The first strategy forces all robots to evenly patrol the entire operating area using the same shared cycle. The second strategy evenly distributes robots across a geographic operating area, partitions the area surrounding each of the robots, creates a unique survey routine in each partition, and assigns each robot to patrol only in their specific unique partition. The first strategy tests a scenario in which robots are allowed basic inter vehicle communication to continuously space themselves equally along a single global survey. The second strategy tests a scenario in which robots cannot share their position with each other and thus must patrol in their own box to avoid collision.

Measuring Robot Productivity

As previously described, swarms are most productive when they minimize the average time since a robot last visited every portion of the operating area (idle time) while also minimizing the total average area closest to each robot (Voronoi polygon area). To track time since last visit, each simulation will divide the operating area into square cells. Any time a robot enters a cell, the simulation resets the cell's time since last visit. Periodically, the simulation averages and records all current cell times as described in the section 3.3.1 paragraph describing `pLatencyGridReporter`. The scenario will track Voronoi polygon area using the `uFldVoronoi` application described in section 3.3.1.

Chapter 3

Developing Resource Management Algorithms for the Common Scenario

The common robot productivity scenarios' continuous robot patrol mimics activities vehicles would do when performing collaborative tasking. Measuring the performance impact caused by robots refueling during this patrol requires developing and implementing resource management algorithms into the simulation. For this section, resource management algorithms refer to methods and tools required to track and fuel patrolling robots. Given the limited relevant work related to refueling robot swarms, this section instead focuses on implementing naïve fueling strategies and comparing their performance to the analytical ideal performance.

3.1 Related Work

Despite significant work dedicated to optimizing multi-robot patrol algorithms, very few studies explore or implement resource constraints into their optimization strategies. For example [JHL21] develops a patrol algorithm to visit the most points in a cycle for a given amount of energy. However, this paper does not discuss how to optimize routes over multiple restocking periods. In other words, the paper only plans routes over a single robot charge cycle. [DMK11] presents a novel way to maintain sustained robot area coverage using Voronoi polygons. Specifically, their algorithm

links a robot's charge state to its respective Voronoi centroid. This weighting causes low energy robots to gradually move towards depots. While these experiments develop a novel way for robots to continuously cover an area, they do not force the working robots to move continuously. Further, these experiments also allocate one charger (depot) per robot. These design choices ensure a robot will always be close to its own depot. In other words, these experiments don't integrate depot scarcity into their analysis. [HLH09] implements a cooperative auction where robots bid for patrol points based on energy, point idle time, location, and other factors. While these algorithms do allow for energy aware multi robot patrol, they require significant inter robot communication making them difficult to implement in large, low bandwidth environments. Finally, none of these experiments designed their resource management algorithms to focus on routinely visiting all points in an operating area.

Outside of robot patrol, significant supply chain work has been dedicated to optimizing and solving variations of the multi-vehicle routing problem. These works attempt to optimize how to route groups of replenishing vehicles to efficiently restock groups of consumers. [Li+14] attempts to route fuel trucks to restock consumers while minimizing the total time trucks spend restocking. [Cor+08] attempts to route fuel trucks to restock consumers while minimizing the total cost spent to restock all consumers. While these works illustrate methods to track and view resource consumption across a network, they only optimize delivery routes for stationary consumers.

3.2 Creating Patrol Specific Resource Algorithms

Reviewing related supply and robot patrol work indicates there are limited works dedicated to developing swarm resource management algorithms. Further, with limited real time swarm operations, no historical restocking data exists to compare with any simulated resource management algorithms. Consequently, in this work, resource management algorithms will instead be validated by implementing naïve algorithms and comparing their performance against ideal best-case calculations. Specifically,

resource management algorithms will be evaluated using the following methods:

- Develop a naïve restocking strategy to prevent robots from running out of fuel during a patrol.
- Mathematically estimate the average grid idle time for robots patrolling on a global polygon with infinite fuel. Analytically calculating the average Voronoi polygon area is not necessary since the average Voronoi polygon area always works out to the operating area divided by the number of vehicles. These values represent the absolute best possible metrics a swarm could achieve.
- Simulate and measure robots patrolling with infinite fuel. Verify these numbers approximate the best case calculations.
- Analytically estimate the average cell idle time and average Voronoi polygon area for robots fueling with stationary depots.
- Simulate and measure robots patrolling with finite levels of fuel. Verify empirical performance metrics match analytical calculations.

This analysis does not guarantee the implemented resource management algorithms create the smallest possible mission disruption when fueling robots. However, the analysis does ensure the simulation performs as expected and benchmarks the gap between patrol with basic resource management algorithms and patrolling under an infinite resource ideal scenario.

3.3 Restocking strategy

The robots will utilize a naïve restocking strategy during their patrol. Specifically, neither the vehicles nor depots apply any forward-looking algorithms when fueling. A vehicle only considers its current fuel level when asking to restock and will not opportunistically short cycle fueling when it drives past a depot. Further, depots carry unlimited fuel and will always approve a vehicle to leave patrol and refuel.

They can also fuel an unlimited number of vehicles at once. These depot vehicle interactions mean the entire patrol fleet can pull off patrol to refuel at once should all vehicles run out of fuel at the same time.

To accomplish the naïve fueling strategy, each robot patrolling a survey tracks their own fuel consumption. Robots consume fuel exponentially based on their current speed. To determine when they must refuel, each robot calculates a minimum order point (MOP) corresponding to the minimum fuel level required to sustain a transit to and return from the closest available fuel depot. Each robot calculates their MOP from their moving average speed, moving average fuel consumption, and distance to their closest depot. Depots spawn symmetrically at the edge of the global operating area when the simulation begins. When the ships reach their minimum order point, they leave their patrol scheme, refuel at a depot, and return to the place where they left off in the survey.

Each experiment's robot patrol strategy dictates how other robots in a swarm respond to other refueling robots. For scenarios using robots patrolling in a shared global survey routine, removing a robot to fuel makes other ships speed up or slow down to restore equal distance in a survey cycle. For scenarios where robots patrol in their own partitioned survey box, removing a robot for fueling causes no change to other robot behavior. In other words, when robots refuel in a partitioned scenario, they leave their partitioned survey area completely un-patrolled while they refuel.

3.3.1 Implementing the Common Productivity Scenario in MOOSIvP

MOOS-IvP Core Description

MOOS-IvP is an open source software codebase dedicated to autonomous (primarily marine) vehicles. It has been publicly available since 2006 and is currently used on many types of marine robotic platforms, underwater and surface, around the world. The codebase allows software developers to turn marine robots into autonomous vessels capable of advanced decision-making, data collection, and safe retrieval.

MOOS-Ivp's software architecture and licensing enable users to augment the core capabilities with new modules in a manner that (a) does not require coordination or changes to the public codebase, and (b) provides the user with the option of retaining privacy of code augmentations, or redistributing under an open source or commercial license.

MOOS-IvP is comprised of two architectures, each with their own notion of what constitutes a new plug-in module. The IvP Helm is a behavior-based architecture and modules have the form of new behaviors. Specifically, the IvP helm and supporting IvP software libraries allow developers to write vehicle behaviors that take in sensor information, plan responses to this information, and act without user assistance[Benb]. These vehicle behaviors and supporting subsystems all communicate using MOOS. MOOS is a publish-subscribe robot middleware, and new modules come in the form of new MOOS apps. Initially created at MIT by Dr. Paul Newman and now maintained by Dr. Newman at the Univ. of Oxford, MOOS's tools build multithreaded, distributed, and real-time communication capabilities into vehicle applications[Benb]. Combined, these two open-source applications decouple a vehicle's autonomous capabilities from its platform specific robot control software.

The MOOS-IvP codebase contains modules for robot autonomy as well as a full multi-vehicle simulation and post-mission analysis toolset. Augmentations described in this thesis include new MOOS apps for both autonomy, simulation and performance analysis. Section 3.3.1 "Creating a MOOS-IvP Mission" further describes these MOOS apps.

Publish-Subscribe Architecture Description The MOOS-IvP ecosystem and applications used to measure swarm productivity share data through a publish-subscribe architecture. All MOOS applications in a vehicle or shoreside system form a MOOS community. Every vehicle and shoreside community contain their own database (MOOSDB) for storing variables needed by their respective MOOS applications. MOOS applications publish variables to a MOOSDB when they want to make

these variables available to other applications. These same applications can then subscribe to variables published by other MOOS application to use in their own computation. Subscribing to only needed variables prevents an application from wasting time processing unneeded data. Finally, MOOS-IvP's uFldNodeBroker, uFldNodeComms, and pShare applications allow MOOS communities to share variables with each other using standard TCP/IP networking protocol. The following two examples illustrate how MOOS applications use the publish and subscribe architecture.

- Intra Vehicle Communications: A vehicles GPS receiver sends signal data to a MOOS application designed to process the GPS signal. After processing the signal, this GPS application then converts the Vehicles Latitude and Longitude to (X,Y) coordinates and publishes real time coordinates to the vehicles' MOOSDB as "NAV_X" and "NAV_Y". The vehicle's Odometer, an application subscribed to "NAV_X" and "NAV_Y", then receives notification that these variables were updated, ingests their new values, and computes how far the vehicle has traveled from its previous location. The Odometer then publishes "ODOMETER_DISTANCE" to the vehicles MOOSDB.

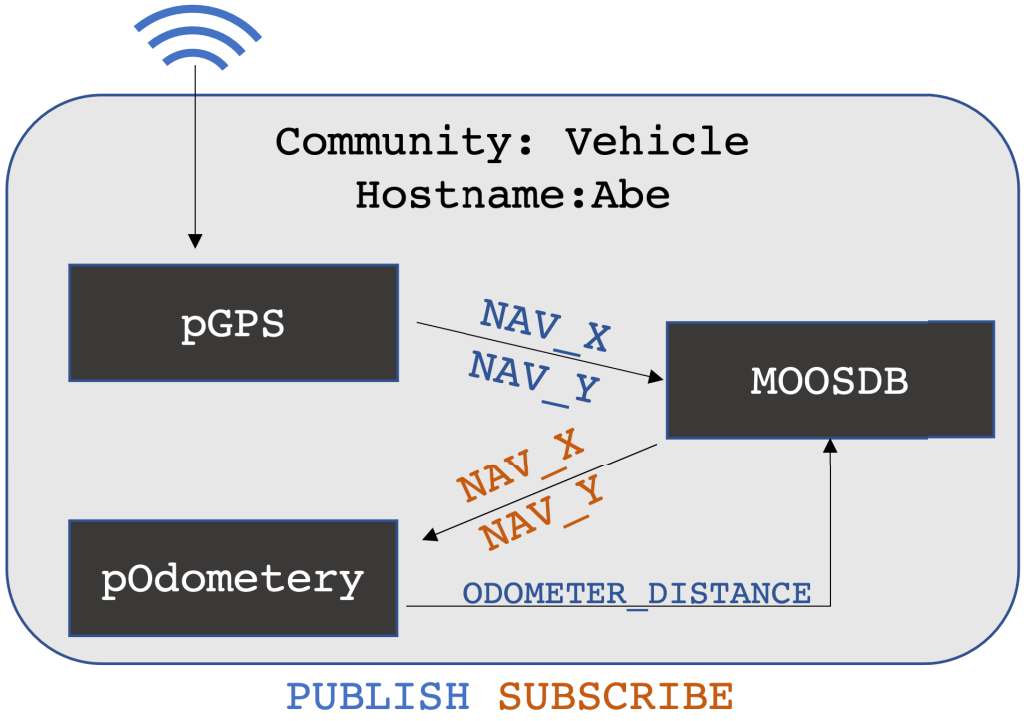


Figure 3-1: Intra vehicle publish and subscribe behavior.

- Vehicle/Shore Communications: A vehicles publishes its current helm status and location in the string variable "NODE_REPORT" to its MOOSDB. The vehicle's ip sharing application pShare then sends receives notification "NODE_REPORT" was updated in the MOOSDB. pShare sends "NODE_REPORT" to the shoreside instance of pShare who then publishes "NODE_REPORT" to the shoreside MOOSDB where its status can then be displayed to an operator.

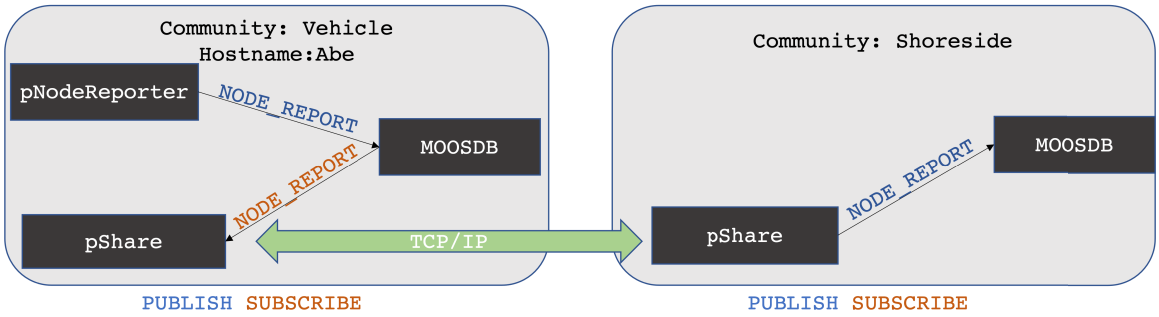


Figure 3-2: Inter vehicle publish and subscribe behavior.

MOOS-IvP Simulation Capabilities While MOOS-IvP communities normally run on physically independent vehicles, MOOS-IvP's unique qualities allow users to simulate multiple communities on a single host. First MOOS-IvP's platform independent architecture allow users to run MOOS communities on any host capable of compiling C++ code and communicating over TCP/IP. Consequently, the same applications that run on an operational vehicle can be run on a virtual vehicle located on shore testing server with no additional software configuration. Further, MOOS-IvP contains an application called uSimMarineV22 capable of simulating ship movements. uSimMarineV22 ingests helm commands and past vehicle locations into a PID controller, calculates resulting ship location changes, and publishes these ship location changes to its respective MOOSDB. Finally, MOOS-IvP's time-warp functionality allows a user to run virtual applications at update speeds faster than normal time. Consequently, users with hardware capable of supporting fast paced applications can run week long virtual events in just a few hours.

Launching And Killing MOOS-IvP Simulations MOOS-IvP contains several applications that allow users to automatically launch, start, monitor and kill MOOS simulations. MOOS-IvP communities contain numerous applications required to complete each mission (pHelmIvP, pLogger, uSimMarineV22, etc). Users launch all these applications at once using the MOOS application pAntler. This application reads configuration files dictating what applications need to run in a given community, parses how each application should be configured (time-warp, startup variables, etc), and then launches all applications listed in the configuration file. With the community running, users can start missions using the application uPokeDB. This program simply adds any user defined variable to a communities MOOSDB. Users can then program helm behaviors in pHelmIvP to activate when uPokeDB updates this variable. While the mission runs, users can periodically monitor the progress of their missions using uQueryDB. uQueryDB reads a running mission's configuration file, identifies variables of interest, queries these variables in a communities MOOSDB, and compares their values to pre-defined pass-fail conditions. Killing pAntler after

a successful uQueryDB condition brings down all child processes and ends each missions. Combined with Bash scripting, these tools can simulate diverse autonomous environments with no user input.

