Energy Efficient Distributed Adaptive Sampling using Networked Autonomous Underwater Vehicles

Mohammad Nadeem
Under the guidance of Mehdi Rahmati and Professor Dario Pompili
Cyber-Physical Laboratory (CPS-Lab), ECE Department, Rutgers University

***Abstract*—Near-real-time water quality monitoring in rivers, lakes, and water reservoirs of different physical variables is critical to protect the aquatic life and to prevent further propagation of the potential pollution in the water. In order to measure the physical values in a Region of Interest (ROI), adaptive sampling is helpful as an energy- and time-efficient technique since an exhaustive search of an area is not feasible. Adaptive sampling using one robot is subject to the many constraints, such as a single point of failure, energy and delay inefficiencies. A robot could run out of energy midway during adaptive sampling–when scanning a large area, the sampling could take a long time even with adaptive sampling. If the robot doing the sampling fails, then the entire data is lost. To rectify these issues, we propose a distributed adaptive sampling algorithm using multiple robots. The algorithm ensures energy efficiency and time efficiency while also ensuring higher accuracy in the measurement by first specifying regions of interest, afterward, allocating a team of robots to search these regions more thoroughly without colliding or redoing work of another robot. By nature, the algorithm also accounts for a robot failure. Experiments are conducted in the Raritan River, New Jersey to evaluate the proposed solution.**

***Index Terms*—Autonomous Vehicles, Distributed adaptive sampling, Underwater Networked Robots, Robot Coordination.**

I. Introduction

Overview: Underwater networks enable various applications such as oceanographic data collection, pollution monitoring, disaster prevention, and tactical surveillance using static nodes or mobile vehicles [1]–[4]. In our case, we need mobile vehicles rather than static nodes in order to traverse the area and measure the differences in the measured values across this given area. Static nodes pose many limitations on data collection. Because of their nature, we would need multiple static nodes in order to accurately reconstruct a map of an environment. Static nodes are not helpful in order to track areas of pollution throughout a body of water either. We will use autonomous vehicles for our data collection [5]. There are multiple classes of autonomy. Fully autonomous vehicles are able to completely control their movement and trajectory without any outside input. On the opposite end, ROVs are completely human controlled. In the middle of these lie semi-autonomous vehicles. Semi-autonomy is a broad scale of measurement. A robot that holds it's heading constant while all other aspects are human controlled is considered semi-autonomous while a robot that controls most aspects of it's motion save one or two (such as obtaining the initial movement path) is also semi-autonomous. We aim to make our robots semi-autonomous in the sense that when the command to conduct sampling is sent to them, they are able to completely conduct sampling on their own and coordinate between themselves automatically. Underwater Remotely Operated Vehicles (ROVs) are used to monitor and investigate conditions in water. For example, ROVs could be used to measure the salinity in regions in a river or it could be used to visually see underwater structures to gain a deeper understanding of marine science. Sampling is the process of obtaining some measurement within the water. For example, a robot can conduct sampling by measuring pH in the river. Other values that can be measured in a body of water include temperature, salinity, conductivity and turbidity. But, sampling is exhaustive if we want to reconstruct an accurate map of the area. For efficiency, we use adaptive sampling which changes the exhaustive degree of search being performed at a given point based on the measured values around that point. Normally, adaptive sampling works by searching an area by identifying points of interest and searching those points more thoroughly. A traditional adaptive sampling method involves doing some predefined path and then narrowing the width of the path in regions of interest. Adaptive sampling is normally used for one mobile vehicle. But, using one vehicle offers issues especially in terms of energy. Each vehicle has limited energy so for a large area, it might not be able to scan it completely even with adaptive sampling. Then, there arises the problems with time since sampling will take a long time. For time critical applications, this method is not helpful.

Motivation: As discussed above, we want to use mobile vehicles (ROVs) to enable data collection in a body of water. ROVs are also manually controlled which raises issues in communication. The issue arises of how to identify regions of interest. The ROV must be sending the data back and then the person has to define regions of interest and move the robot there. There is a lot of human error in this case, not to mention a lot of human effort, especially for multiple vehicles. We want to computationally solve this issue by allowing vehicles to make decisions on their own and amongst themselves.

Our Vision: In order to rectify the aforementioned issues with adaptive sampling, we propose a solution using a team of robots to conduct sampling. We will call this distributed sampling since we are using a distributed team of machines to share a workload. In order to achieve this goal, we will have to create an algorithm in order to distribute the work among the robots. Because we want the robots to be autonomous, we will allow the robots to communicate among themselves using a communication protocol we design.

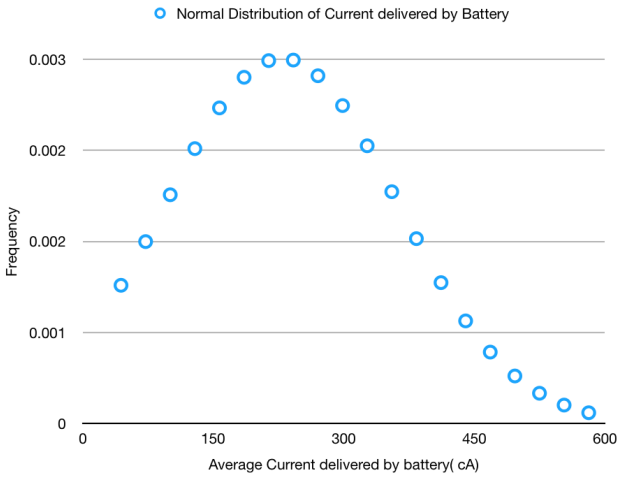


Fig. 1: Current delivered by the battery to the robot.

