Coordinated Sensor-based Area Coverage and Cooperative Localization of a heterogeneous fleet of Autonomous Surface Vessels (ASVs)

Parisa Mojiri Forooshani

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

GRADUATE PROGRAMME IN MASTER OF COMPUTER ENGINEERING
YORK UNIVERSITY
TORONTO, ONTARIO

April 2015

Abstract

Sensor coverage with fleets of robots is a complex task requiring solutions to localization, communication, navigation and basic sensor coverage. Sensor coverage of large areas is a problem that occurs in a variety of different environments from terrestrial to aerial to aquatic. In this thesis we consider the aquatic version of the problem. Given a known aquatic environment and collection of aquatic surface vehicles with known kinematic and dynamic constraints, how can a fleet of vehicles be deployed to provide sensor coverage of the surface of the body of water? Rather than considering this problem in general, in this work we consider the problem given a specific fleet consisting of one very well equipped robot aided by a number of smaller, less well equipped devices that must operate in close proximity to the main robot. A boustrophedon decomposition algorithm is developed that incorporates the motion, sensing and communication constraints imposed by the autonomous fleet. Solving the coverage problem leads to a localization/communication problem. A critical problem for a group of autonomous vehicles is ensuring that the collection operates within a common reference frame. Here we consider the problem of localizing a heterogenous collection of aquatic surface vessels within a global reference frame. We assume that one vessel – the mother robot – has access to global position data of high accuracy, while the other vessels – the child robots – utilize limited onboard sensors and sophisticated sensors on board the mother robot to localize themselves. This thesis provides details of the design of the elements of the heterogeneous fleet including the sensors and sensing algorithms along with the communication strategy used to localize all elements of the fleet within a global reference frame. Details of the robot platforms to be used in implementing a solution are also described. Simulation of the approach is used to demonstrate the effectiveness of the algorithm, and the algorithm and its components are evaluated using a fleet of ASVs.

To my husband, Saeid

and to my father and mother

Acknowledgements

It is now time to play backward and remember all those to whom you believe the minimum you owe is a thank you message. This thesis has been a long time in the making, and it would not have been possible without the help of various people along the way. First of all, I would like to thank my supervisor, Prof. Michael Jenkin, for his support and patience with me. He has calmly guided me through the different stages of this thesis, despite various delays and setbacks. His reading, rereading and comments have helped shape this thesis into what it is. And, I would like to thank Hui, Robert in the VGRLab for all the help they have given me for several field trials in different weather conditions.

Also, I would like to thank my dearest parents for their support and patience with me. Of course, this thesis would not have been finished without the support, encouragement, help, and love of my husband, Saeid. You are the best! And, before and after everything, and in the midst of everything, All praise goes to Allah, the Lord of the worlds.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis deals with the task of providing sensor coverage of a large body of water. Sensor coverage of a large body of water is a task that has wide applicability, but to ground the task in a specific example consider the problem of responding to a major oil spill accident such as the Deepwater Horizon disaster [12]. This incident caused serious damage to the marine and wildlife habitats and the fishing and tourism industry of the Gulf of Mexico. The disaster began on April 20, 2010 and is considered to be the largest accidental marine oil spill ever reported[1]. Following the explosion of the oil rig, oil gushed from the broken drill head for 87 days contaminating much of the Gulf. It is estimated that the total surface area impacted by the spill exceeded 180,000 km^2. The scope of the disaster required the deployment of a wide range of different technologies to both estimate the scope of the disaster and to begin the process of cleaning up the environment and repairing the damaged environment. Dealing with an environmental disaster of this magnitude introduces a range of complex tasks related to distributed sensing. For example, it becomes critical to be able to answer questions such as where, within this environment, is the contamination and how serious is it at different locations. From a technical point of view answering this type of question involves providing sensor coverage of the impacted area. Although satellite and airborne sensing can provide some of this information, determining the depth and concentration of oil on and near the surface requires sensors to actually be deployed over an extremely large surface area. Of course, distributed sensor coverage finds wide applications in other areas too, including automated vacuum-cleaning, carpet-cleaning, lawn

[1] "BP leak the world's worst accidental oil spill". The Daily Telegraph (London). 3 August 2010. Retrieved 15 August 2010.

mowing; chemical or radioactive spill detection and cleanup, water sampling; and humanitarian de-mining.