Swarm Toolbox Description Created by MIT's Marine Autonomy Lab (Pavlab), the swarm toolbox contains scripts, behaviors, applications, and monitoring tools designed to operate large swarms of MOOS vehicles over great distances. Many experiments detailed in later sections used the toolboxes' uFldVoronoi and pProxonoi applications to stage ships for testing and later evaluate their Voronoi area performance when patrolling. Descriptions of their Voronoi tessellation algorithm can be found in later sections. Besides Voronoi tools, these experiments also used the toolboxes' swarm launch scripts and random vehicle spawning tools.

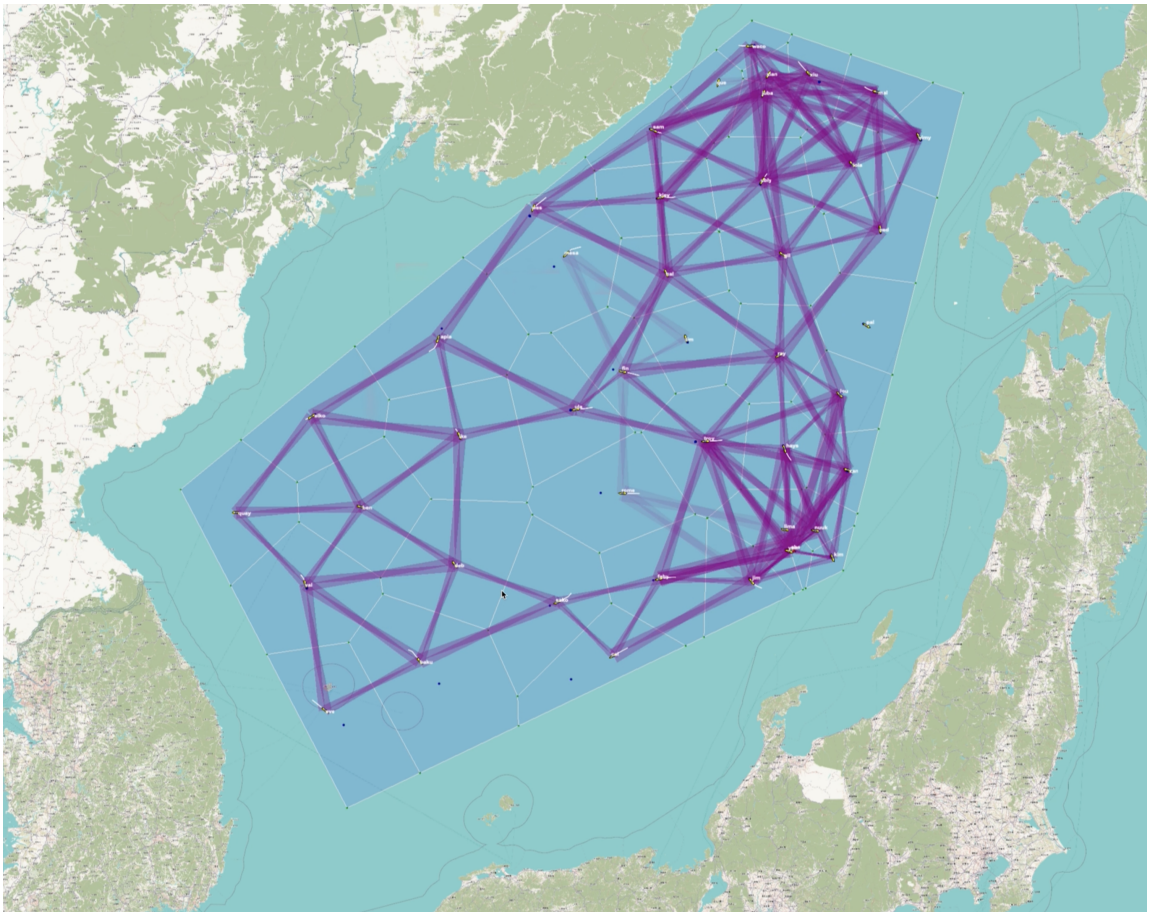


Figure 3-3: Swarm toolbox mission demonstration showing 50 vehicles dynamically spreading out in the Sea of Japan under a limited communication scheme [Ben22]

Creating a MOOS-IvP Mission

Running or simulating scenarios in MOOS-IvP first requires creating a mission. Missions dictate where vehicles operate, what applications run on vehicles, how vehicles behave, what helper applications run on the mission's shoreside control node. Implementing the common productivity scenario required modifying the uFldSaxis mission originally created to test multi-vehicle deployments. The original uFldSaxis mission can be found in the stock missions bundled with the MOOS-IvP swarm toolbox. After final modification, the uFldSaxis mission contained the following major components:

Meta_Shoreside.MOOS This file launches a "shoreside" mission community. Shoreside communities contain software used to track, record, communicate with, and display vehicles and depots. The shoreside community runs the following unique applications to accomplish its mission.

`uFldVoronoi` The swarm toolbox application `uFldVoronoi` provides Voronoi calculations and metrics for the shoreside node. Specifically, uFldVoronoi takes in position reports from all vehicles, calculates Voronoi polygons for each vehicle, and publishes Voronoi stats such as minimum and maximum Voronoi polygon area. `uFldVoronoi` also calculates when vehicles reach an Voronoi equilibrium state by determining when the distance between a vehicle and its Voronoi centroid falls within a designated capture radius. Beyond standard swarm toolbox capabilities, the uFldSaxis mission runs a modified `uFldVoronoi` version containing additional capabilities. This modified version now publishes a variable indicating that all vehicles reached a Voronoi equilibrium state. `bhvSurvey` uses this Voronoi equilibium status to transition the helm mode from COVERING to PATROLING.

`pLatencyGridReporter`**(PLG)** PLG logs running instantaneous grid idle time, by recording and averaging the time since a robot last visited each cell in an operating area, using a grid object and vehicle positions. Written as an `AppCastingMOOSApp`, PLG runs on the MOOS shoreside community and tracks vehicle positions by sub-

scribing to vehicle NodeReports passed to the shoreside community. The program then records and averages grid cell idle time using the algorithm described in Alg 5. The `Meta_Shoreside.moos` file specifies the size of the grid as well as grid cell size. Further, if a mission runs `pMarineViewer` on its shoreside community, PLG can output individual cell idle times for colorful rendering in `pMarineViewer` (Fig 3-4).

Algorithm 5 Instantaneous Grid Idle Time

INPUT: XYGenPolyGrid m_grid ,double $m\_last\_increment\_time, Map < vehicle, NodeReport > m\_vehicles$:
OUTPUT: Instantaneous Grid Idle time

```
1: for each iteration ∈ Mission do
2:     moos_elapsed_time ← m_curr_time − m_last_increment_time
3:     for each cell ∈ m_grid do
4:         cell.value ← cell.value + moos_elapsed_time
5:         m_last_increment_time ← m_curr_time
6:     end for
7:     for each vehicle ∈ m_vehicles do
8:         cell_index ← getCellIndex(vehicle.x, vehicle.y)
9:         m_grid[cell_index] ← 0
10:    end for
11:    if m_curr_time − m_last_log_time > 60 then
12:        totalCount ← 0
13:        for each cell ∈ m_grid do
14:            totalCount ← totalCount + cell.value
15:        end for
16:        log(totalCount/m_grid.size)
17:        m_last_log_time ← m_curr_time
18:    end if
19: end for
```

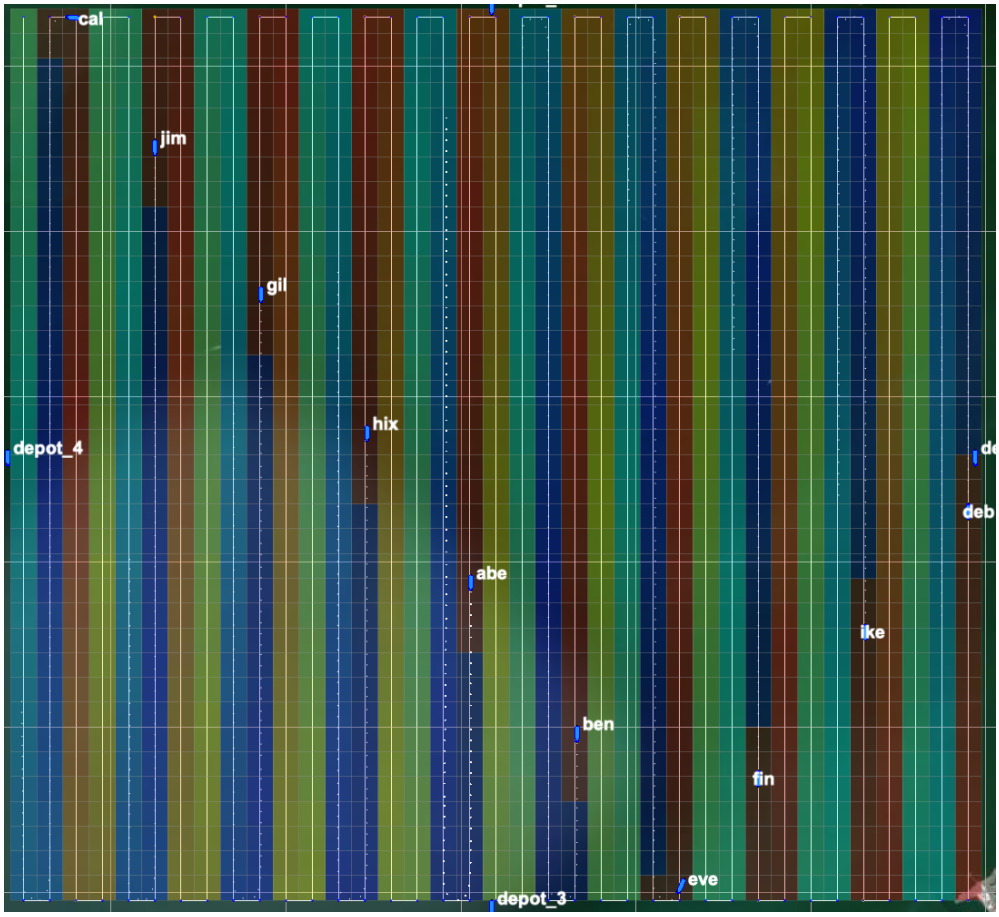


Figure 3-4: Graphical rendering showing latency grid tracking 10 ships patrolling an operating area. Cells recently visited by ships render in blue. As time elapses without a ship visit, the squares turn red. While PLG maintains near instantaneous grid tracking in the background it infrequently updates its graphics output. Consequently, several cells may remain red after a ship visit until PLG re-renders the grid for display

`pickPos` Given a quantity of ships, a start heading and polygon coordinates, `pickPos` spawns unique random ship coordinate sets inside the given polygon. For example, given 10 ships and a square polygon, `pickPos` returns 10 sets of ship headings and (X,Y) coordinates located inside the square polygon. This behavior allows uFldSaxis's mission launch script launch dynamic numbers of vehicles in a given mission. `pickPos` comes bundled with the MOOS-IvP Swarm Toolbox.

`pickDepot` Given a depot quantity and square operating area coordinates, pick depot symmetrically spawns depots coordinate sets clockwise around an operating area. Specifically, `pickDepot` first determines how many depots belong on each side of the operating area. For quantities less than four, `pickDepot` will place one depot per side counterclockwise until it runs out. For depot quantities greater than four but less than eight, pick depot will determine how many sides receive two depots given that each side must receive at least one depot. `pickDepot` then spawns two depot sides in a clockwise direction for all sides receiving two depots before spawning one depot on the remaining sides. `pickDepot` repeats these behaviors for depot quantities greater than 8 using progressively more depots per side. `pickDepot` spaces each depot evenly across its respective side. For example, `pickDepot` will spawn a depot in the middle of a side for sides with only one depot. For three depots, `pickDepot` spawns each depot separated by one third of the side length.

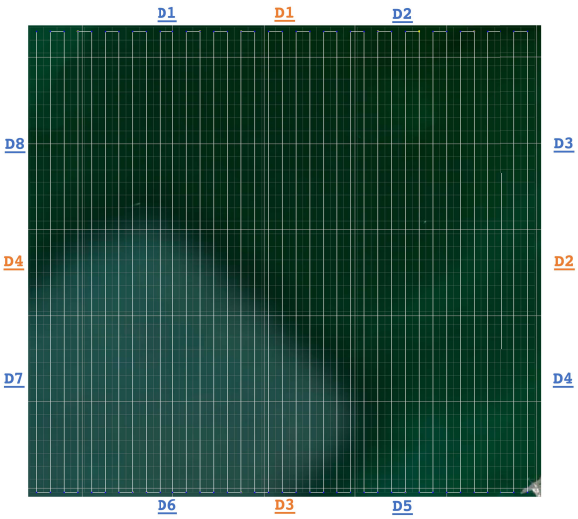


Figure 3-5: Depot spawn locations scheme for four and eight depots. In each case, `pickDepot` spawns depots counter clockwise around the operating area.

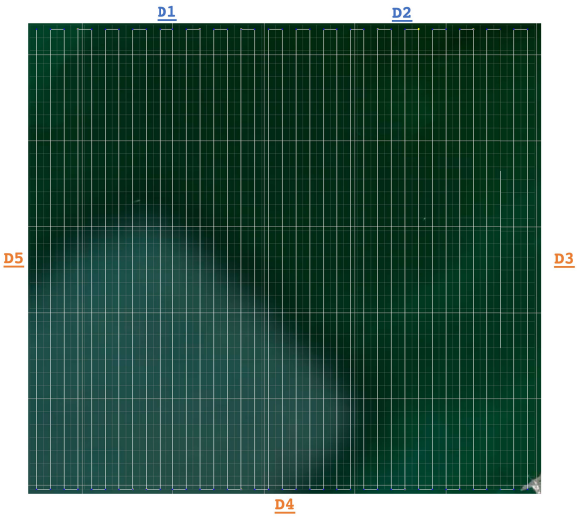


Figure 3-6: Depot spawn location for five depots. In this case, `pickDepot` populates the north side with two symmetric depots and then continues populating one depot on the east, south, and west side.

Meta_Vehicle.MOOS, Meta_Depot.MOOS, Meta_Shoreside.MOOS Each meta MOOS file contains configuration parameters for all `AppCastingMOOSApp` applications running on vehicles in their respective community. At mission start, a launch script uses `nsplug` and each meta MOOS file to create unique .MOOS files for each vehicle. For example, for a vehicle named "abe", the script would create a unique "targ_abe.moos" file. `pAntler`, then launches and configures the applications specified in the vehicles respective .MOOS file.

Meta_Vehicle.bhv *Meta_ Vehicle.bhv* contains vehicle helm modes required to run the *uFldSaxis* mission(Figure 3-7). Each helm mode represents different vehicle events and dictates what behaviors influence vehicle course and speed. When vehicles first deploy at the start of the mission, *Meta_ Vehicle.bhv* ensures all vehicles start in the ACTIVE:REGION_COVER mode. In this mode, `BHV_Voronoi` operates to spread vehicles evenly across an the operating area. Once all vehicles achieve equilibrium, `uFldVoronoi` activates the ACTIVE:PATROLLING mode. In this mode, the `bhvSurvey` behavior begins robot patrolling. When vehicles must refuel, they operate in either the TRANSITING, RECEIVING, or RETURN_FROM_FUELING mode depending on their refueling status.