Contribution: We want to make the robot autonomous in order to conduct adaptive sampling without external input. We are using a team of robots. First, a map of the environment is needed. Second, once a map is obtained, we need to identify regions of interest. Then, these regions of interest must be searched in parallel by the team of robots. We propose networking the robots together in order to achieve the distributed workload without issues like collisions. The algorithm we define must also be energy efficient and time efficient while also allowing a sufficient search of the environment. This paper discusses the communication protocol that allows coordination between the robots, the distributed sampling algorithm and the analysis of the energy and time efficiency of the algorithm.

Report Organization: The remainder of the report is organized as follows. In Sect. II, we go over the state-of-the-art and similar research in the literature, and then define the problem. In Sect. III, we present our proposed solution for underwater distributed adaptive sampling using multiple vehicles and. In Sect. IV, we present the experimental results and discuss the benefits of implementing our solution. Finally, in Sect. V, we draw the main conclusions.

II. Problem Definition

In the underwater environment, searching and investigating water characteristics such as temperature, salinity, acidity, conductivity, pH, dissolved oxygen, and turbidity is a necessity, which motivates applying either data driven or model driven adaptive sampling strategies. There are many issues that arise with adaptive sampling from a technical standpoint. In many applications, the energy budget and the time are limited. Multiple searches of the same area require more energy and obviously require more time. Figure 1 shows that the average current delivered by the battery is around 1.5-2 A which is rather high. Note that this value is independent of whether we use distributed sampling or adaptive sampling. Thus, in order to actually reduce our energy consumption, we will have to optimize other parameters.

We can use a signal reconstruction method to reconstruct the signal in the water given a minimum number of points. Pompili et. al. discuss reconstruction of a map given the minimum number of points using RCS [6]. K-SVD is a dictionary learning algorithm that is used for creating a dictionary from sparse data and is used prevalently in image processing. Thus, we can accurately recreate a map without necessarily commiting to an exhaustive search of the environment. Our aim is to decrease the energy usage. In order to do so, we will need to minimize the distance traveled and the time that the sampling occurs in. The parameters that our algorithm will depend on will be distance to travel, time to travel and current battery levels. Task allocation between multiple robots will use these parameters to optimize the energy used between robots. For example, if two robots are considering an area to search, the algorithm will use these parameters to determine which robot should be allocated this task and ideally, the robot with the least energy expenditure will be allocated the task.

An underwater environment also poses many constraints on navigation. We are unable to use the Global Positioning System under water or use any type of electromagnetic communication. Thus, we need to either remain on the surface or use sonar based communication underwater. Even on the surface, the GPS is not necessarily accurate and occasionally, the signal is lost due to the currents and waves in the water. We want our algorithm to account for these issues as well. One solution is to use an IMU for localization in lieu of GPS. However, an IMU is very noisy and is not useful for localization by any means. The IMU, however, is very useful for orienting and stabilizing the robot. In order to rectify this, we propose a hybrid solution using both GPS and IMU where the IMU is used for orientation and the GPS is used for location. When GPS signal is lost, we switch to IMU integration. The error should be low since we are not using the IMU for a long time but, rather for a couple of seconds. There are also many other issues associated with an aquatic environment. While navigating, a lot of error and deviation can occur due to (strong) currents and waves. We need to account for this error using control loops.

Another challenge faced in an aquatic environment is that the vehicles are prone to failure. It is very easy for a robot to get something stuck in it's motors resulting in motor failure, and ultimately, robot failure. Thus, our algorithm must also account for robot failures as well.

Now, that we discussed the challenges faced by the mobile vehicles, let us discuss the vehicles themselves and their hardware. The robot which we exploit is a BlueROV2 made by BlueRobotics [7] (Fig. 2(a)). The BlueROV is outfitted with 4 horizontal motors and 2 (4) motors for vertical movement in the regular (heavy) structure. The upper portion of the BlueROV is outfitted with four buoyancy foams. Each motor is connected to an ESC which is in turn controlled by the Pixhawk (Fig. 2(b)). A Pixhawk is an open hardware, autopilot flight controller. [8] The Pixhawk also incorporates sensors such as an IMU, gyroscope and a magnetometer. The IMUs on the Pixhawk consist of: an ST Micro L3GD20 3-axis 16-bit gyroscope, ST Micro LSM303D 3-axis 14-bit accelerometer / magnetometer and an Invensense MPU 6000 3-axis accelerometer/gyroscope. The Pixhawk in turn is connected to a Raspberry Pi. The Raspberry Pi runs a preconfigured Raspbian

(a) (b)

Fig. 2: (a) BlueROV2 [7], the vehicle (initially an ROV) which is used in this project. We program it to perform in the autonomous mode; (b) The Pixhawk controller [8] used in this project.

TABLE I: The hierarchy of the layers of autonomy.

Layers of autonomy	Functionality
5	Route planning
4	Localization
3	Stability and error control
2	Directional movement
1	Individual motor control

image based on Ardusub. [9] The Raspberry Pi is the main computational processor on the BlueROV. All computation and control is directed by the Pi. The Raspberry Pi allows control of the robot through a communications protocol known as MAVLink which allows communicating with unmanned vehicles. [10] MAVLink sends commands to the Pixhawk which in turn does some specific action based on the command received. Non-mechanical sensors, such as a conductivity sensor, interface directly with the Raspberry Pi. These sensors are controlled through a serial interface. The BlueROV is also outfitted with a ping echosonar sensor from BlueRobotics. The echosonar is connected directly to the Raspberry Pi and interfaces through a serial port. The BlueROV2 is also outfitted with an HD camera attached to a servo motor. However, for our case, we are not using this specific hardware. The hardware of interest to us is primarily the motors (for movement), the Pixhawk for control of the motors and for IMU data, and the Raspberry Pi for an interface. All of these components will allow us to make the ROV autonomous.