Many sensor coverage applications require complete coverage. The goal of complete coverage is to provide a path algorithm to guide a robot or a collection of robots to pass over the *entire* accessible area of the target environment using their sensors. Complete coverage guarantees that no accessible area is left unmonitored by the robot's sensors. Some applications may only require partial coverage. Others may require the robot(s) to actually move such that they physically pass over each all points in the environment [103]. In each of these scenarios the basic task is the same: How do we plan motion for a group of robots so that their sensors provide coverage of some previously mapped environment?

The performance of coverage algorithms for a given environment is limited by the robot sensor footprint and its speed through its environment. Given the time sensitive nature of providing area coverage for tasks such as oil spills and the large potential extent of the disaster, a single autonomous device is unlikely to be able to cover such an environment in a timely manner. One potential solution to this problem is to deploy a collection of autonomous devices rather than a single robot. Introducing multiple robots provides advantages in terms of efficiency and robustness but increases the complexity of the coverage algorithm and also requires that the various elements to operate within a common frame of reference. There is also the critical question of the nature of the group of robots being deployed to the task. The group of robots could be homogeneous. That is, each member of the group could have the same capabilities and characteristics. Another approach would be to imagine a heterogeneous group of robots where each robot may have different capabilities. In such a case it becomes important to deploy the group of robots so that each robot can exploit its own characteristics. For example, a heterogeneous team may consist of some robots equipped with high accuracy localization sensors, others with less effective sensors, and perhaps others with no form of position perception at all. How can all the elements in this group operate within a common frame of reference? Surveying the environment with such a team requires the team to share information allowing the more capable robots to assist those with lower quality instrumentation. The main technical challenges associated with the development of a heterogenous multi-robot coverage system are:

- The development of a coverage algorithm suitable for deployment on a heterogeneous fleet of autonomous robots that can deal with the non-holonomic motion properties associated with many autonomous surface craft.

- The development of a cooperative localization system such that sensor information obtained from multiple robots can be easily integrated into a common representation.
- The development of an appropriate communication infrastructure so that state and other information can be easily communicated among the members of the fleet.

Developing such algorithms requires the development of an appropriate infrastructure upon which to test them.

1.1 Research objective

This work examines specific theoretical and practical issues associated with deploying a heterogenous fleet of autonomous surface vehicles (ASVs) to provide complete sensor coverage of a known environment. Rather than addressing the problem in general, this work considers the problem of providing area coverage using a heterogeneous robot fleet consisting of one extremely sophisticated robot that is equipped with extensive sensor capabilities (like an accurate localization system) aided by a fleet of less capable robots. One particular aspect of this differentiation is that the sophisticated robot is assumed to be equipped with an extremely accurate global localization system, while the less capable robots are not.

The focus of this thesis is on resolving the tasks required to enable a heterogeneous fleet of ASVs to operate in a coordinated fashion in order to provide coverage of a known environment. The world model is assumed to be known and to be well modelled as a plane with an exterior polygon boundary with polygonal islands contained within it. The individual robots are assumed to have known sensor and communication capabilities. A restricted communication model is considered in which communication failures are expected and must be dealt with, and, the development of a team-based coverage and localization algorithms of heterogeneous systems are of particular interest. The specific type of heterogeneous robot system explored in this work consists of two types of robots: a *mother robot* named *Eddy* (see Figure 1.1(a), 1.1(b) and Appendix A) with capable sensors including differential GPS, and a group of multiple smaller but less expensive *child robots* named *Minnows* (see Figure 1.1(c), 1.1(d), Appendix B, and [39]) with less effective sensors. While the *mother robot* is able to robustly localize itself within the environment, the *child robots* must rely on the mother robot for localization. The small mobile robots are less expensive and thus can be employed in larger numbers. Eddy can be considered as a holonomic