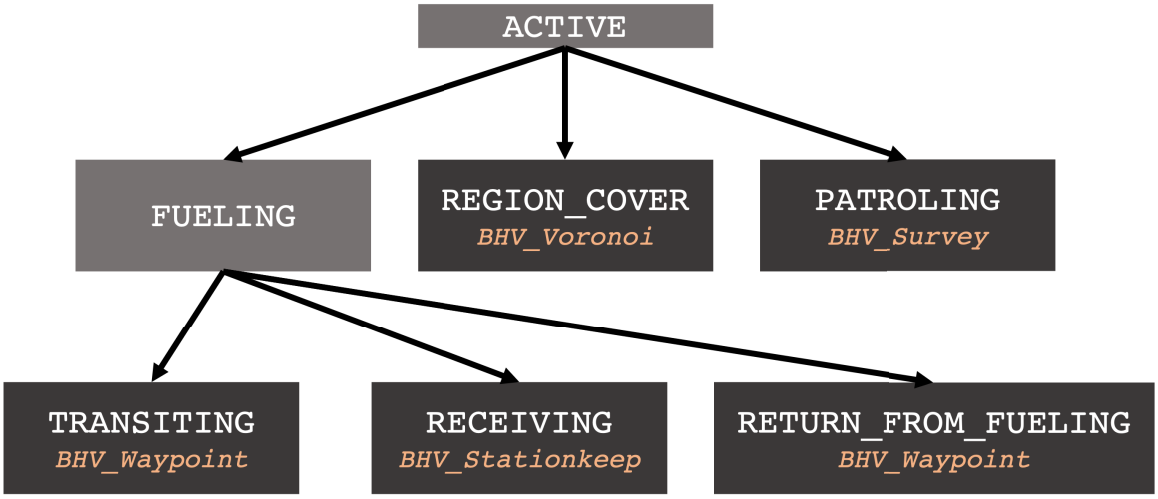


Figure 3-7: Possible vehicle modes (white) and their the behaviors running during each mode (orange). Dark grey modes represent a final mode state a vehicle can be in. For example, A vehicle can be in mode ACTIVE:PATROLLING or mode ACTIVE:FUELING:RECEIVING

`uResourceBroker` `uResourceBroker` runs on every depot and manages all depot-vehicle refueling interactions. While `uResourceBroker` can operate in several refueling modes, in the `uFldSaxis` mission, `uResourceBroker` operates in an open refueling state. In this state, `uResourceBroker` accepts all vehicle refueling requests regardless of quantity previously accepted requests. `uResourceBroker` then tracks each vehicle en route to refuel and will refuel the closest vehicle that falls within its refueling capture radius. Finally, in an open refueling state, `uResourceBroker` refuels vehicles instantly. In other words, any vehicle that negotiated a refueling contract with a depot will obtain a full fuel tank as soon as they pass into the depot's capture radius.

`pGas` Running on every vehicle, `pGas` tracks fuel consumption, determines when a vehicle needs to refuel, and brokers refueling operations with depots. `pGas` knows its vehicle's location and uses this location to calculate a vehicle's moving average speed (Algorithm 7). `pGas` also knows the vehicles current thrust output. Instead of vehicle speed, `pGas` uses this thrust output to approximate a vehicles instantaneous fuel consumption (Algorithm 6). Basing consumption off of vehicle thrust instead of speed allows accurate fuel calculations even if a vehicles fluid resistance changes.

Every instantaneous fuel consumption reading can then be accumulated to find a moving average consumption.

With consumption C_{Avg} and moving average speed V_{Avg} calculated, the vehicle calculates its minimum order point using the equation below. This minimum order point allows a depot to abort travel to a depot that becomes unavailable, return to its original position, and drive to a new available depot.

$$Op_{min} = 3 \cdot \frac{d_{farthestDepot} \cdot C_{Avg}}{V_{avg}}$$

To broker refueling messages, `pGas` maintains a list of all available depots. When the vehicle reaches its minimum order point, `pGas`, contacts the closest available depot via `NodeMessage` to request refueling. If the depot denies a vehicles request, `pGas` continues to contact other available depots. If a depot accepts a vehicles request, `pGas` changes the vehicle mode state to FUELING:TRANSITING and sends BHV_Waypoint the target depot's coordinates. When a depot notifies `pGas` that it successfully fueled the vehicle, `pGas` updates the vehicle's tank level, changes the vehicle mode state to FUELING:RETURN_FROM_FUELING, and sends bhvWaypoint return position coordinates.

Algorithm 6 Instantaneous Vehicle Fuel Consumption

INPUT: double $time\_initial$,double $time\_final$,double $thrust\_initial$,double $thrust\_final$, ConsumptionMap m_cmap, double $m\_fuel\_inventory$:
OUTPUT: Updated $m\_fuel\_inventory$

1: $delta\_t \leftarrow time\_final - time\_initial$
2: $avg\_thrust \leftarrow (thust\_final + thrust\_initial)/2$
3: $avg\_consumption \leftarrow m\_cmap$.`getConsumption(`$avg\_thrust$`)`
4: $quantity\_consumed = delta\_t \cdot avg\_consumption$
5: $m\_fuel\_inventory \leftarrow m\_fuel\_inventory - quantity\_consumed$

`pProxonoi/bhvVoronoi` The swarm toolbox applications `pProxonoi` and `bhvVoronoi` run on each vehicle and work in tandem to evenly distribute vehicles across an operating area even when all vehicles cannot communicate with each other. In the

Algorithm 7 Moving Average Speed Calculation

INPUT: vector<pair<double,double> $m\_odometer\_history$,double $moving\_average\_time\_window$: **OUTPUT:** double $moving\_average\_speed$

```
1: pair < double, double > cur_r ← m_odometer_history[0]
2: for all odo_r ∈ m_odometer_history do
3:     if cur_r.time − odo_r.time > moving_average_time_window then
4:         return (cur_r.distance − odo_r.distance)/(cur_r.time − odo_r.time)
5:     end if
6: end for
7: last_r = m_odometer_history[m_odometer_history.size() − 1]
8: return (cur_r.distance − last_r.distance)/(cur_r.time − last_r.time)
```

`uFldSaxis` mission, these applications spread vehicles across the Saxis operating using the helm COVER mode. `pProxonoi` takes in updated reports of nearby vehicle known positions and calculates each vehicles Voronoi polygon. `pProxonoi` then sends this polygon information to the vehicle's `bhvVoronoi` behavior. `bhvVoronoi` finds the centroid of the vehicles individual Voronoi polygon. It then creates a helm course to drive the vehicle towards this Voronoi polygon centroid. This process of repeatedly calculating a Voronoi polygon, calculating its centroid, and driving towards that centroid forces each vehicle to spread evenly thoughout an operating area.

bhvSurvey `bhvSurvey` uses survey generation libraries from the `Swarm Toolbox` to generate and follow surveys patterns located within a polygon. `bhvSurvey` operates in two different modes depending on the a users desired patrol algorithm.

In *global* polygon mode, a user gives all vehicles the same operating area polygon at startup. Each vehicle then patrols this same pattern while remaining equally spaced along the survey track. Vehicles achieve this equal spacing using variable speed control. Since the survey ends in the southeast corner, the most westward vehicle acts as the lead vehicle and sets a constant pace. Vehicles then speed up or slow down based on their position in relation to the lead vehicle. Algorithm 8 describes this speed control scheme in detail and figure 3-9 shows ten vehicles patrolling in this scheme.

In *local* polygon mode, each vehicle patrols in its own unique polygon. Initially each vehicle's `pProxonoi` and `bhvVoronoi` applications spread each vehicle out over an entire operating area. While they spread out, the shoreside application `uFldVoronoi` tracks how close vehicles are to reaching a spread out equilibrium state. When vehicles reach a satisfactory equilibrium state, `uFldVoronoi` sends each vehicle its own current Voronoi polygon to patrol. Importantly, `uFldVoronoi` only passes vehicles their polygons to standardize logging and patrol simulation start times for later observation in this thesis. Each vehicle tracks and maneuvers using their own decentralized Voronoi polygon created by `pProxonoi`. Vehicles could easily commence patrolling using only calculations from this decentralized polygon however each vehicle's `pProxonoi` would transition to vehicle patrol at slightly different times. With polygons passed to each vehicle, `bhvSurvey` then generates the largest possible survey in its given polygon and begins patrolling. Figure 3-8 demonstrates ten vehicles operating in a local polygon survey scheme.

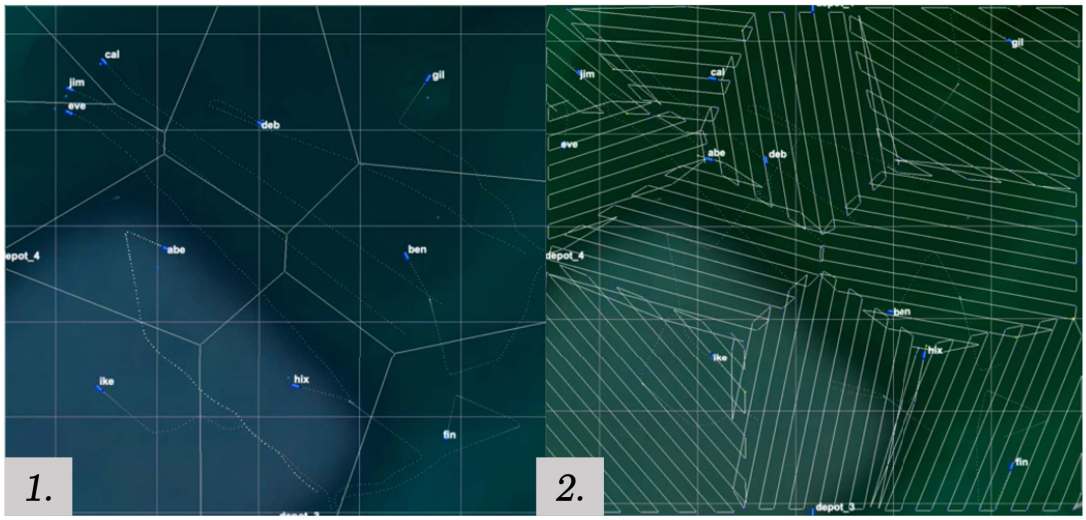


Figure 3-8: bhvSurvey functioning in a local polygon mode. In state one, vehicles spread themselves out using pProxonoi and bhvVoronoi behavior in the "Cover" mode. Once uFldVoronoi detects vehicles have spread out, passes individual Voronoi polygons to bhvSurvey (State two). Each vehicles' bhvSurvey then creates its own survey pattern and begins patrol.

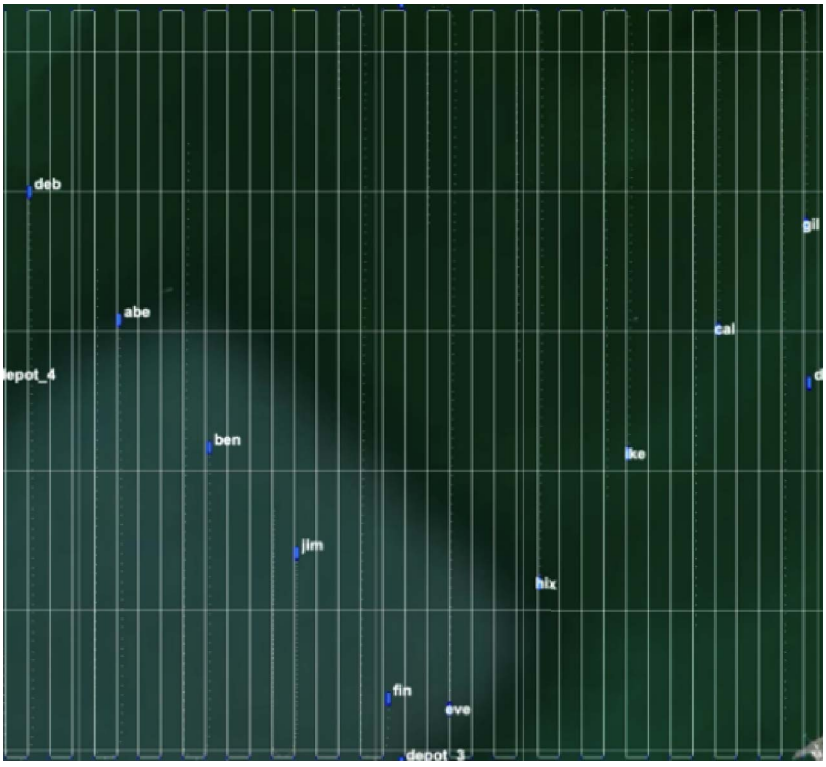


Figure 3-9: bhvSurvey functioning in a global polygon mode. In this state, every vehicle patrols the same survey polygon. Vehicles use Algorithm 8 to spread themselves equally along the survey.

Algorithm 8 Global Polygon Speed Selection

INPUT: XYSeglist *survey_points* ,Map<string,NodeRecord> *m_node_records*, XYPoint *m_curr_position*,double *target_vehicle_speed* : **OUTPUT:** double *ordered_vehicle_speed*

```
1: double vehicle_position ← 1
2: double m_total_patroling_vehcles ← 0
3: for all node_record ∈ m_node_records do
4:     if node_record.status() = "Patroling" then
5:         m_total_patroling_vehcles ← m_total_patroling_vehcles + 1
6:         if node_record.getX() > m_curr_position.getX()" then
7:             vehicle_position ← vehicle_position + 1
8:         end if
9:     end if
10: end for
11: double req_survey_distance ← lead_vehicle_distance + (vehicle_position ·
    survey_points.getDistance())/m_total_patroling_vehcles
12: if req_survey_distance < getDistanceFromEndOfCycle() then
13:     return target_vehicle_speed · 1.5
14: end if
15: if req_survey_distance > getDistanceFromEndOfCycle() then
16:     return target_vehicle_speed · 0.5
17: end if
18: return target_vehicle_speed
```

bhvWaypoint `bhvWaypoint` comes bundled as a standard MOOS-IvP library behavior and makes vehicles drive to given waypoints. In this mission, `bhvWaypoint` takes patroling vehicles to and from a refueling depot. When a vehicle must refuel, its `pGas` application sends `bhvWaypoint` updated depot coordinates and activates `bhvWaypoint` via a mode switch. Once active, `bhvWaypoint` guides the vehicle into a depot and transitions the vehicle to station keeping behavior using an endflag posted to the vehicles `MOOSDB`. Once fueled, `uResourceBroker` sends `bhvWaypoint` the vehicle's previous survey interruption point and activates `bhvWaypoint` to guide the vehicle back to patroling. When the vehicle reaches its endpoint, `bhvWaypoint` returns the vehicle to patroling behavior using an endflag posted to the vehicles `MOOSDB`.

Selecting Mission Parameters To Simulate

Producing good simulation data for productivity prediction requires simulating missions over diverse parameter combinations. Further, hardware and time limitations reduce the practical amount of simulatable mission combinations. Consequently, each simulation varied depot quantity, vehicle quantity, and fuel tank size combinations. Specific variations were selected as follows:

Selecting Fuel Tank Sizes Selecting fuel tank size first requires calculating the total distance a vehicle travels in a cycle D_t and the total time required to cover this distance. Calculating D_t requires calculating the total distance traveled in a survey D_s plus the total recycle distance D_r required to reset from the survey end point to the survey beginning.

$$\begin{aligned} D_s &= TotalSurveyRunLength + TotalSurveyTurnLength \\ &= (NumRuns \cdot SurveyRunLength) + (NumTurns \cdot SurveyTurnLength) \\ &= (37 \cdot 5350) + (36 \cdot 150) \\ &= 203,350m \end{aligned}$$

$$\begin{aligned} D_r &= SurveyDiagonalLength \\ &= \sqrt{surveyLength^2 + surveyWidth^2} \\ &= \sqrt{2 \cdot (5350^2)} \\ &= 7566m \end{aligned}$$

$$\begin{aligned} D_t &= D_s + D_r \\ &= 210,916m \end{aligned} \tag{3.1}$$

Estimating the graph idle time rise caused by refueling requires assuming that vehicles always refuel on average from the center of the operating area. Ensuring vehicles have enough fuel to make it to the center of the operating area following a survey requires calculating the expected fuel quantity required F_r to complete a survey and run out half way through the recycling. Given vehicle fuel consumption

$C_v = .0694$ units/sec, F_r can be calculated as follows

$$\begin{aligned} F_r &= \frac{D_s + \frac{1}{2}D_r}{V_{avg}} \cdot C_v \\ &= 719.21 units \\ &\approx 720 units \end{aligned} \tag{3.2}$$

To test how changing vehicle fuel tank size affects overall node idle time, simulations will run with vehicles fuel tanks of size F_r, 50% F_r, and 150% F_r. These fuel tank values will run in missions using both a global patrol and individual polygon patrol schemes to simplify patrol algorithm comparison.