Layers of Autonomy: In order to make the robot autonomous, we define layers of control for the robot in order to allow the robot to be fully autonomous. Each layer can access the data and control loops from the layers below it. This design leads to a highly scalable, flexible autonomous system. Each layer can be built on top of another, abstracting the layers below. Consider the similarities between this design and the TCP/IP stack.

As seen in Table I, the layers of autonomy increase in complexity and functionality. Each layer depends on the abstractions of the previous layers for implementation.

The first layer of control, as seen in Table 1, we define as individual motor control. In essence, this layer involves

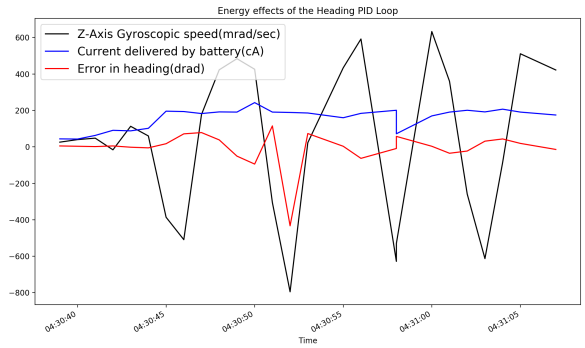


Fig. 3: The convergence of the error using the PID loop along with energy usage compared with angular speed.

the ability to control individual motors. This control is implemented by the the Pixhawk flight controller and is not a layer we can access. This layer allows us to control the motor speed.

The second layer is what we define to be directional movement. This layer of control is implemented for us by ArduSub. Each direction of movement is associated with a channel and we are able to directly control the speed of each channel using MavProxy (a proxy for MavLink). This layer allows us to incorporate the built in channel controls in MAVLink to control the directional movement of the robot.

The third layer is what we define to be the control layer. The control layer involves using control loops to make the robot account for errors and also to allow it to obtain a desired target with regards to movement. To explain further, we use control loops to allow the robot turn a certain number of degrees, or allow it to move a certain number of meters, and also allow the robot to move straight when encountering currents or drift. Specifically, we are using PID loops in order to guarantee this. This layer relies on PID loops using data from the motion related sensors on the robot (gyroscope, IMU, magnetometer and GPS). The robot should be able to keep a certain trajectory using these sensors and control loops. When we ran tests, we found that as the robot moved forward, it tended to drift to the right considerably more than was normal. In order to rectify this, we used a PID loop to correct the orientation of the robot and the drift based on the compass heading. Fig. 3 shows a graph showing the energy usage of the PID loop. When the error in heading increases, the loop adjusts the Z-axis angular speed of the ROV. The current delivered by the battery is fairly consistent throughout, implying that there is not a large energy expenditure associated with the PID control loop.

The next layer involves distance metrics. Let's call this the localization layer. Because we can make the robot move forward, turn and move vertically using the channel controls and the PID loops, we should be able to theoretically be able to do any three dimensional maneuver. The localization layer takes into account the IMU, in case there is no GPS, and the GPS in order to accurately control the movement of the

robot for a specified distance in a two dimensional plane. Specifically, the layer by default uses GPS for localization but in the case of no localization, the IMU is used instead. Localization is found through the IMU using integration principles. For vertical movement, the distance from the seafloor is measured using a ping echosonar module. For turns, a compass is used to ensure accuracy. On top of the localization layer, we have implemented a route planning layer which is part of our distributed sampling algorithm that we will discuss later in this report.

The next layer involves the robot making more higher level decisions of it's trajectory based on sensor data with regards to the water (i.e. conductivity, temperature, salinity, etc.). Let us call it the route planning layer. We would have to define some route planning algorithm built on top of the previous layer in order to achieve this. This algorithm is the distributed sampling algorithm we discuss in this paper. With this layer, we can safely state the robot is mostly autonomous since the robot can make it's own decisions about where to go and reach that destination without any outside input. In this layer, for our purposes, we are using sensors such as salinity, conductivity, temperature and pH to identify regions of interest and move to that area.

Our goal is to have a team of autonomous robots all sharing the tasks of sampling between themselves. As a result, there must be coordination between all of the robots to prevent collisions and to prevent redundant work being done. Thus, as part of the route planning algorithm, we must add a communication protocol for all of the robots to allocate tasks efficiently amongst themselves. This communication protocol is explained further in the paper.