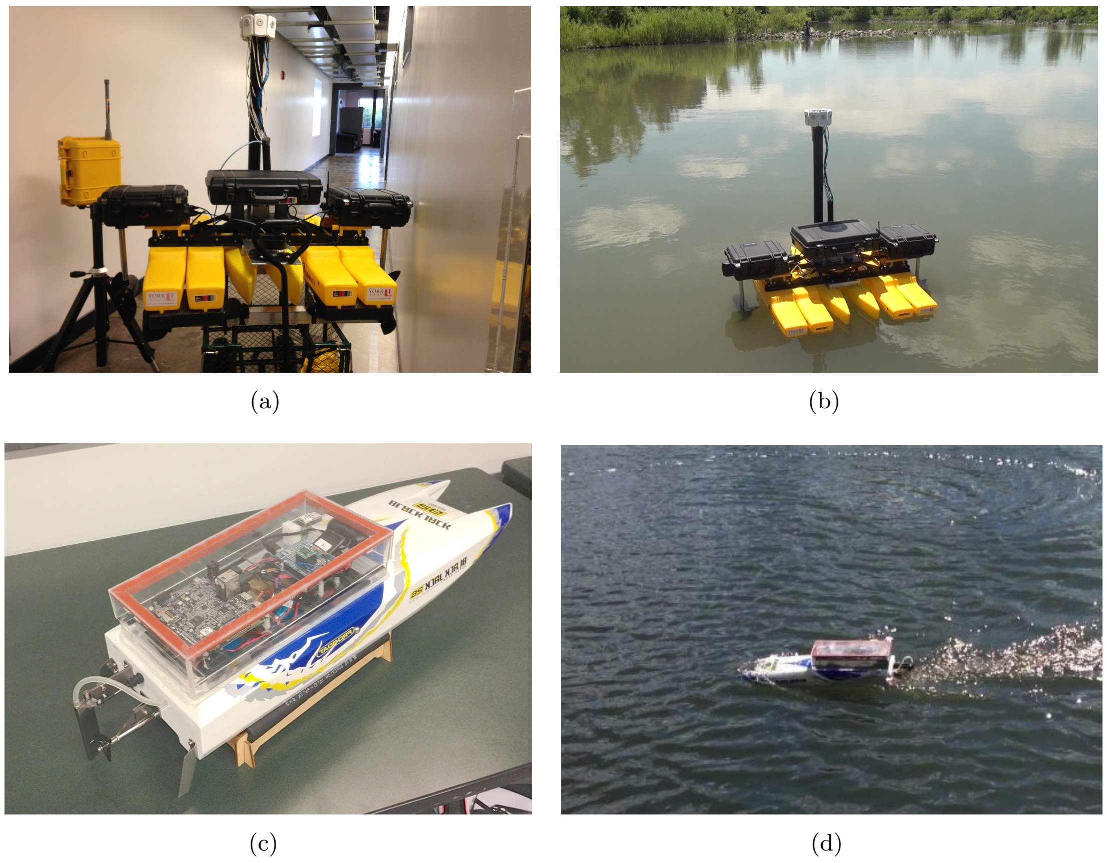


FIGURE 1.1: The robots used in this work. (a) Eddy bench view. (b) Eddy operating in the water. (c) Minnow side view. (d) Minnow operating in the water.

robot, that is it is capable of motion in any direction. The Minnow platform, on the other hand, is a non-holonomic platform. Its locomotive strategy limits its motion perpendicular to its current orientation. Addressing the area coverage problem related for this heterogeneous fleet involves developing solutions to a range of problems related to the actual design and construction of the fleet, communication strategies to deal with signal loss, localization, and the actual coverage algorithm itself. This thesis seeks to specifically answer the following questions:

- Given dynamic constraints of the various robots (mother and child robots), how can the robots be controlled to provide coverage of a known body of water?
- Given that the mother robot can trivially solve the global localization problem, how can the localization problem be solved for the entire fleet?
- Given a team-based coverage and localization algorithms of heterogenous fleet of robots, how can we deal with intra-fleet communication so that state and information can be easily communicated among the members of the fleet.