$50\%F_r$	F_r	$150\%F_r$
360	720	1080

Table 3.1: Table showing fuel tank sizes used in simulation

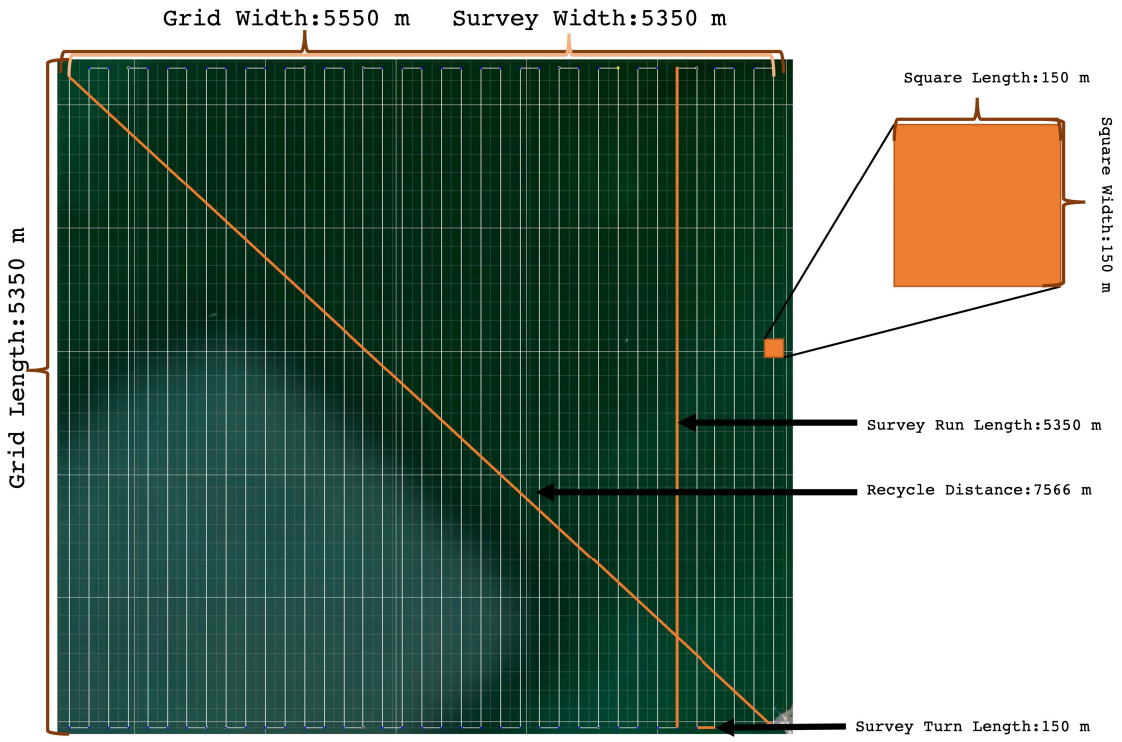


Figure 3-10: Overall mission layout used for simulations with a global patrol polygon.

Selecting Mission Completion Metric Each simulation operating under a "global polygon" patrol scheme operated for ten complete vehicle cycles. A cycle completes when a vehicle completes an entire survey tour. Ten cycles guaranteed vehicles would refuel multiple times during a simulation, vehicles could return to patrol position equilibrium after fueling, and that the vehicles created sufficient performance data samples to train a prediction engine on. Ten cycles also ensured even large vehicle swarm missions could be simulated within two days. Each simulation operating under a "local polygon" patrol scheme operated for 3150 vehicles cycles. In a local polygon patrol scheme, higher vehicle missions create smaller Voronoi polygons. These smaller polygons mean each vehicle patrols less and completes a patrol cycle sooner. Patrolling for 3150 cycles ensures that even thirty vehicle missions produce quantities of performance data equivalent to "global polygon" patrol schemes.

Selecting Quantity of Depots to Vary The large computational energy required to operate large swarm patrol missions make some performance simulations last days.

Maximizing run performance diversity while minimizing this time spent simulating required determining which depot quantity threshold produced meaningful changes to grid idle time and grid average area. Simulating runs containing five, ten, and fifteen vehicles showed missions with greater than five depots produced little meaningful change in vehicle performance (Figure 3-12, 3-11). Consequently, all further supply testing was conducted using missions containing five or less depots.

Figure 3-11: Non-Linear Idle Time Changes Occur Most Within the First 5 Depots [Truncated Y Axis To Highlight Rate of Change]

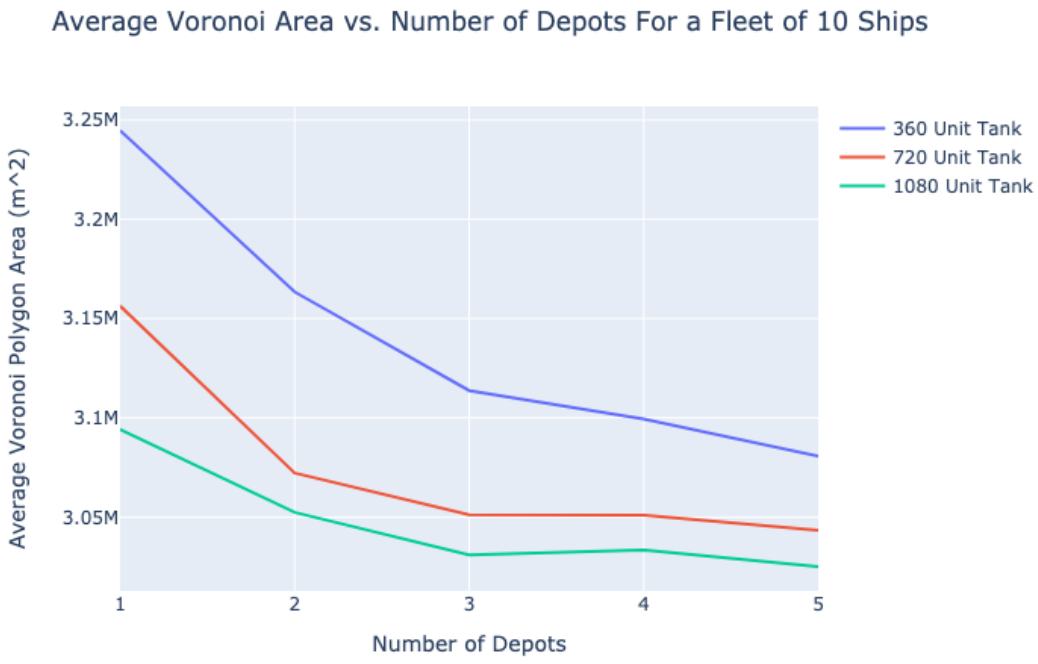


Figure 3-12: Non-Linear Area Changes Occur Most Within the First 5 Depots [Truncated Y Axis To Highlight Rate of Change]

3.3.2 Analytically Validating Mission Performance

The following section demonstrates how to calculate grid average idle time and average Voronoi polygon area as described in Section 3.2 .

Mission Variables For Example Calculations

These calculations assume the following:

1. Any time a boat resides on a square, the squares time since last visit = 0.

2. Vehicles move at a constant speed across every square. *A constant speed assumption means every vehicle takes the same amount of time to cover each square. This constant time allows idle times to be computed with a simple summation.*

3. Vehicles immediately leave the patrol area when fueling is required.

4. Vehicles always reach their fuel minimum order point at the center of the operating area. *When vehicles always refuel at the center of the operating area, they always travel the same round trip distance to refuel at a depot.*

5. Four symmetric depots surround the operating area.

The calculations use the following variables:

Variable	Value	Description
V_{avg}	20 m/s	Vehicle Average Speed
C	0.0694 Units/Sec	Vehicle Consumption Required for Clean Hull 20m/s speed
d_{square}	150 m	Idle Grid Square Size
d_{diag}	7672 m	Operating Area Diagonal Distance
d_{depot}	5350 m	Average Round Trip Depot Distance

Calculating Grid Dimensions Let N_h be the total horizontal grid square count, N_v be the total vertical square count, and N_t be the total grid square count.

$$
\begin{aligned}
N_h &= \frac{GridWidth}{GridSquareWidth} \\
&= \frac{5550}{150} \\
&= 37
\end{aligned}
$$

$$
\begin{aligned}
N_v &= \frac{GridLength}{GridSquareLength} \\
&= \frac{5350}{150} \\
&= 36
\end{aligned}
$$

$$
\begin{aligned}
N_t &= N_h \cdot N_v \\
&= 36 \cdot 37 \\
&= 1332 Squares
\end{aligned}
$$

Example: Calculating No Fuel Grid Average Idle Time Define the following variables:

S	# of Empty Squares
B	# of Boats
T	Time Boat Spends on Each Square
I_{PAvg}	No Fuel Grid Average Idle Time

$$
\begin{aligned}
S &= \frac{(N_{square} - 1)}{B} \\
&= \frac{(1332 - 1)}{10} \\
&= 133.1
\end{aligned}
$$

$$
\begin{aligned}
T &= \frac{d_{square}}{V_{avg}} \\
&= \frac{150}{20} \\
&= 7.5
\end{aligned}
$$

$$
\begin{aligned}
I_{PAvg} &= \sum_{i=1}^{S} \frac{iT}{S} \\
&= \frac{T}{S} \sum_{i=1}^{S} i \\
&= \frac{T(S+1)}{2} \\
&= 502.12
\end{aligned}
$$

Example: Calculating the Total Average Grid Idle Time Caused During Patrolling, Recycling, Refueling Define the following variables:

F_p	Fuel Consumed During a Patrol
P	Avg # of Refuelings Per Patrol
F_t	Time to Refuel and Return to Patrol
I_{FAvg}	Avg Fueling Delay
$I_{Recycle}$	Avg Recycle Delay
I_{Total}	Grid Average Idle Time With Refueling

$$
\begin{aligned}
F_P &= (Squares/Patrol) \cdot (OneSquareTravelTime) \cdot (HourlyConsumptionAtV_{avg}) \\
&= \frac{N_{square}}{B} \cdot \frac{d_{square}}{V_{avg}} \cdot C \\
&= \frac{1332}{10} \cdot \frac{150}{20} \cdot 0.06944 \\
&= 69.37
\end{aligned}
$$

$$
\begin{aligned}
P &= \frac{F_P}{Q_{tank}} \\
&= \frac{69.37}{720} \\
&= 0.0963
\end{aligned}
$$

$$
\begin{aligned}
F_T &= \frac{d_{depot}}{V_{Avg}} \\
&= \frac{5350}{20} \\
&= 267.5
\end{aligned}
$$

$$
\begin{aligned}
I_{FAvg} &= P \cdot F_T \\
&= 25.8Sec
\end{aligned}
$$

$$
\begin{aligned}
I_{Recycle} &= \frac{d_{diag}}{V_{Avg}} \cdot \frac{1}{B} \\
&= 38.36Sec
\end{aligned}
$$

$$
\begin{aligned}
I_{Total} &= I_{PAvg} + I_{FAvg} + I_{Recycle} \\
&= 566.25Sec
\end{aligned}
$$

Example: Calculating Average Voronoi Polygon Area Average Voronoi polygon $A_{VPTotal}$ area can be calculated by simply averaging the time weighted average Voronoi polygon areas when a vehicle is patroling $A_{VPPatrol}$ and not patroling A_{VPN}. Given $A = TotalOperatingArea$

$$A_{VPPatrol} = \frac{A}{B}$$

$$A_{VPN} = \frac{A}{(B-1)}$$

$$A_{VP} = \frac{I_{PAvg} \cdot A_{VPPatrol}}{I_{Total}} + \frac{(I_{FAvg} + I_{Recycle}) \cdot A_{VPN}}{I_{Total}}$$

3.3.3 Comparing Analytically Performance Estimates With Observed Results

The following tables show analytical vs observed Average Grid Idle Time and Voronoi Polygon area for several different mission scenarios that most closely resemble analytical calculations. Specifically, all missions compared have four depots in order to closely approximate the analytical solution's round trip refueling distance. As previously stated, these results don't prove the resource management algorithms create the smallest possible mission disruptions when fueling robots. However, the results validate that the mission simulation performs as expected and benchmarks the gap between different calculations and real world performance.

Grid Average Idle Time

No Fuel Observed and Calculated Average grid idle time remain almost identical regardless of the number of vehicles in the scenario. This lack of difference indicates that `pLatencyGridReporter` successfully captures vehicle activity and that robots in a no fuel patrol state can be approximated by simple calculations. By contrast, experimental vehicle cycle restart time varies appreciably from its analytical calculation. Grid sampling frequency likely causes most of this difference. Since vehicles recycle quickly, `pLatencyGridReporter` does not run fast enough to capture the small and instantaneous disruptions caused by vehicle recycling.

# Vehicles	I_P Calc	I_P Obs	% Diff	$I_{Recycle}$ Calc	$I_{Recycle}$ Obs	% Diff
5	1002	1003	0.099	76	65	14.4
10	502	504	0.398	38	27	28.9
15	336	336	0	25	22	24

Table 3.2: Avg Grid Idle Times for Scenarios W/ 4 Depot, Vehicles W/ 720 Unit Fuel Tank.

Average Voronoi Polygon Area With Fueling

Calculated and Observed Average Voronoi Polygon Area remain almost identical regardless off the number of vehicles in the scenario. This lack of difference indicates that `uFldVoronoi` successfully captures vehicle activity. Further, it indicates that the average Voronoi polygon disruptions caused by refueling can be approximated using simple calculations.

# Vehicles	A_{VP} Calc	A_{VP} Obs	% Diff
5	6185961.3	6108062.0	1.2
10	3035396.3	3051079.6	0.51
15	2014661.0	2030085.9	0.76

Table 3.3: Avg Voronoi Area for Scenarios W/ 4 Depot, Vehicles W/ 720 Unit Fuel Tank.

3.3.4 Post Processing Simulation Data

This section described the programs and processes required to log, clean, store, and display simulation data. All summary statistics, time series figures, and summary histograms can be found in chapter 4.

Processing and Displaying Logged Simulation Data

Properly analyzing mission productivity data requires each MOOS community log multiple MOOS variables throughout each simulation. To accomplish this logging, each community runs the MOOS app pLogger. pLogger logs moos variables in plain-text files. Following mission completion, a Python MOOS log processor parses the text logs[Tur]. Other Python functions connected to this processor then convert desired MOOS variables to their own Pandas data frames. This processing framework creates data frames for instantaneous fuel level, instantaneous average grid idle time, instantaneous average Voronoi polygon area, and refueling location data. These data frames can then be read by a supply visualization application.