Related Work: Previous work on adaptive sampling primarily uses one robot. A previous proposed method was using SLAM to conduct adaptive sampling. The method consisted of a static node and a mobile vehicle. SLAM was used to adjust the trajectory of the robot. The sampling involved two phases. The first phase involved sampling the area with equal spacing. Phase two used the values from phase one and adjusted the spacing based on the change in sampling. Thus, for an area with higher change, there were more data points. [3]. Our proposal involves robots on the surface so we use the Global Positioning System(GPS) instead. Another method using adaptive sampling involved taking the map of the environment and gridding the map. Then, allowable path points were found based on the distance to these points [11]. A previously proposed solution on cooperation between robots in terms of adaptive sampling was proposed by E. Fiorelli. The coordination between these robots was based on Virtual Body and Artificial Potential. Basically, there is an artificial potential between each pair of vehicles. The path planning of each robot is based on gradient climbing. [12]. A previous paper on distributed sampling uses multiple AUVs to follow a predefined path in parallel without communication between the robots. The process involves two phases. In the first phase, the AUVs traverse the area using a traditional lawnmower-style trajectory. Phase two involves the AUVs adaptively scanning the area by reducing the width of the lawnmower path based on sensor data. The second phase is run multiple times and the sampling done in the previous run is used to determine the sampling in the current run. The map of the region is then reconstructed using either Random Compressive Sensing (RCS) or Deterministic Compressive Sensing (DCS) techniques which reconstruct a signal given minimal samples [6]. The algorithm was optimized for energy efficiency and error minimization of multiple vehicles. [13]. A previous paper discussed the use of layered control for AUVs. The layered control used was known as subsumption architecture whereby each hierarchical layer could access data from layers below it and also influence those layers. For their solution, they gave each robot a predefined behavior. For mapping, each vehicle was given a feature map and a visitation map. The visitation map held all the points visited and the feature map held all locations where a feature of interest was detected [14]. Robot coordination is an interesting, hot topic. A recent paper on multi-path robot coordination. The paper involves n robots each starting at an origin and going to a goal specific to that robot. The goal of the algorithm was to minimize collisions. In order to do that, a rapidly-exploring random tree(RRT) was used. The algorithm found the distance from one point to a random point such that no collision occurred. From that random point, the process continued and the algorithm found a path to another random point such that no collisions occurred. In order to prevent conflicts and deadlocks of robots, synchronization of robots had to occur. Specifically, one robot waited until the other finished moving a shared path [15]. Yan et. al. [16] discuss communication and path planning between multiple robots. They discuss the two different kinds of communication, explicit and implicit communication. Implicit communication is interactions between robots, but the robots are not sharing data with each other. For explicit communication, three different approaches were discussed. The approaches were a market based approach, an auction-based approach and a trade-based approach. Each approach involves a buyer and a seller. Our approach is a mix of a trade-based approach and a market based approach. Evangeline et. al. discuss task allocation between multiple wireless sensor networks based on criteria of energy, tima quanta, queue length and sequence tables [17].

III. Proposed Solution

In this section, we propose our solution for distributed adaptive sampling using multiple vehicles.

For our solution, we propose to split our distributed sampling process into two phases. First, we section the area into equal regions so that each vehicle can traverse this region in a lawnmower fashion. As each vehicle is traversing the area, the vehicle records sensor data from the sensors outfitted on the robot. With the values obtained from the sensors, we can create a map of the region. We can define regions of interest (ROI) that we want to further explore. Once the traversal is finished, each robot has to broadcast the map of it's region to the other vehicles to obtain a global map of the environment.

Now, we would begin phase two of our process. For this phase, we need to allocate the regions of interest(ROI)

to different vehicles. Each vehicle will further search these regions leading to an overall more accurate map. After all vehicles have received the map and regions of interest, each vehicle can begin searching ROI. Each vehicle will find the closest region of interest to it. Next, each vehicle will broadcast it's decision to go to that region along with the distance it has to travel to go there. If another vehicle is going to that region at the same time and can reach that region in a shorter distance, the other vehicle will notify the other robots that it can go there faster. The other vehicle will then broadcast to other robots it's choice to go that region and the cycle will repeat. If no robots notify the others that they can go there faster within a given time interval, the vehicle will send a packet stating that it assumes that all robots have reached a consensus to allow it to go to that region of interest. After all ROI are explored, each robot will share it's map with the others to obtain a global map.

A. Phase 1: Initial Exploration

For phase 1, our first task is to split an area into equal subsections for each vehicle to traverse. For this part, we assume that the area is given, is rectangular and is bounded. Consider an area A. We want to split A into n regions such that for all n vehicles, each vehicle will optimally search it's given region in terms of time and energy. For the i^{th} vehicle, consider the cost of searching it's region to be the cost, c_i. We can prove that the optimal way to split the area into subsections is to split the area equally into n regions. Assume that the i^{th} vehicle searches region r_i and the j^{th} vehicle searches region r_j. The cost of exploring r_i is c_i and the cost of exploring r_j is c_j such that $c_j > c_i$. This is not an optimal solution because the j^{th} vehicle has a larger energy expenditure that the i^{th} vehicle. The total distance traveled cumulatively by all vehicles should remain near constant which implies that the total cumulative energy expenditure should remain near constant as well for this phase. Thus, an algorithm that creates unequal subsections of the area is non-optimal because on vehicle will have a higher energy expenditure than the other. Once the area is split into subsections, each vehicle traverses a lawnmower path of the subsection. During this traversal, each vehicle collects data through the on-board sensors. For this task, we need to employ autonomy to the vehicle.

Autonomy: As was discussed previously, we propose multiple layers of autonomy for the ROV with each hierarchical layer being more complex than previous layers and being able to use functionality and modify parameters within the previous layers. As seen in Table I, layers 1 and 2 are implemented within the Pixhawk and related firmware. What we are concerned with are the more complex layers. Later on in the paper, we will discuss layer 5 of autonomy.

Layer 3 is the error control layer. In this layer, we use PID loops to control the orientation of the vehicle. The vehicle has a tendency to drift while moving forward so the PID loop accounts for these errors. The PID loop prevents deviation in the heading to ensure accuracy of movement. We also propose a control loop to ensure we reach our destination. Every second, we correct our trajectory if needed to ensure we reach our destination.