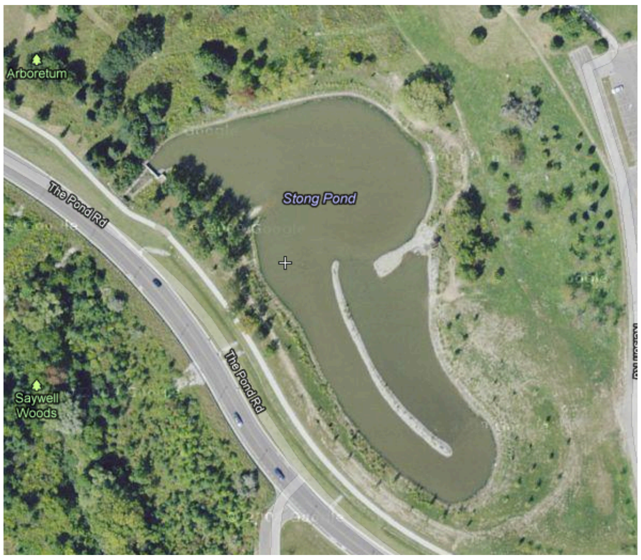


FIGURE 1.2: The aerial picture of Stong Pond located at York University reprinted from Google.

In this work the above questions are answered in both the theoretical sense through the development of algorithms that address the task, as well as via experimental validation using robots designed and built as part of this thesis. Although the algorithms developed within this work are designed to operate in generic environments, much of the actual field work described in this thesis takes place on Stong Pond at York University (see Figure 1.2). Stong Pond is approximately $114\,m \times 165\,m$ in size, and is located on the south west side of York University campus in Toronto.

1.2 Organization of the thesis

Chapter 2 provides a review of previous work related to this thesis. In particular, it reviews relevant work in sensor coverage algorithms with particular attention to coverage algorithms using collections of autonomous agents. Chapter 3 deals with the details of the various algorithms that have been developed to address the problems given above with a collection of robots. In Chapter 3 the problem is considered in a pure (simple) form first. This approach is then adapted to the problem of non-holonomic robots with real communication complexities. The experimental validation provided in Chapter 4 through both simulation and validation with the robots operating in the real world. A summary and future work are presented in Chapter 5. Finally, Appendices A and B provide details of the robot system deployed in this work.

Chapter 2

Previous work

2.1 Introduction

This chapter starts with a brief review on the history of main concepts of coverage path planning. Then, different types of coverage algorithms, both single robot and multiple robot algorithms, are reviewed. In Section 2.3, the problem of two-dimensional (2D) pose estimation for a collective robots is described and a number of approaches reviewed. The Section 2.4 deals with the problem of distributing ROS (Robot Operating System) control over a collection of robots subject to failures in communication between the robots. The chapter concludes with some remarks on the need for further research for coverage algorithms based on the literature review presented.

2.2 Coverage path planning algorithms

The problem of robotic coverage and exploration has developed quickly, from early heuristic-based single robot techniques (e.g., [37, 58]) to multiple heterogeneous robots working in a coordinated fashion (e.g., [70, 87]) in only a few years. The problem has been extensively investigated in both the single-robot domain [46, 56] as well as for multi-robot systems [66, 95, 97, 107]. Prior to presenting a detailed description of the various algorithms, it is worth first defining formally the terms *coverage* and *exploration*: in the exploration problem, there is an unknown environment and a robot or a team of robots cooperate to build a map of the environment [38]; in the coverage problem, the map of the environment may be known or unknown, and the goal of the algorithm is to jointly observe/sweep the whole area with their sensors or physical actuators.

Complete coverage path planning involves determining a path such that every point in a given workspace is covered at least once by the robot or its sensors [41]. In some applications, such as vacuum cleaning, the robots are expected to physically move over all points [92]. In other applications, such as patrolling and search and rescue operations, it is enough to move the robot such that the sensors on the robot cover all the points of the environment [28, 103].