The supply visualization application ingests Pandas data frames for all simulated depot, tank, and ship combinations to output summary statistics and figures in a Python Dash page. Specifically, for each data frame, supply visualization eliminates all data logged before or after patrol start and converts each time series data frame to human readable datetime format. Supply visualization then create figures from each frame showing individual ship fuel level, instantaneous average grid idle time, instantaneous average Voronoi polygon area size, and a plot showing geographically where vehicles left patrol to refuel. It also creates histograms showing which instantaneous grid idle times and polygon areas occurred most frequently during a mission. Finally, supply visualization creates a table summarizing overall average grid idle time and Voronoi polygon area for all missions. Figure 3-13 shows the Supply Visualization tool dashboard. For layout and printing purposes, all graphs generated in the dashboard are reformatted for print as "png" files with standard white backgrounds.

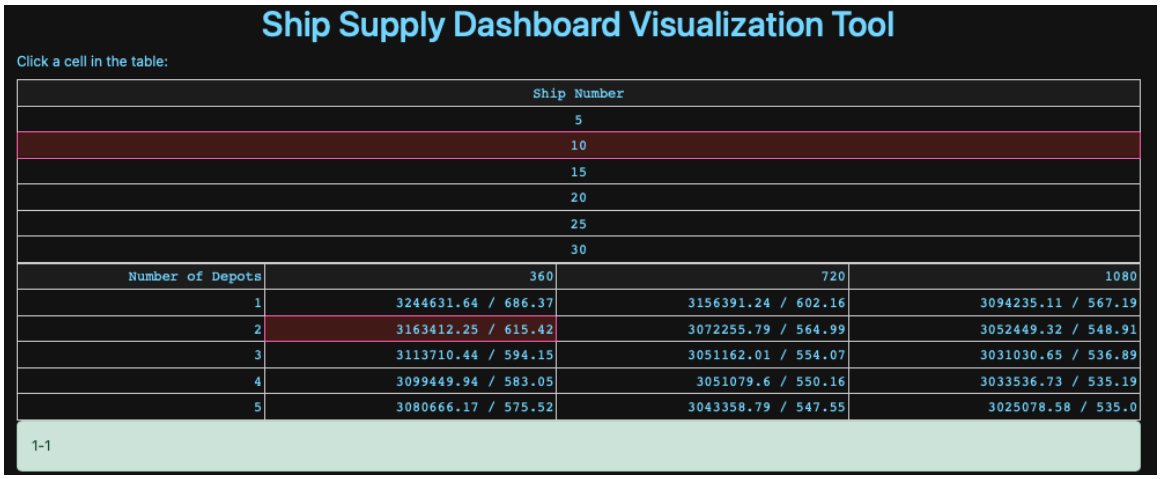


Figure 3-13: Top of the Ship Supply Visualization dashboard. Clicking on any of the summary statistics displays specific time series an histogram data for the selection.

Interpreting Visualization Figures

Summary Statistics Table Each summary statistics table represents data collected over missions containing a given quantity of patrol vehicles using a given patrol algorithm. The table's rows contain summary data collected for missions containing a given quantity of depots. The tables columns summarize data collected for a given fuel tank size. For example, in the figure below, a mission with 10 ships, 4 depots, a 720 unit sized fuel tank size, and operating in a global polygon patrol algorithm had grid average idle time of 550.16 seconds and Average Voronoi Polygon Area of $3051079.6m^2$.

Avg Voronoi Area/Grid Avg Idle Time for 10 Vehicles in Global Polygon Patrol

Number of Depots	360 Unit Tank	720 Unit Tank	1080 Unit Tank
1	3244631.64 / 686.37	3156391.24 / 602.16	3094235.11 / 567.19
2	3163412.25 / 615.42	3072255.79 / 564.99	3052449.32 / 548.91
3	3113710.44 / 594.15	3051162.01 / 554.07	3031030.65 / 536.89
4	3099449.94 / 583.05	3051079.6 / 550.16	3033536.73 / 535.19
5	3080666.17 / 575.52	3043358.79 / 547.55	3025078.58 / 535.0

Units: m^2/Seconds

Time Series Figures All time series figures plot readings for a mission with specific patrol algorithm, ship, depot, and tank values. The vehicle fuel level figure plots units of fuel for each vehicle in a given mission over time. Each colored trace represents a vehicle name.

Figure 3-14: Time Series Plot Showing Changing Vehicle Fuel Levels Over Time

Histograms The supply visualization tool also generates histograms that model the frequency of grid average idle time (Figure 3-16) and average Voronoi polygon area (Figure 3-15. For example, in the figure below, a mission with 10 ships, 4 depots, a 720 unit sized fuel tank size, and operating in a global polygon patrol algorithm most commonly saw idle time around 525 seconds and average Voronoi polygon area around 3M m^2.

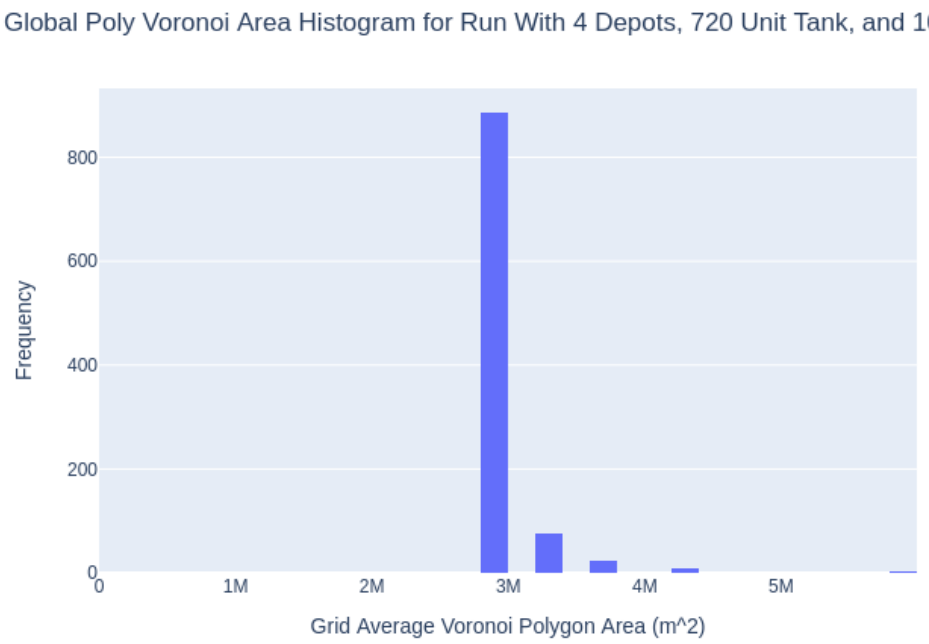


Figure 3-15: Average Voronoi Polygon Area Histogram

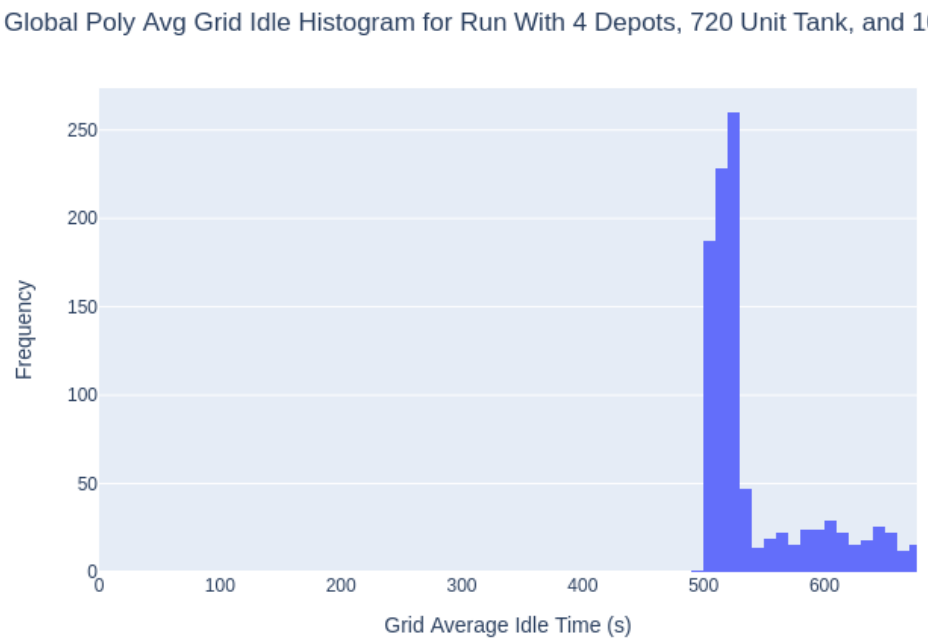


Figure 3-16: Average Grid Idle Time Histogram

Chapter 4

Predicating Future Swarm Needs from Past Mission Performance

The last section implemented and tested two vehicle patrol algorithms. It then established a framework to evaluate how these patroling vehicles perform when they need to periodically refuel. Finally, the last section developed and validated naive ways to fuel patroling vehicles.

Testing these resource constrained patrol strategies required significant computational effort. Testing a single mission with specific depot quantities, vehicle quantities, and vehicle fuel tank sizes required starting an entire MOOS-IvP mission, running the mission to completion, and post processing the raw mission data. Running a mission requires dedicated computing hardware and takes hours to complete. Section 4 focuses on how to use this simulated mission data to rapidly predict future mission performance.

Rapidly predicting future mission performance requires analyzing the simulated data's structure and attributes. This structure can then illustrate what attributes best highlight strong mission performance. Finally, a machine learning algorithm can be implemented to rapidly predict these attributes.

4.1 Prominent Patrol Data Attributes

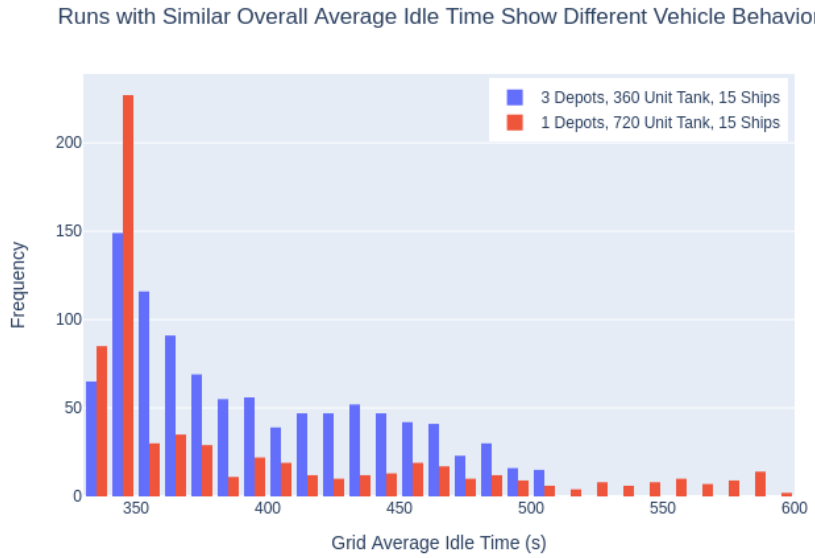
4.1.1 Averaging Overall Performance Allows General Mission Comparison but Doesn't Fully Capture Vehicle Behavior

As previously shown, the Ship Supply Visualization Dashboard (Figure 3-13 summarizes total mission performance using just two metrics. Specifically, the dashboard averages time sampled instantaneous grid idle time and instantaneous average Voronoi polygon area to display overall average idle time and polygon area. These metrics provide easy ways for a policy maker to roughly determine what resources they need to achieve a specific performance level. Further, they allow a planner to rapidly ascertain which parameter changes most affect mission performance. For example, examining averages from global polygon patrols missions show that ship quantity linearly reduces grid idle time but not all depot quantity changes linearly reduce idle time.

While these metrics broadly show how top line mission parameters (e.g depot quantity) change average performance, they fail to capture how reliably a vehicle can maintain this average during a mission. Further, they don't capture how great each performance metric deviates from its average when vehicles must refuel.

Figure 4-1 demonstrates a case where these overall performance averages fail. In this example, two different missions share nearly identical overall grid idle time and Voronoi polygon area values. However, each missions's performance histograms indicate each mission achieved this average with vastly different vehicle behavior. In the first mission (blue traces), vehicle fueling behavior ensures grid idle time never rises above 500 seconds and instantaneous average polygon area never rises above 3M m^2. In the other missions (red traces), large fuel tanked vehicles spend far more time patrolling but occasionally must all leave and fuel at a single depot. This behavior normally maintains area and idle time low but occasionally allows them to spike significantly higher than the first mission. During operations like adversarial

patrol, mission planners may not be able to deploy a mission that occasionally leaves a grid completely unpatrolled, even if it generally produces favorable mean performance. Consequently, any future performance prediction method must provide a mission planner with some indication of underlying vehicle behavior.



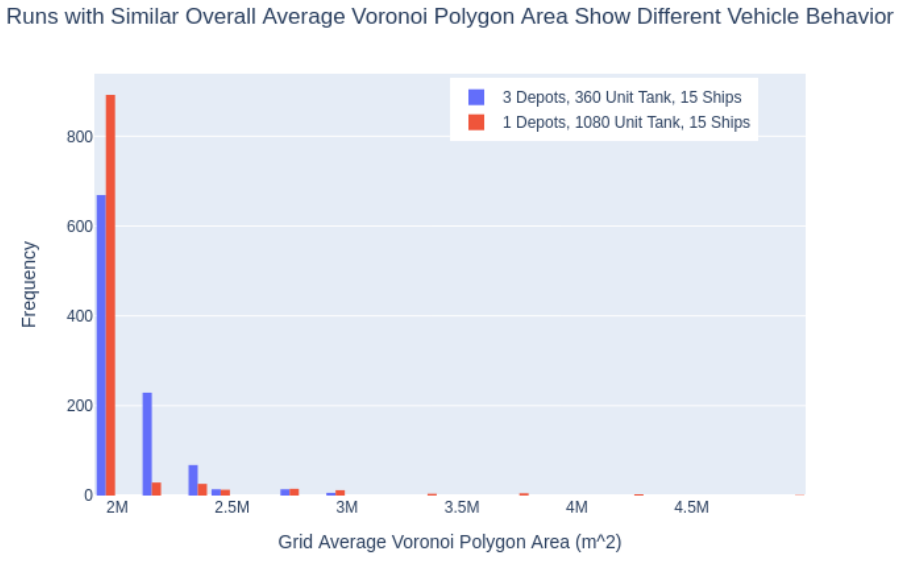
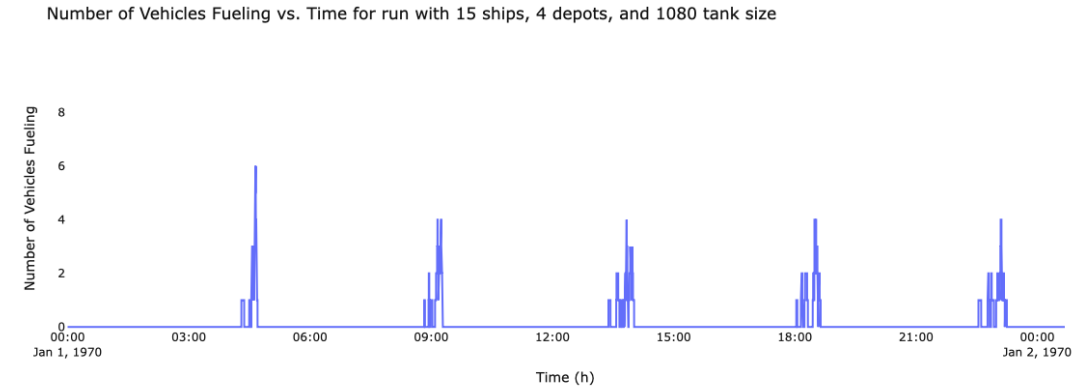


Figure 4-1: Histograms comparing vehicle performance between missions

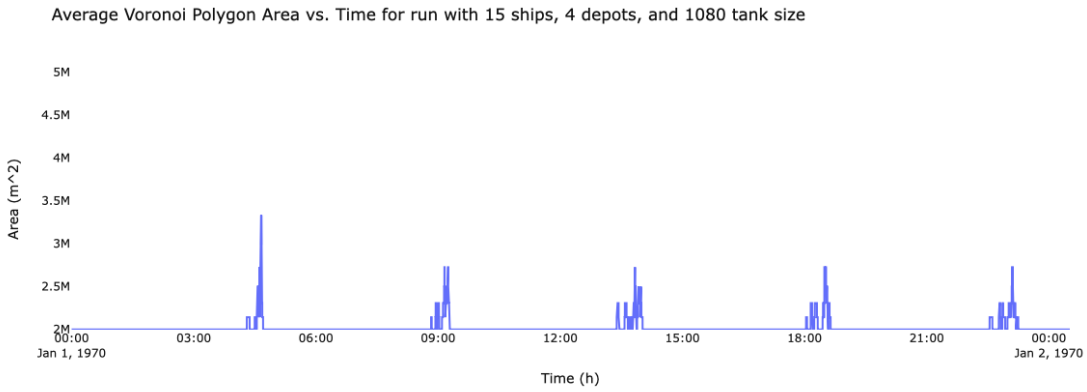
4.1.2 Instantaneous Performance Becomes Difficult to Predict As Vehicle Refuel

Despite running simple patrol algorithms under weather free conditions, vehicle fueling alone introduces uncertainty into each model. For example, Figure 4-2 shows a mission where vehicles run large fuel tanks. Further, this mission also surrounds the operating area with depots. Together, these two design parameters ensure that vehicles rarely need to refuel and experience minimal depot travel disruption when they do fuel. Consequently, vehicles in this mission generally run out of fuel at the same time, take the same amount of time to fuel, and maintain this consistent fueling period throughout the mission. This consistent fueling period ensures grid area and idle time plots follow a fairly cyclic and predictable pattern.