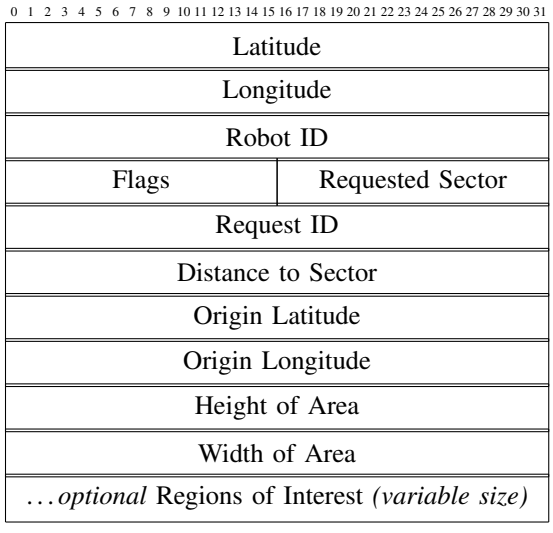
Map Reconstruction: Once each vehicle finishes traversal of it's allocated subsection, all of the maps of the subsections must be shared between robots in order to construct a global map of the environment so that each robot has this map for the next phase. As a solution, we use K-SVD in order to share the maps. Bajwa et. al. propose a distributed cloud K-SVD algorithm that uses an iterative process to generate a dictionary. We will use this algorithm in order to create a map when all of the robots share data with each other.

B. Phase 2: Task allocation

After phase 1 is completed, the ROI have to be allocated to each vehicle. For this phase, we want to minimize the distance each robot has to traverse in order to minimize the total time and energy expenditure. In order to do so, we necessitate a task allocation protocol for coordination between the vehicles. In order for the team of robots to distributively split the work, they have to decide how to split the work while also deciding which robot is allocated which task. We propose a real time, on-the-fly task allocation. In this sense, each robot knows only it's current task and possibly it's next task. In order to implement this task allocation, we implement coordination between the robots which necessitates the design of a communication protocol. In order for the robots to communicate with each other, we use the Raspberry Pi's on the robot and connect them to each other creating a communication link between robots.

Let us call the communication protocol Distributed Sampling Protocol (DSP). The protocol must allow for multiple types of communication. First, the protocol must allow for the robots to communicate the map and the regions of interest with each other. Second, the communication protocol must allow the identity of the sender of the packet to be known. Third, the protocol must prevent collisions between robots. Fourth, the protocol must prevent redundant work (e.g. searching the same region twice). Finally, the protocol must allow for task allocation based on energy in a robot, distance a robot has to travel and time that will be used.

ACK, NAK, REQ, MAP, CON, FIN



For the protocol, we define a header—with the size of 40 bytes—which will contain all of the necessary information. Based on the necessary features described above, we will need certain values in the header. The header will contain the following fields:

Latitude, Longitude, Robot ID, Flags, Request ID, Requested Sector, Distance to Sector, Origin Latitude, Origin Longitude, Height, Width

There will be an optional payload for sharing the regions of interest. We will not need to share ROI much so this is not part of the header.

The fields of latitude and longitude are necessary fields. When a robot broadcasts a packet with it's location, there must be fields in the header for location. A robot must broadcast it's location regularly in order to prevent collisions between robots and to prevent redundant work from being done. The packet that a robot sends with it's location must allow identification of the robot. Otherwise, it would be impossible to distinguish the locations of any robot. Flags allow us to specify the type of data we are sending an enable acknowledgments and refusals. Next, we have the fields related to requests. For requests, a robot needs to request a sector to search and needs to pass the distance to that sector. The distance is necessary because if a robot is currently planning on going to that sector and will be able to go in a shorter distance, then priority is given to that robot. Because we will be using multiple robots, we risk the chance of multiple robots sending requests at the same time. Thus, to distinguish between requests and any replies to the requests, we use a request ID. Next, we have headers related to the map. For simplicity, we define our area that we search to always be rectangular in nature. We define our map with an origin which is defined by it's latitude and longitude, and the width and the height of the area. These four parameters sufficiently give the area and the boundaries of the area.

Next, we have to allow for sending the regions of interest when a robot broadcasts the map to the other robots. This data can be sent as an additional payload attached to the header. The size of the payload will be variable depending on the number of regions we have.

A flag notifies the receiver of the kind of data in the packet. A flag is a signaling mechanism. Our first flag is an ACK. This flag is used by robots to send an acknowledgement to other robots of some data that might be received or some decision that was decided upon.

The second flag is the NAK. The NAK is used for robots to communicate that they received a packet, but do not agree with the decision or information related to the packet. Primarily, the NAK will be used by a robot to contest a decision during requests. For example, say robot 1 broadcasts it's decision to go to sector 2, which is 10 meters away, to search that region further. When a robot wants to request a sector, it sets the flag to REQ. The robot also has to use a unique request ID so that the request can be tracked by all robots. Robot 2 receives this packet and at the same time, is also considering going to sector 2 which is 5 meters away. Robot 2 realizes it can reach sector 2 in a shorter distance which allows it to use less energy and get there quicker. So

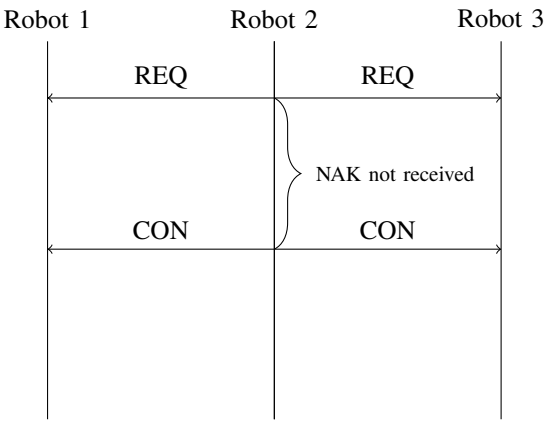


Fig. 4: The sequence diagram for when consensus is reached.