Although a range of different formal problem definitions exist for the coverage problem, typically, a bounded work area, possibly containing obstacles, is given to a robot assuming the robot has a detector or tool with an associated sensor range or coverage. This tool corresponds to the robot's relevant sensor or actuator range which is affixed to the robot, which then moves to visit all the points within the work area. The robot may be subject to specific motion constraints (both holonomic and non-holonomic). Since the work area is typically much larger than the sensor coverage size, the robot's task consists of determining and then moving along a path that will take the robot-mounted detector over the entire terrain subject to the limitations of the robot's motions.

The problem of coverage path planning of a known environment has a long history and is strongly related to the well known art gallery problem [85]. In two dimensions, the basic version of the art gallery problem is defined as follows. We are given a known simple polygon P, and we wish to know if a set of points $S \subset P$, has the property that for every point $p \in P$, there exists some $q \in S$ such that the line segment between p and q does not leave the polygon [85]. In essence, the set of points S 'guards' all of P. This decision problem is known to be NP-hard [85]. In the computational geometry version of the problem the layout of the art gallery is represented by a simple polygon and each guard is represented by a point in the polygon as indicated in Figure 2.1. Here the problem is to find the minimum number of guards needed to station in a polygonal gallery, and hence each point in the gallery is visible to at least one guard. The Art Gallery problem does not require a determination of the path that the guard must follow nor does it consider a limited sensor range for the guard's sensor.

Other closely related problems include the safari problem, the zoo-keeper problem, and the watchman route problems [52]. For the safari and zookeeper problems, we are given a simple polygon P and disjoint convex polygons $P_1, ..., P_k$ inside P. In the zookeeper problem the shortest route inside P is sought that visits each of the P_i's but never enters any of them. This problem

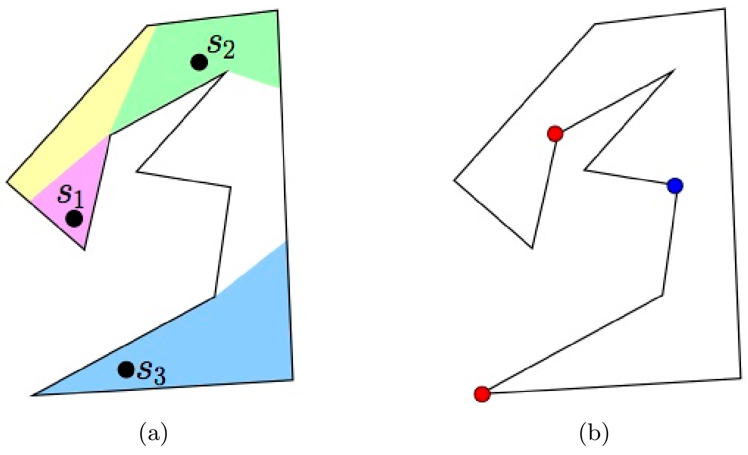


FIGURE 2.1: (a) Three guards in a polygon P. The yellow region denotes the points that are visible from both s_1 and s_2. The purple region is the area visible only from s_1, and the green region is the area visible only from s_2. The blue region is visible from s_3. (b) A minimum guard placement required for this polygon.

can be solved in $O(n^2log(n))$ time where n is the total number of vertices specifying the polygons [52]. In the safari problem a shortest route inside P is sought that visits each of the P_i's, and is allowed to enter them [52]. The watchman route problem involves finding the shortest route, typically within a simple 2D polygon, such that a watchman travelling the route will view the entire polygon. Interestingly, this problem is known to have an $O(n^3log(n))$ solution where n is the total number of vertices specifying the polygons [52].

These graph-theoretic problems are closely related to the coverage problem, where limited sensing is introduced and the nature of the environment is relaxed. The coverage problem is defined as follows [25]. Given a known environment E, and a robot with a sensor that provides some coverage S(x) when the robot is in state x, the goal is to find a path (or route) such that following the route x(t) covers all of space E. That is $E \subseteq \int S(x(t))dt$. This basic definition can be trivially expanded to N robots, each with their own sensor S_i. In this case we seek:

$$E \subseteq \bigcup_{i \in N} \int S_i(x_i(t))dt \tag{2.1}$$

that is, we seek path sequences for the N robots such that over the time interval their sensors cover all of E.