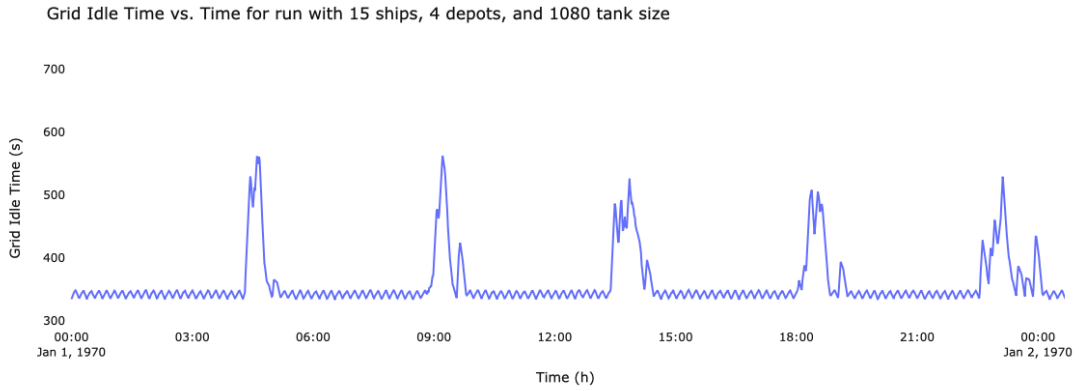
By contrast, Figure 4-3 shows a mission whose performance metrics quickly become unpredictable. In this mission, vehicles run small fuel tanks and must travel varying distances to refuel at a single depot. This distance delays some vehicles from refueling quickly. Consequently, vehicles begin refueling at different intervals. These uneven intervals introduce randomness into grid area and idle time plots. These plots cannot be reproduced or approximated with simple math equations. Any future prediction method must be able to handle this uncertainty.



(a)

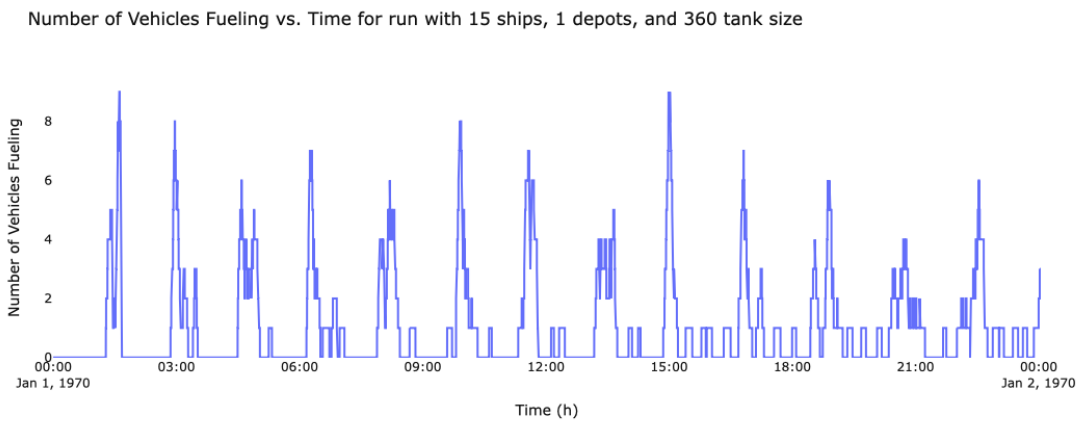


(b)

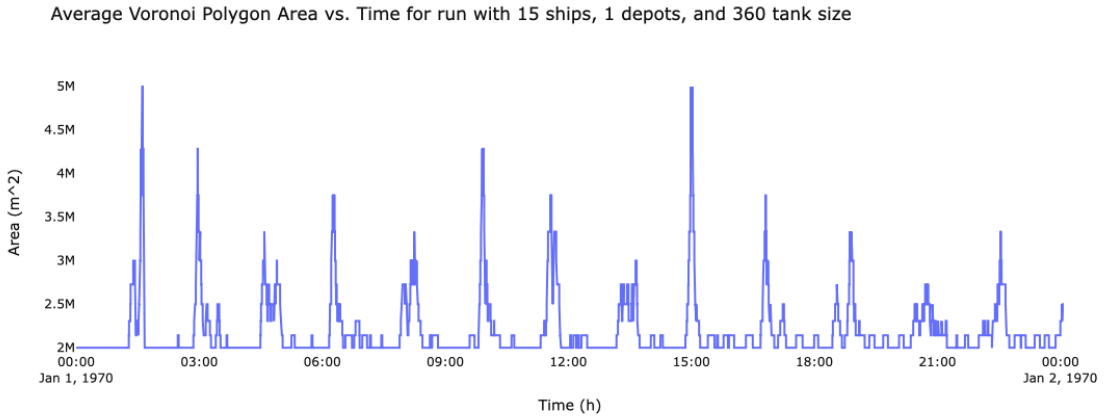


(c)

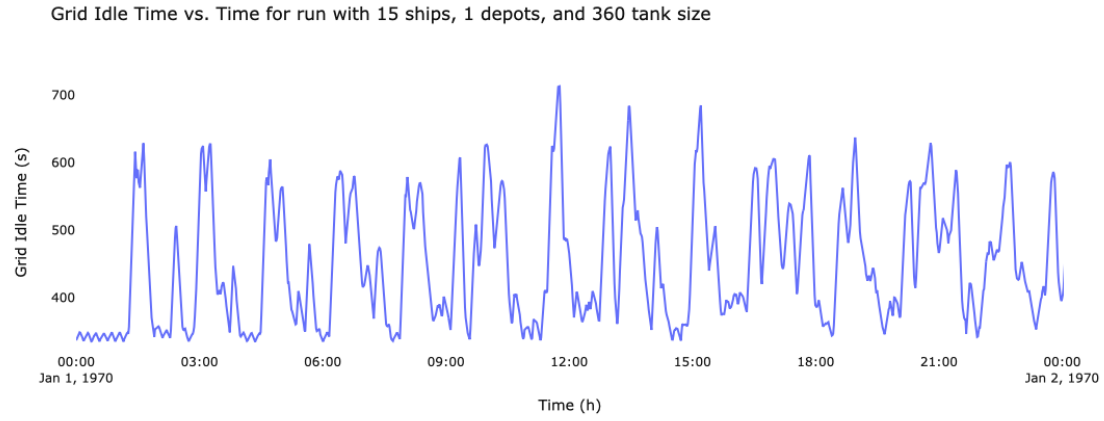
Figure 4-2: Infrequent fueling and small fuel delays make mission performance metrics relatively easy to predict.



(a)



(b)



(c)

Figure 4-3: Frequent fueling and uneven fuel delay introduce randomness into mission performance metrics

4.2 Testing Machine Learning Algorithms to Predict Mission Performance

The simulation data revealed missions display complex, non linear mission behavior whose performance cannot be easily modeled by simple math equations. This section tests two methods to approximate this behavior and display it in a usable manner to a mission planner.

4.2.1 Time Series Transformer

Transformers utilize deep learning technology to infer and predict future patterns in data given prompts or past data trends. Invented by a team of Google researchers in 2017, transformers utilize an attention system to infer and represent information based on a sequence of input items and their orientation within that input sequence. These sequences can be used to then map a query to an output [Vas+17]. Transformers unique attention architecture allow them to predict future data given only a robust training dataset. Companies such as Open-AI and Google have created models using a transformer's predictive abilities to summarize emails, answer questions and even pass the BAR exam[Ope23] [Dev+18].

Transformers could predict vehicle mission performance by using information inferred from the time series idle time and area data generated from a mission. In this method, a transformer would train on a performance value, the previous performance value, the values time, and the static features (e.g depot/ship quantity) contributing to each value. Once trained, the model could then predict future time series performance values given a time and set of unseen static features. These predicted values could then be either read or given to a mission planner in histogram form to show future vehicle performance.

Despite their incredible inference power, transformer prediction models struggle to predict mission performance for several major reasons. First, transformer's signifi-

cant resource requirements limit what models can be run on consumer hardware. For example, transformer memory requirements scale quadratically with input sequence length [Vas+17]. Further, most of this memory must be located on a computer's GPU since the GPU acts as the primary computation engine to train the model. These memory constraints limits how far back a transformer can look when inferring from a value, how many features a transformer can use when inferring from a value, and how much time series context a transformer can use to predict future performance values. After training a model using Huggingface's univariate time series transformer for prediction[RR22], the model was unable to accurately predict mission performance. Further, computational limitations limited the quantity of performance data predictable by the model. Improving this performance would likely require training a multivariate time series model, using a less memory intensive transformer implementation, and allocating more system resources for model training.

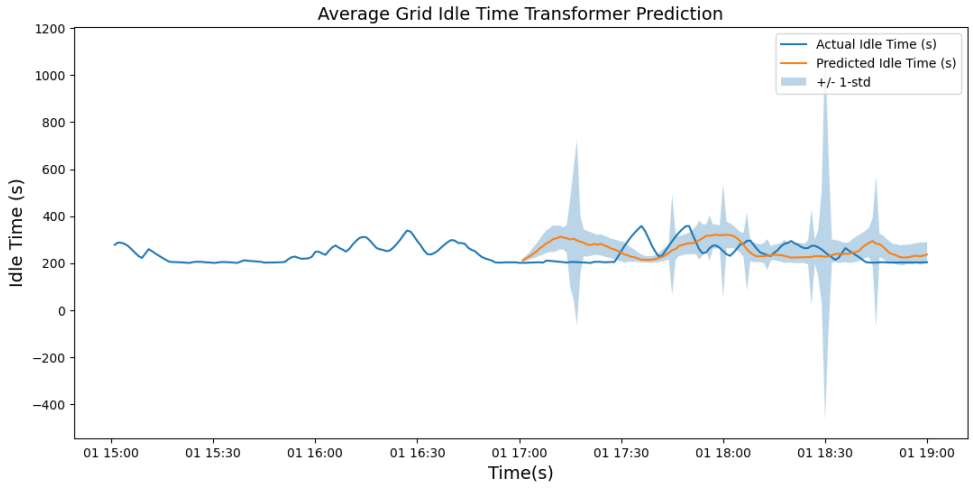


Figure 4-4: Transformers Struggle to Predict an Idle Time Series Run

4.2.2 *Selected Metric:* Predicting Histogram Confidence

As previously stated, uneven fueling delay prohibits directly predicting mission performance as a time series. However, when predicting future vehicle resource needs, planners likely only need to know how their vehicles *usually* perform instead of know-

ing *exactly* how vehicles perform at a given time. Confidence intervals help describe how vehicles usually perform.

Confidence intervals represent bounds where a certain percent of mission values lie. In other words, confidence intervals bound how bad and good a vehicle will perform in a mission. For example, in a mission with a 95 percent grid idle time confidence interval of 302-500 seconds, instantaneous grid idle time remains between 302 and 500 seconds for 95 percent of the mission. Confidence intervals excel as vehicle behavior descriptors for several reasons. First confidence intervals intuitively explain complex vehicle behavior. Since each interval's unit of measurement matches the underlying performance metric's units, confidence intervals can be easily explained to people without a statistics background. Besides simplicity, planners can also tailor confidence intervals to their specific needs. For example, a planner may choose an 80 percent vice 95 percent confidence interval if they care less about extreme data points and more about where mission performance metrics frequently fall. Finally, confidence intervals excel at describing vehicle behavior because they can be predicted on consumer grade hardware without complex GPU's or excessive memory. Generally, confidence intervals for a given performance metric can be predicted as follows:

1. Simulate m missions with diverse depot, tank, and ship combinations.
2. Select n representative performance samples from each dataset. For example, a single sample might consist of an idle time, depot quantity, ship number, and fuel tank size.
3. Select an ML model capable of quantile regression.
4. Train 3 models on $n \cdot m$ samples using the following optimization scheme.
    (a) 1st Model: Optimize with quantile regression to predict the lower bound confidence interval
    (b) 2nd Model Optimize with RMSE loss function to predict a most likely performance value

(c) 3rd Model: Optimize with quantile regression to predict the upper bound confidence interval

(d) Test model using mission combinations unseen by model. Compute root mean squared error RMSE for predictions.

Selected ML toolkit: XGBoost The software library XGBoost allows users to create highly accurate predictive and regressive models over large datasets based on the methods proposed in [Fri01]'s paper. Specifically, XGBoost simultaneously applies both boosting and regularization to prediction trees in order to find optimal model weights capable of accurate future predictions. In boosting, XGBoost uses a given optimization function to generate residual error trees. XGBoost then weights these residual trees and incorporates the trees into more training rounds. Adding residual trees to the model improves the overall model prediction accuracy [Com]. In regularization, XGBoost uses a function to penalize complex tree structures. Penalizing this complexity ensures XGBoost doesn't over fit its model to training data [Com]. XGBoost can predict upper and lower confidence intervals by training with a cosh objective function. Similar to a quantile regression objective function used in linear regression, cosh roughly approximates a rotated mean absolute error function (MAE) allowing XGBoost to optimize to a specific confidence interval [Gui20]. Unlike a quantile regression objective function, cosh also maintains a continuous 2nd derivative as required by XGBoost.