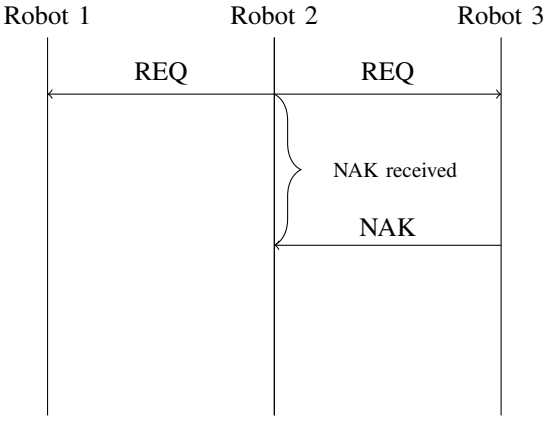


Fig. 5: The sequence diagram for when consensus is not reached.

robot 2 broadcasts a NAK with the respective request ID. The requesting robot then finds another region of interest to search. The robot that broadcasted the NAK has to broadcast it's own request. The NAK just prevents other robots from searching the area.

The CON flag is also used in the context of requesting sectors. If a NAK is not received within a given time frame, the robot that initially broadcasted the request broadcasts a CON with the respective request ID to all robots in the vicinity. A CON flag denotes the assumption that no other robot can reach that sector more quickly and in a shorter distance. Thus, it is assumed that all robots agree on the requesting robot to go to the desired sector and that a consensus is reached in that regard. Once a robot has no other regions that it can search further, it broadcasts a FIN letting all other robots know that it is done sampling. Once the battery level of a robot goes below a certain threshold, the robot will also send a FIN after finishing up it's current task and stop sampling.

To send the map to all robots, the MAP flag is set in the packet that was broadcast and the map is broadcast using the origin longitude, origin latitude, height and width and regions of interest fields. The first four parameters map out the area

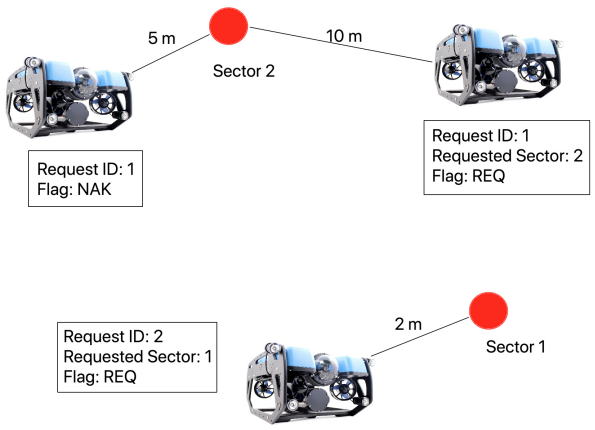


Fig. 6: An example situation of the request mechanism.

including the boundaries of the area. The regions of interest is an optional payload that defines the regions we want to search further. If this payload is not supplied when the map is shared between robots, then the robots assume there are no regions of interest to search.

Since we want to prevent robots from colliding, we want every robot to know the location of all other robots. In order to do this, each robot must broadcast it's location to all other packets. Each robot must do this regularly for all robots to have up to date information. However, we do not want to broadcast a location packet so often that we cannot transmit other data packets. Thus, every four seconds, each robot will broadcast a location packet with it's current latitude and longitude to all robots in the vicinity.

This protocol is highly scalable. In the case of many robots, such that each of them cannot communicate with all others, we split that area into smaller subsections with robots assigned to a subsection. In the subsection, we can use our algorithm and communication protocol for distributed sampling.

C. Distributed Sampling

We have so far defined the communication protocol which allows for our distributed sampling algorithm to run. Now, let us define our algorithm.

The first step in the algorithm involves creating a map of the environment. The map of the environment involves bounding the area we are going to search by the origin point and the width and height of the area. However, since we are conducting sampling, we also need a map of the values at points in this area. In order to obtain this map of values, we will have each robot traverse a sectioned area in a lawnmower path. As the robot traverses the path, it will record the measurements at all points it traverses along with the latitude and longitude of those points.

Once each robot has the map of the environment, each robot needs to define regions of interest to search further. These regions of interest could be regions with relatively high values or relatively low values. We standardize our definition of relatively low and high values to be any value that is within 10% of the maximum and minimum values recorded in the area. Regions of interest could also be regions with a high gradient. In order to define regions with a high gradient, we will have our initial vehicles that does the lawnmower path find the maximum and minimum values in the area. Since the vehicle has knowledge of the coordinates and the values at the coordinates, the vehicle can calculate the gradients within a certain distance. If the gradient is 40% or greater across a distance that is 5% of the area, then we mark that as a region of interest.

Once the regions of interest are defined, each robot will broadcast the local map and the maps, including the ROI, will be joined using Cloud K-SVD.

Once all robots have received the global map, we are ready to begin distributed sampling. For distributed sampling, we define a greedy algorithm approach. The goal of our algorithm is to minimize our energy usage while maximizing our data points. In order to minimize our energy use, we minimize the distance each robot travels. We also only allow a robot to move if it's battery level is above a threshold, 20%.

For distributed sampling, each robot will allocate to itself one region to search after all robots have reached a consensus on which explores a certain region as defined in the communication protocol. Once this region is allocated to a robot, the robot will go to the region and search it further. When the robot finishes searching this region, it moves on to another region.

The robot determines which region it wants to go to next by factoring in the distance it has to travel to that region. Each robot knows it's own coordinate position and also knows the coordinates of all regions of interest. The robot can easily calculate the distance from itself to all region of interest and quickly find the region closest to it. Because each robot chooses the region closest to it, we minimize the distance traveled and thus minimize the energy used.

When a robot broadcasts a REQ to go to a certain region, another robot can contest that and send a NAK if it can go to that region more in a shorter distance and is currently idle. In this case, the algorithm chooses the robot that can move a shorter distance to go to that region of interest. Thus, the algorithm also ensures energy efficiency of the entire system, not just individual robots. However, if a robot can go to that region in a shorter distance but is currently preoccupied with another task, the algorithm chooses the robot that broadcasted the REQ to go to that position. As a result, we ensure that no robot remains idle for a long period of time which helps ensure energy efficiency as well. The algorithm ensures that the sampling finishes as quickly as possible, reducing the energy consumed by minimizing the time used for sampling.

1) Proof: This algorithm is optimal for energy efficiency because each robot moves the minimum distance it can travel by moving to the closest region to it's position.

Consider each task to have cost C_i denoting the energy and time expenditure. The cost is a function of the current energy remaining for the vehicle and the distance traveled. Consider

a region R_k. For all vehicles 1 to n, let the cost of exploring this region be $C_1, C_2, ...C_i, ...C_n$. We want to find a vehicle i that minimizes the cost such that $C_i < C_j \forall j \neq i$. By finding a C_i that is minimal, we can optimally explore region R_k. For all regions, we can optimize the cost associated with exploring a region. By minimizing the cost of exploring one region, we minimize the sum of the costs of exploring all regions. Since our algorithm is time based and energy minimizing we thus minimize our energy and time expenditure.

IV. Performance Results

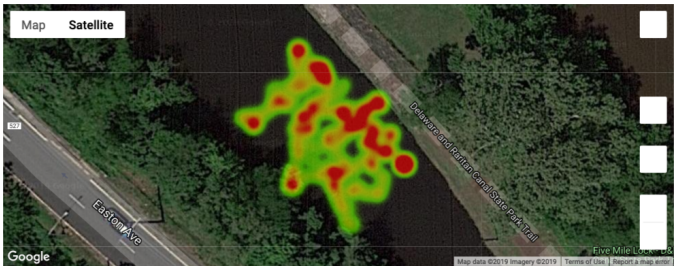


Fig. 7: A temperature map of the environment mapped with a lawnmower path.

Fig. 8: A temperature map of the environment mapped with a square path.

Consider figures 7 and 8. The data corresponding to these figures was found within an twenty minutes of each other. Thus, the data obtained should theoretically be very similar. From the maps, it is clear that the data obtained from the lawnmower path is far more accurate and comprehensive compared to the square path. The square path only obtains data for the perimeter and the data is concentrated. The lawnmower path, on the other hand, gives a more comprehensive, accurate map of the environment.

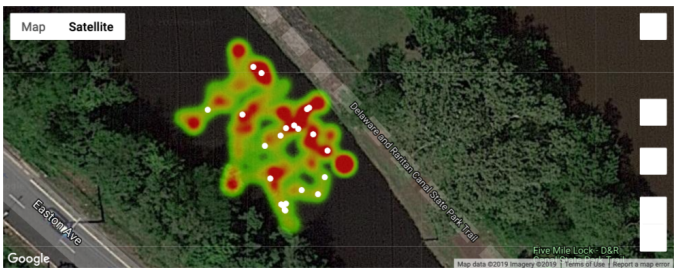


Fig. 9: A map with the regions of interest shown in white.

Algorithm 1: DistributedSampling(area, vehicles)

Result: Map of environment
n ← len(vehicles);
V[] ← vehicles;
data ← [];
R[] ← section area into N regions;
for $i = 1...n$ **do**
  Let $V[i]$ traverse $R[i]$ in lawnmower fashion;
  append collected data to data;
end
ROI ← [];
for *data* **do**
  S ← STDDEV(data);
  Mean ← MEAN(data);
  Max ← FINDMAX(data);
  Min ← FINDMIN(data);
  Diff ← Max − Min;
  for *Region in data* **do**
    if $abs(Region - Max) < 0.1 * Diff$ *OR* $abs(Region - Min) < 0.1 * Diff$ **then**
      ROI.append(Region);
    end
    gradient ← CalcGrad(Region);
    if $gradient >= 40\%$ *and* $dist >= 0.5 * size(area)$ **then**
      ROI.append(Region);
    end
  end
end
for *each robot* **do**
  packet ← ROI, data;
  Broadcast packet;
  for *packets received* **do**
    ROI.append(packet.ROI) using K-SVD;
    data.append(packet.data) using K-SVD;
  end
end
while *len(ROI) > 0* **do**
  distance, region ← find closest region from ROI;
  packet ← region,distance;
  Broadcast packet;
  if *packets received* **then**
    if *region == packet.region and distance < packet.distance* **then**
      broadcast NAK;
    end
  end
  if *NAK received* **then**
    continue;
  else
    explore region;
  end
end
for *each robot* **do**
  packet ← data;
  Broadcast packet;
  for *packets received* **do**
    data.append(packet.data) using K-SVD;
  end
end
return data;

Figure 9 shows the regions of interest allocated after phase I traversal of the area. These regions were found using the criteria discussed above.

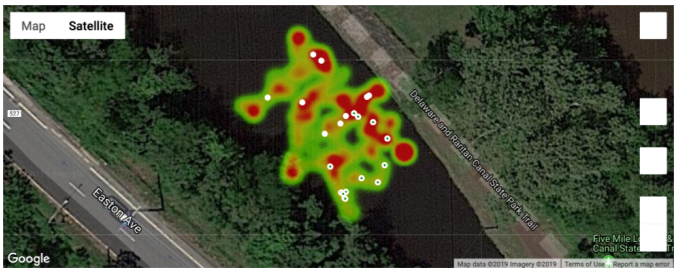


Fig. 10: A map with the allocated regions of interest.

Figure 10 shows the regions that were allocated to each vehicle. The white regions were allocated to ROV 1 while the green dot regions were allocated to ROV 2. As is evident, the regions that ROV 1 searches are very close to each other and the same is true for ROV2.