XGBoost Prediction Results

Predicting confidence intervals required training three models (under prediction, likely prediction, over prediction) on two different performance metrics (Grid Idle time and Voronoi Polygon Area) for both unique mission patrol types (Global and Local polygon patrol). Each of the twelve models train on 60,000 samples captured from 60 missions. Each model was then used to predict a corresponding or 95 percent confidence interval for 30 unseen missions. Tables 4.1, 4.2, 4.3, and 4.4 show these results.

RMSE_UP	RMSE_AVG	RMSE_median	RMSE_OP
12.09	23.52	26.9	57.23

Table 4.1: Root Mean Squared Error (RMSE) (Units: s) When Predicting local polygon idle time performance

RMSE_UP	RMSE_AVG	RMSE_median	RMSE_OP
11.1	17.56	21.92	42.46

Table 4.2: Root Mean Squared Error (RMSE) (Units: s) When predicting global polygon idle time performance

RMSE_UP	RMSE_AVG	RMSE_median	RMSE_OP
25392.66	38908.48	105808.44	289682.93

Table 4.3: Root Mean Squared Error (RMSE) (Units: m^2) when predicting global Voronoi polygon area performance

RMSE_UP	RMSE_AVG	RMSE_median	RMSE_OP
44182.14	70485.85	153602.2	167361.05

Table 4.4: Root Mean Squared Error (RMSE) (Units: m^2) when predicting local Voronoi polygon area performance

Interpreting Results Tables 4.1, 4.2, 4.3, and 4.4 show XGBoost's root mean squared error when predicting confidence intervals for performance parameters in various mission types. For example, table 4.3 shows that, over thirty unseen missions, XGBoost could predict the worst grid case idle time seen during 95 percent of a mission with an average error of 57 seconds. Further it could predict the best case grid idle time seen during 95 percent of the mission with an error of 12 seconds. Also, over thirty unseen missions, the table shows XGBoost could predict the median and

average grid idle time with an average error rate of 23 and 26 seconds respectively. Similarly, 4.4 show that over thirty unseen missions, XGBoost could predict the worst and best case Voronoi polygon area seen during 95 percent of a mission with average errors of $.04km^2$ and $.167km^2$ respectively. For more granularity, the tables in Appendix A show XGBoost's prediction performance and performance percent differences for each of the individually predicted missions.

Generally, XGBoost can predict missions involving global polygon patrol with a lower root mean squared error (Tables 4.2, 4.3) than missions involving local polygon patrol (Tables 4.1, 4.5). This performance benefit likely comes from the error introduced by uneven polygons in a local patrol scheme. Specifically, the decentralized Voronoi spreading algorithm used in a local patrol scheme cause each vehicle to patrol polygons of different sizes. These differing sizes mean that vehicles patrolling abnormally large or small polygons can unpredictably raise or lower a grid's idle time and Voronoi polygon area due to their uneven patrol frequency and uneven refuel distance.

Generally, these results show XGBoost can predict a missions performance with reasonable accuracy. In many cases, XGBoost's performance predictions differ from actual performance metrics by less than ten percent. In some cases, XGBoost can predict mission performance to within one percent of actual simulated values. While not perfect, these predictions can easily demonstrate a mission's character and provide operational planners needed mission insight far faster than simulation alone. *In other words, XGBoost can predict a mission's behavior in a matter of minutes with a low error rate.*

4.3 Practical Mission Planning Using ML Assisted Simulation

Section 4.2.2's results demonstrate that XGBoost can predict a mission's average performance metrics and performance confidence intervals with reasonable accuracy. While these models are critical to illustrate vehicle behavior, they do not allow a planner to estimate how many depots and vehicles they need to hit a broad performance category. However, combining overall average mission performance with XGBoost's predictive capabilities allows a planner to estimate both resource needs while tailoring the resources to produce a specific vehicle behavior. This combination can be implemented using the following method:

1. Determine the maximum resources available for a given swarm mission.

2. Simulate missions with extreme parameter combinations (max/min depots, vehicles etc) and average each mission's overall performance to broadly estimate mission resource needs.

3. Build XGBoost model from missions that can be simulated rapidly. Estimate model RMSE.

4. Input Resource estimates into XGBoost models to obtain predicted vehicle behavior.

5. Given model error rate, select specific mission parameters based on desired vehicle behavior.

4.3.1 Example: Planning a Saxis Survey Mission

In this example, a malfunctioning chemical factory leaks hazardous chemicals into the Saxis River Basin. The government tasks a survey company to continuously sample water quality throughout the basin to track and model chemical spread. The companies' model remains most accurate when samples are taken less than 10 minutes

apart. Saxis maritime traffic density prohibits more than 30 vehicles from surveying at one time. While the company deployed all 30 vehicles to survey, they want to reduce their vehicle operating count to minimize costs while maintaining a high model accuracy for government review.

1. *Determine Max Resources Available:* The survey company can deploy 30 sample vehicles into the Saxis Basin and maintain them with a maximum of 3 floating supply depots.

2. *Simulate Extreme Combinations:* The survey company simulates the following mission combinations:

   (a) 30 Vehicles, 1 Depot

   (b) 30 Vehicles, 3 Depots (This "simulated" data can come from actual real time results)

   (c) 20 Vehicles, 1 Depot

   (d) 20 Vehicles, 3 Depots

   (e) 10 Vehicles, 1 Depot

   (f) 10 Vehicles, 3 Depots

   The company reviews the simulation data (Table 4.5) and discovers they can significantly reduce the number of vehicles surveying the chemical spill area while maintaining a 10 minute survey interval.

-	10 Ships	20 Ships	30 Ships
1 Depot	602 Secs	304 Secs	206 Secs
3 Depots	554 Secs	276 Secs	184 Secs

Table 4.5: Example data showing overall time between vehicle samples in the saxis operating area.

3. *Build XGBoost Model From Rapid Simulation:* The company conducts many rapid simulations of 5-10 vehicle missions at 20x real time and then trains XGBoost models from their performance results. The company then tests their

model with a 95 percent confidence interval and obtains a 40 second root mean squared error.

4. *Predict Behavior from Resource Estimates* Based on the performance trends from step 2, the company estimates they will likely need between 10 and 15 ships to patrol the operating area. The company then uses their trained XGBoost model to predict mission performance for all 15 different mission combinations.

5. *Select Best Performing Mission:* Since the survey company wants to maintain time between surveys in all parts of an operating area less than 10 minutes (600 seconds), they need to find a the lowest quantity of vehicles and depots whose predicted 95 percent performance plus prediction RMSE remains below 600. In other words, with a 40 second RMSE, they need to find the smallest mission combination that delivers a predicted performance rate below 560 seconds. Based on predictions, the company determines that 12 vehicles and two depots can represent the lowest vehicle combination capable of surveying the operating area with satisfactory performance.

Chapter 5

Conclusion

5.1 Reflection On Thesis Goals

This thesis aimed to maximize collaborative swarm productivity, by predicting and managing robot resource needs, using operations theory, simulation, and machine learning. To achieve this goal, chapter two investigated what common swarm traits indicate high productivity. This investigation found that measuring how much robots spread out and move during tasking can illustrate overall swarm productivity. To track spreading, this thesis proposed measuring a swarm's average Voronoi cell area. To track movement, this thesis proposed measuring a mission area's time sampled average node idle time. This thesis then applies these metrics to a modified Multi Robot Patrol (MRPP) problem that forces ships in a swarm to move constantly, consume abundant amounts of resources, and work under a decentralized control scheme.

Chapter three adds naive resource awareness to the MRPP productivity scenario. Specifically, the chapter outlines a naïve restocking strategy to maintain robots fueled, implements this strategy as a collection of decentralized patrol robots performing a simulated MRPP in MOOS-IvP, and analytically validates that this simulation performs as expected. Following analytical validation, the section then describes specific mission parameters chosen to best highlight how the robot's behavior impacts

resource management strategies and overall mission performance.

Chapter four begins by highlighting how dynamic robot behavior impacts naïve resource management strategies and overall mission performance. Specifically, it shows how when vehicles control their own naïve refueling strategies, they begin transiting to refuel from different locations in the operating area. This location difference changes when each vehicle refuels, when it returns to patrol, and when it must refuel in the future. These refuel time changes preclude easily calculating future vehicle performance. Thesis Chapter four then details how to use XGBoost with an approximated mean absolute error function (MAE) to predict vehicle performance with confidence intervals, thereby improving simple averaged performance predictions without the need for full simulation.

Combined, these chapters describe this thesis' three major contributions to marine robotics. Specifically, this thesis develops

1. A low communication resource aware patrol algorithm using many ships operating in a constrained marine environment.

2. Tangible software modules, testing strategies, and data processing modules to evaluate how this dynamic robot behavior impacts naïve resource management strategies.

3. A novel way to predict future swarm resource needs based on machine learning and past mission simulation.

4. Tangible software modules that implement this resource prediction algorithm using past simulated mission data.

5.2 Future Work

This thesis tested how naive refueling algorithms affect patrolling vehicles ability to continuously cover and visit all portions of an operating area. These naive algorithms

may not best represent how swarms refuel and they almost certainly do not deliver the best patrol performance. By modifying how depots broker fueling, how they communicate with vehicles, and where they could travel, researchers could easily build refueling algorithms that better represent and optimize actual swarm behavior. Further, researchers could also build intelligent fueling behavior into vehicles so they could opportunistically refuel at close points in their patrol. Finally, researchers could implement even greater communication restrictions to determine how degraded communication affects a vehicles ability to refuel.

Besides testing refueling algorithms, this thesis tested if a mission planner could predict future vehicle behavior based on past mission simulations. While simulations accurately show how vehicles respond in ideal conditions, they don't incorporate sea state, wind, vehicle traffic, and other stochastic effects. Future testing on operational swarm missions could help determine how mission predictions hold up when subjected to these stochastic effects. Further, operationally testing these predictions could also help determine what level of prediction accuracy a mission planner needs in order to operate a swarm successfully.

Most importantly, any future swarm autonomy researcher should integrate resource awareness and endurance requirements when designing collaborative algorithms and operational swarm systems. As previously stated, most operational autonomous marine robotic missions remain in the research and testing phases. Further, marine robotics competitions like RobotX require vehicles avoid obstacles, manipulate vehicles, and find items in a small operating area [rob]. While these tasks test robot sensing and path planning, they don't test a robot's endurance. Extending these tasks to last for several hours or several days could incentivize researchers to focus more closely on how they sense and manage their resource consumption during competition.

Appendix A

Detailed XGBoost Prediction Results

Depot	Tank	Ship	Pred_UP	Act_UP	%Diff	Pred_Val	Act_AVG	%Diff AVG	%Diff Median
5	360	5	1054.68	1015.29	3.88	1123.44	1105.32	1.64	2.36
1	720	5	1033.75	1012.48	2.10	1102.89	1131.82	2.56	3.15
3	1080	5	1011.21	1006.01	0.52	1058.86	1054.65	0.40	1.46
5	1080	5	1019.78	1005.42	1.43	1053.91	1053.62	0.03	0.92
2	360	10	529.67	512.33	3.39	584.15	603.28	3.17	2.20
3	720	10	510.10	504.77	1.06	542.66	544.50	0.34	3.61
1	360	15	349.27	342.91	1.86	380.39	446.92	14.89	12.48
2	360	15	349.27	341.02	2.42	380.39	407.68	6.69	4.80
3	360	15	349.48	339.40	2.97	380.39	396.51	4.07	0.99
4	360	15	350.79	338.42	3.65	380.39	385.97	1.45	1.56
1	720	15	343.69	336.70	2.07	365.94	390.20	6.22	4.69
5	720	15	338.51	335.72	0.83	357.88	361.30	0.95	3.75
5	360	20	259.79	254.35	2.14	287.51	287.85	0.12	4.88
2	720	20	256.20	253.14	1.21	267.89	277.86	3.58	3.07
3	720	20	255.85	253.03	1.12	267.86	271.00	1.16	3.87
5	1080	20	255.16	252.65	1.00	261.30	261.97	0.25	1.78
4	360	25	209.17	202.54	3.28	236.83	234.05	1.18	10.46
3	1080	25	203.75	201.98	0.88	206.58	209.19	1.25	1.12
4	1080	25	202.83	201.59	0.62	206.55	208.49	0.93	1.27
1	720	30	176.41	167.72	5.19	237.60	197.66	20.21	36.08
1	1080	30	174.22	167.39	4.09	237.58	184.43	28.82	39.96
4	1080	30	168.28	167.20	0.65	172.24	172.34	0.06	2.01

Table A.1: Predicted vs actual local idle time performance comparison(Units: s)

Depot	Tank	Ship	Pred_OP	Act_OP	%Diff
5	360	5	1199.96	1209.54	0.79
1	720	5	1240.54	1382.70	10.28
3	1080	5	1136.66	1169.48	2.81
5	1080	5	1124.86	1169.79	3.84
2	360	10	660.95	728.37	9.26
3	720	10	616.34	662.43	6.96
1	360	15	455.17	585.04	22.20
2	360	15	432.73	502.70	13.92
3	360	15	436.75	485.01	9.95
4	360	15	427.42	466.19	8.32
1	720	15	438.76	541.15	18.92
5	720	15	414.42	448.27	7.55
5	360	20	335.26	357.05	6.10
2	720	20	310.75	354.29	12.29
3	720	20	310.92	333.36	6.73
5	1080	20	290.22	311.48	6.82
4	360	25	276.00	298.86	7.65
3	1080	25	238.27	248.20	4.00
4	1080	25	231.83	252.31	8.11
1	720	30	275.85	293.17	5.91
1	1080	30	256.86	267.10	3.83
4	1080	30	196.33	196.66	0.16

Table A.2: Predicted vs actual local idle time performance comparison (cont)(Units: s)

Depot	Tank	Ship	Pred_UP	Act_UP	%Diff	Pred_Val	Act_AVG	%Diff AVG	%Diff Median
2	360	5	1055.95	1031.45	2.38	1135.55	1192.14	4.75	4.55
3	720	5	1014.14	1007.64	0.65	1078.61	1079.61	0.09	2.01
3	360	10	527.57	508.43	3.76	577.17	583.51	1.08	0.41
5	360	10	527.46	509.65	3.49	577.17	568.40	1.54	3.48
2	1080	10	519.78	504.78	2.97	551.75	536.69	2.81	5.81
3	1080	15	339.66	335.64	1.20	351.58	349.91	0.48	2.31
3	360	20	269.14	253.67	6.10	305.77	294.71	3.76	9.51
4	360	20	261.27	253.31	3.15	288.04	287.32	0.25	7.25
1	720	20	258.05	253.58	1.77	279.41	295.27	5.37	7.72
1	360	25	217.14	203.41	6.75	253.75	285.20	11.03	10.67
3	360	25	207.91	202.68	2.59	234.39	236.35	0.83	7.29
5	360	25	207.60	202.83	2.35	234.38	236.76	1.00	0.76
1	720	25	205.32	202.15	1.57	225.13	235.13	4.26	7.97
5	720	25	204.02	201.82	1.09	213.19	214.68	0.69	4.29
1	360	30	181.25	169.36	7.02	212.05	235.34	9.90	8.68
2	720	30	169.08	167.56	0.91	180.57	190.21	5.07	1.47
3	720	30	169.08	167.41	1.00	176.75	180.06	1.83	4.21
1	1080	30	174.38	167.32	4.22	199.66	186.21	7.22	17.63
3	1080	30	168.39	167.36	0.63	172.25	174.32	1.18	1.68

Table A.3: Predicted vs actual global idle time performance comparison (Units: s)

Depot	Tank	Ship	Pred_OP	Act_OP	%Diff
2	360	5	1246.69	1361.79	8.45
3	720	5	1184.20	1218.98	2.85
3	360	10	652.46	695.49	6.19
5	360	10	639.83	659.98	3.05
2	1080	10	633.21	641.79	1.34
3	1080	15	401.04	412.66	2.81
3	360	20	351.85	375.84	6.38
4	360	20	336.51	355.55	5.35
1	720	20	355.94	405.79	12.28
1	360	25	300.17	383.91	21.81
3	360	25	276.79	304.60	9.13
5	360	25	274.80	293.41	6.34
1	720	25	285.60	329.42	13.30
5	720	25	245.75	264.16	6.97
1	360	30	261.70	310.06	15.60
2	720	30	233.91	247.59	5.52
3	720	30	225.13	229.21	1.77
1	1080	30	246.24	277.27	11.19
3	1080	30	211.33	207.01	2.09

Table A.4: Predicted vs actual global idle time performance comparison (cont) (Units: s)

Depot	Tank	Ship	Pred_UP	Act_UP	%Diff	Pred_Val	Act_AVG	%Diff AVG	%Diff Median
3	720	5	5958582.88	5995000.00	0.61	6134994.51	6091919.17	0.71	2.34
5	720	5	5989101.41	5995000.00	0.10	6085286.62	6085924.17	0.01	1.51
1	1080	5	6083047.39	5995000.00	1.47	6328359.60	6210320.42	1.90	5.56
4	720	10	3011801.00	2997500.00	0.48	3060453.18	3049397.19	0.36	2.10
3	720	15	1996438.98	1998333.33	0.09	2033489.94	2033412.96	0.00	1.76
1	1080	15	2015395.88	1998333.33	0.85	2132350.44	2066187.74	3.20	6.71
5	1080	15	2003254.18	1998333.33	0.25	2023618.70	2018779.81	0.24	1.27
1	360	20	1501840.11	1498750.00	0.21	1567612.65	1614349.36	2.90	0.64
4	360	20	1500742.67	1498750.00	0.13	1538296.10	1542210.75	0.25	2.64
5	360	25	1201406.36	1199000.00	0.20	1231438.64	1229140.74	0.19	2.71
3	720	25	1199424.51	1199000.00	0.04	1215512.63	1219155.75	0.30	1.38
4	720	25	1199109.91	1199000.00	0.01	1213749.29	1216040.34	0.19	1.23
3	360	30	1001974.70	999166.67	0.28	1030889.75	1031500.31	0.06	3.17
1	720	30	996712.21	999166.67	0.25	1029554.49	1042641.92	1.26	3.04
4	1080	30	997902.10	999166.67	0.13	1009987.71	1009183.33	0.08	1.08

Table A.5: Predicted vs actual global area performance comparison(Units: m^2)

Depot	Tank	Ship	Pred_OP	Act_OP	%Diff
3	720	5	6772048.00	7493750.00	9.63
5	720	5	6733738.42	7493750.00	10.14
1	1080	5	7155328.27	7493750.00	4.52
4	720	10	3330888.99	3330555.56	0.01
3	720	15	2207044.84	2305769.23	4.28
1	1080	15	2614740.85	2497916.67	4.68
5	1080	15	2188844.68	2141071.43	2.23
1	360	20	1783293.49	1873437.50	4.81
4	360	20	1683065.65	1763235.29	4.55
5	360	25	1335947.04	1303260.87	2.51
3	720	25	1307933.33	1303260.87	0.36
4	720	25	1300658.58	1303260.87	0.20
3	360	30	1131550.07	1110185.19	1.92
1	720	30	1222660.06	1248958.33	2.11
4	1080	30	1090782.05	1033620.69	5.53

Table A.6: Predicted vs actual global area performance comparison (cont)(Units: m^2)

Depot	Tank	Ship	Pred_UP	Act_UP	%Diff	Pred_Val	Act_AVG	%Diff AVG	%Diff Median
2	720	5	6099487.78	5995000.00	1.74	6353841.30	6190836.67	2.63	5.99
1	1080	5	6062000.75	5995000.00	1.12	6373164.65	6162360.42	3.42	6.31
2	1080	5	6138278.96	5995000.00	2.39	6362501.14	6224308.75	2.22	6.13
1	360	10	3010001.90	2997500.00	0.42	3165569.54	3248469.26	2.55	5.61
4	360	10	3000283.00	2997500.00	0.09	3097269.30	3106730.33	0.30	3.33
5	360	10	3039702.18	2997500.00	1.41	3097980.02	3080121.57	0.58	3.35
3	720	10	3003489.26	2997500.00	0.20	3064356.33	3063123.84	0.04	2.23
4	720	15	2007515.67	1998333.33	0.46	2037473.92	2027564.20	0.49	1.96
4	1080	15	2008477.93	1998333.33	0.51	2020849.23	2021739.38	0.04	1.13
5	360	20	1503065.47	1498750.00	0.29	1542378.19	1534294.46	0.53	2.91
3	720	20	1498857.86	1498750.00	0.01	1521340.73	1520206.32	0.07	1.51
1	1080	20	1515421.99	1498750.00	1.11	1547569.16	1553284.96	0.37	3.26
1	360	25	1205103.64	1199000.00	0.51	1252459.05	1291637.50	3.03	0.28
3	360	25	1183110.83	1199000.00	1.33	1229800.46	1237917.72	0.66	2.57
4	360	25	1189179.66	1199000.00	0.82	1229115.96	1230561.89	0.12	2.51
1	720	25	1201731.09	1199000.00	0.23	1242227.32	1253789.12	0.92	3.61
3	720	25	1198991.18	1199000.00	0.00	1217619.66	1217287.21	0.03	1.55
3	1080	25	1194303.51	1199000.00	0.39	1210191.01	1213733.33	0.29	0.93
5	360	30	1014148.95	999166.67	1.50	1032884.84	1027497.05	0.52	3.37
5	1080	30	999984.68	999166.67	0.08	1008728.03	1009049.49	0.03	0.96

Table A.7: Predicted vs actual local area performance comparison(Units: m^2)

Depot	Tank	Ship	Pred_OP	Act_OP	%Diff
2	720	5	7150909.90	7493750.00	4.58
1	1080	5	7178284.65	7493750.00	4.21
2	1080	5	7049148.08	7493750.00	5.93
1	360	10	3712442.16	3746875.00	0.92
4	360	10	3414295.20	3746875.00	8.88
5	360	10	3418306.11	3330555.56	2.63
3	720	10	3340511.56	3330555.56	0.30
4	720	15	2213905.33	2305769.23	3.98
4	1080	15	2184574.13	2141071.43	2.03
5	360	20	1687332.15	1665277.78	1.32
3	720	20	1644076.82	1665277.78	1.27
1	1080	20	1763687.49	1763235.29	0.03
1	360	25	1430455.80	1498750.00	4.56
3	360	25	1336395.14	1362500.00	1.92
4	360	25	1327599.64	1362500.00	2.56
1	720	25	1419782.28	1498750.00	5.27
3	720	25	1336427.93	1303260.87	2.54
3	1080	25	1288726.21	1303260.87	1.12
5	360	30	1130928.04	1152884.62	1.90
5	1080	30	1067198.04	1070535.71	0.31

Table A.8: Predicted vs actual local area performance comparison (cont)(Units: m^2)

Bibliography

[AKJ05] N. Ahmed, S.S. Kanhere, and S. Jha. "Probabilistic coverage in wireless sensor networks". In: *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)l.* 2005, 8 pp.–681. DOI: 10.1109/LCN.2005.109.

[Bena] Mike Benjamin. *Introduction 2.680.* Image Found on Slide 12. URL: https://oceanai.mit.edu/2.680/docs/2.680-01-intro_to_class_2023.pdf. (accessed: 03.06.2023).

[Benb] Mike Benjamin. *Introduction To MOOS.* URL: https://oceanai.mit.edu/2.680/docs/2.680-03-intro_to_moos_2023.pdf. (accessed: 03.05.2023).

[Ben22] Mike Benjamin. *sea_of_japan_fifty_nodes.* Screenshot of Mission Simulation. Swarm Toolbox Capability Demonstrator. 2022.

[Bor+18] Edward Bormashenko et al. "Characterization of Self-Assembled 2D Patterns with Voronoi Entropy". In: *Entropy* 20.12 (2018). ISSN: 1099-4300. DOI: 10.3390/e20120956. URL: https://www.mdpi.com/1099-4300/20/12/956.

[Che04] Yann Chevaleyre. "Theoretical analysis of the multi-agent patrolling problem". In: *Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004.(IAT 2004).* IEEE. 2004, pp. 302–308.

[Chu+07] Hoang Nam Chu et al. "Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation". In: *19th IEEE international conference on tools with artificial intelligence (ICTAI 2007).* Vol. 1. IEEE. 2007, pp. 442–449.

[Com] Distributed (Deep) Machine Learning Community. *Introduction to boosted trees.* URL: https://xgboost.readthedocs.io/en/stable/tutorials/model.html.

[Cor+08] Fabien Cornillier et al. "A heuristic for the multi-period petrol station replenishment problem". In: *European Journal of Operational Research* 191.2 (2008), pp. 295–305. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2007.08.016. URL: https://www.sciencedirect.com/science/article/pii/S0377221707008910.

[CP98] Howie Choset and Philippe Pignon. "Coverage Path Planning: The Boustrophedon Cellular Decomposition". In: *Field and Service Robotics*. Springer London, 1998, pp. 203–209. DOI: 10.1007/978-1-4471-1273-0_32. URL: https://doi.org/10.1007%5C%2F978-1-4471-1273-0_32.

[Dev+18] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[DMK11] Jason Derenick, Nathan Michael, and Vijay Kumar. "Energy-aware coverage control with docking for robot teams". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 3667–3672. DOI: 10.1109/IROS.2011.6094977.

[EAK07] Yehuda Elmaliach, Noa Agmon, and Gal Kaminka. "Multi-Robot Area Patrol under Frequency Constraints". In: *Annals of Mathematics and Artificial Intelligence* 57 (Apr. 2007), pp. 293–320. DOI: 10.1007/s10472-010-9193-y.

[Eva22] Nicholas Craig Evans. "A Practical Search with Voronoi Distributed Autonomous Marine Swarms". MA thesis. Massachusetts Institute of Technology, 2022.

[For86] S Fortune. "A Sweepline Algorithm for Voronoi Diagrams". In: *Proceedings of the Second Annual Symposium on Computational Geometry*. SCG '86. Yorktown Heights, New York, USA: Association for Computing Machinery, 1986, pp. 313–322. ISBN: 0897911946. DOI: 10.1145/10515.10549. URL: https://doi.org/10.1145/10515.10549.

[Fri01] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. ISSN: 00905364. URL: http://www.jstor.org/stable/2699986 (visited on 04/11/2023).

[Guaa] United States Coast Guard. *COMMANDANT INSTRUCTION 7310.1V, REIMBURSABLE STANDARD RATES*. URL: https://www.uscg.mil/Portals/0/NPFC/docs/7310/CI_7310_1v.pdf?ver=Dp73u9f1G-HaW7eyNMhBwg%5C%3D%5C%3D&timestamp=1638384224069. (accessed: 03.04.2023).

[Guab] United States Coast Guard. *NATIONAL 3rd MATE OF SELF-PROPELLED VESSELS OF UNLIMITED TONNAGE UPON OCEANS OR NEAR COASTAL WATERS § 11.407*. URL: https://www.dco.uscg.mil/Portals/9/NMC/pdfs/checklists/mcp_fm_nmc5_05_web.pdf. (accessed: 03.04.2023).

[Gui20] Saupin Guillaume. *Confidence intervals for xgboost*. Sept. 2020. URL: https://towardsdatascience.com/confidence-intervals-for-xgboost-cac2955a8fde.

[HLH09] Kao-Shing Hwang, Jin-Ling Lin, and Hui-Ling Huang. "Cooperative patrol planning of multi-robot systems by a competitive auction system". In: *2009 ICCAS-SICE*. 2009, pp. 4359–4363.

[JHL21] Katharin R. Jensen-Nau, Tucker Hermans, and Kam K. Leang. "Near-Optimal Area-Coverage Path Planning of Energy-Constrained Aerial Robots With Application in Autonomous Environmental Monitoring". In: *IEEE Transactions on Automation Science and Engineering* 18.3 (2021), pp. 1453–1468. DOI: 10.1109/TASE.2020.3016276.

[Joh] Douglas-Westwood Limited John Westwood Paul Newman. *Ocean Survey - The World Market*. URL: https://www.hydro-international.com/content/article/ocean-survey-the-world-market. (accessed: 03.04.2023).

[Lat91] Jean-Claude Latombe. "Approximate Cell Decomposition". In: *Robot Motion Planning*. Boston, MA: Springer US, 1991, pp. 248–294. ISBN: 978-1-4615-4022-9. DOI: 10.1007/978-1-4615-4022-9_6. URL: https://doi.org/10.1007/978-1-4615-4022-9_6.

[Li+14] Kunpeng Li et al. "An inventory–routing problem with the objective of travel time minimization". In: *European Journal of Operational Research* 236.3 (2014). Vehicle Routing and Distribution Logistics, pp. 936–945. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2013.07.034. URL: https://www.sciencedirect.com/science/article/pii/S0377221713006152.

[Mac+03] Aydano Machado et al. "Multi-agent patrolling: An empirical analysis of alternative architectures". In: *Multi-Agent-Based Simulation II: Third International Workshop, MABS 2002 Bologna, Italy, July 15–16, 2002 Revised Papers*. Springer. 2003, pp. 155–170.

[MTR06] Talita Menezes, Patrícia Tedesco, and Geber Ramalho. "Negotiator agents for the patrolling task". In: *Advances in Artificial Intelligence-IBERAMIA-SBIA 2006: 2nd International Joint Conference, 10th Ibero-American Conference on AI, 18th Brazilian AI Symposium, Ribeirão Preto, Brazil, October 23-27, 2006. Proceedings*. Springer. 2006, pp. 48–57.

[Ope23] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[Qu08] Jilin Qu. "Outlier Detection Based on Voronoi Diagram". In: *Advanced Data Mining and Applications*. Ed. by Changjie Tang et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 516–523. ISBN: 978-3-540-88192-6.

[rob] robotX. *Task Descriptions and specifications version 4.0 (final ... - robonation*. URL: https://robonation.org/app/uploads/sites/2/2019/09/RobotX-2018-Tasks-FINAL_v4.0.pdf.

[RR22] Kashif Rasul and Niels Rogge. *Probabilistic Time Series forecasting with hugging-face transformers*. Dec. 2022. URL: https://huggingface.co/blog/time-series-transformers.

[Sof] Jonathan Shaw Sofia Jones Weixian Lan. *Anomaly Detection using Voronoi k-distance*. URL: https://www.weixianlan.com/projects/voronoi_outlier.pdf. (accessed: 03.04.2023).

[Son+14] Cheng Song et al. "Optimal control for multi-agent persistent monitoring". In: *Automatica* 50.6 (2014), pp. 1663–1668. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2014.04.011. URL: https://www.sciencedirect.com/science/article/pii/S0005109814001411.

[Tra] Bureau of Transportation Statistics. *U.S. Coast Guard Search and Rescue Statistics, Fiscal Year*. URL: https://www.bts.gov/content/us-coast-guard-search-and-rescue-statistics-fiscal-year. (accessed: 03.04.2023).

[Tur] Raymond Turrisi. *MWDataMgr*. URL: https://github.com/raymondturrisi/MWDataMgr. (accessed: 03.05.2023).

[Vas+17] Ashish Vaswani et al. "Attention is All You Need". In: 2017. URL: https://arxiv.org/pdf/1706.03762.pdf.