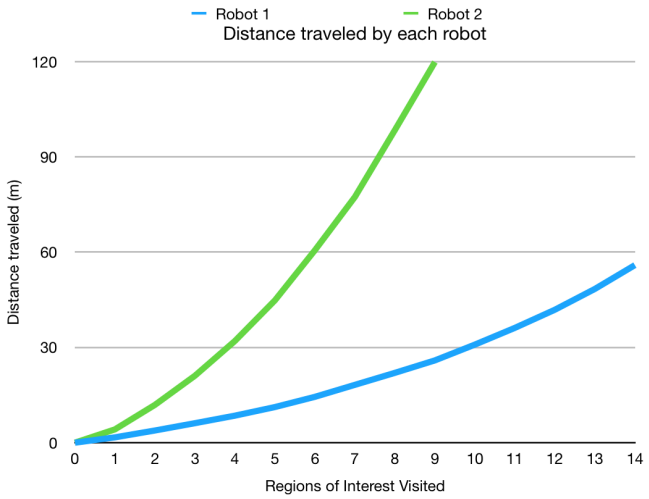


Fig. 11: Distance traveled.

Figure 11 shows the distance traveled by each robot overtime as they explore each region of interest. It is clear that the distance traveled from one region to another increases as the number of regions explored increases. In the beginning, the robots move less distance. But as time goes on, the regions become much more spread out. The regions explored later on are by nature farther away.

V. Conclusion and Future Work

The algorithm described in this paper allows for dynamic, distributed sampling with an aim of reducing energy usage while also attempting to get the most useful data from an environment. The algorithm is scalable and useful for many purposes. In terms of future work, there are many direction in which this project can go. First and foremost, we can add a Markov Decision Process to the task allocation to more intelligently allocate tasks. In phase I, during the exploration, instead of allocating equal regions to each vehicle, we can give each vehicle a certain behavior. Based on this behavior, we can explore the area at large. Then, for phase 2, we can establish relationships between robots based on behaviors. That is, we can have some robots be curious and search unexplored areas, while other robots are safer and only search perimeters. Then, during phase II, we can let the robots allocate tasks based on what we defined previously and also based on their behaviors. To implement behaviors, we can give each robot a risk coefficient which defines the level of risk they are willing to take.

We can also add sound based communication and USBL. These features will enable the vehicles to accurately monitor their locations and the locations of other vehicles in the water and communicate between each other. We can effectively monitor underwater structures and conditions with these features.

References

[1] I. F. Akyildiz, D. Pompili, and T. Melodia, "Underwater acoustic sensor networks: Research challenges," *Ad Hoc Networks*, vol. 3, no. 3, pp. 257–279, Mar. 2005.

[2] M. Rahmati and D. Pompili, "Ssfb: Signal-space-frequency beamforming for underwater acoustic video transmission," in *Proceedings of International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2017, pp. 180–188.

[3] M. Rahmati, S. Karten, and D. Pompili, "Slam-based underwater adaptive sampling using autonomous vehicles," in *Proceedings of MTS/IEEE OCEANS*. IEEE, 2018, pp. 1–7.

[4] M. Rahmati and D. Pompili, "Probabilistic spatially-divided multiple access in underwater acoustic sparse networks," *IEEE Transactions on Mobile Computing, doi: 10.1109/TMC.2018.2877683*, pp. 1–14, 2018.

[5] M. Rahmati, A. Gurney, and D. Pompili, "Adaptive underwater video transmission via software-defined mimo acoustic modems," in *Proceedings of OCEANS Conference*. MTS/IEEE, 2018, pp. 1–6.

[6] P. Pandey and D. Pompili, "Distributed computing framework for underwater acoustic sensor networks," in *2013 IEEE International Conference on Distributed Computing in Sensor Systems*. IEEE, 2013, pp. 318–320.

[7] "Bluerobotics website," https://www.bluerobotics.com/.

[8] "Pixhawk website," http://pixhawk.org//.

[9] "Ardusub website," https://www.ardusub.com/.

[10] "Mavlink documentation," https://mavlink.io/en/.

[11] Y. Wang, C. S. Wu, and Y. Q. Shi, "The application of maximum differential algorithm in adaptive ocean sampling," in *Materials Science, Computer and Information Technology*, ser. Advanced Materials Research, vol. 989. Trans Tech Publications Ltd, 9 2014, pp. 1456–1459.

[12] E. Fiorelli, N. Ehrich Leonard, P. Bhatta, D. A. Paley, R. Bachmayer, and D. Fratantoni, "Multi-auv control and adaptive sampling in monterey bay," *Oceanic Engineering, IEEE Journal of*, vol. 31, pp. 935 – 948, 11 2006.

[13] B. Chen, P. Pandey, and D. Pompili, "A distributed adaptive sampling soluting using autonomous underwater vehicles," in *Proceedings of the Seventh ACM International Conference on Underwater Networks and Systems*. ACM, 2012, p. 29.

[14] A. A. Bennett and J. J. Leonard, "Behavior-based approach to adaptive feature detection and following with autonomous underwater vehicles," *Oceanic Engineering, IEEE Journal of*, vol. 25(2), pp. 213 – 226, 05 2000.

[15] G. Best, M. Forrai, R. R. Mettu, and R. Fitch, "Planning-aware communication for decentralised multi-robot coordination," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 1050–1057.

[16] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013. [Online]. Available: https://doi.org/10.5772/57313

[17] C. C. Evangeline, C. Kushala, and B. K. R. Alluri, "Efficient task allocation strategies for wsns," in *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, Dec 2018, pp. 446–450.