2.2.1 Single robot sensor coverage

Given the wide range of applications for sensor coverage algorithms, there has been much work in the area, especially for a single agent (robot). A survey of coverage algorithms prior to 2001 is provided by Choset [41]. The review here follows Choset's but augments it with examples from more recent results from the literature. Choset distinguishes between *offline* algorithms, in which a map of the work-area is given to the robots in advance, and *online* algorithms, in which no map is given and utilize real-time sensor measurements to observe the target space that these algorithms called sensor-based coverage algorithms. Furthermore, Choset makes a distinction between an *approximate cellular decomposition* and an *exact decomposition*. In an *approximate cellular decomposition*, the free space is approximated, for example by a grid of equally-shaped cells, while in *exact decomposition* the free space is decomposed to a set of regions, whose union fills the entire area precisely.

Approximate cellular decomposition

Under appropriate models of the robot's sensor(s) and the environment, efficient solutions to the coverage algorithm exist. Gabriely and Rimon proposed the *Spanning-Tree Coverage* algorithm in [56] using a single mobile robot. In their work, a basic method for dividing the terrain is presented, and a polynomial time spanning tree coverage algorithm (STC) that is an *approximate cellular decomposition* is described for both complete offline and online coverage of the terrain using a single robot. Later, they proposed two different algorithms for constructing an on-line tree [57]. The two algorithms differ in the incremental rule employed to construct the spanning tree, but the motivation comes from the creation of a spanning tree with a specific scanning direction.

A grid-based coverage algorithm named spatial cell diffusion (SCD) is presented in [101] that can be applied to unknown and known work spaces. This algorithm encodes the entire area as a group of Gray codes [93] for grid cells with sizes less than or equal to the footprint of the robot. The algorithm extends its sweep area by diffusing occupied cells towards the environment's boundary through a continuous movement.

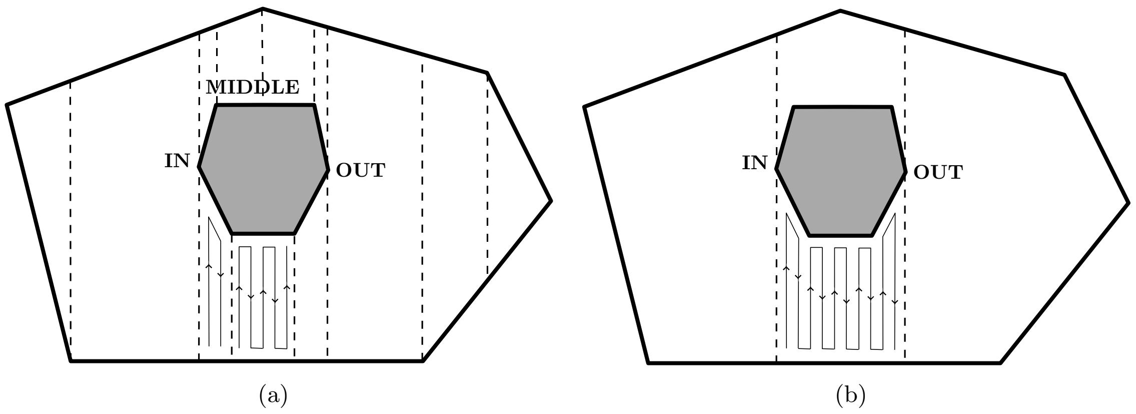


FIGURE 2.2: (a) Trapezoidal decomposition of an environment. (b) The Boustrophedon decomposition of the same environment.

Exact cellular decomposition

An *exact cellular decomposition* representation decomposes free configuration space C_{free} into a collection K of non-overlapping cells such that the union of all the cells exactly equals C_{free}, i.e., $C_{free} = \bigcup_{k \in K} k$. Each cell is represented as a node in a graph, called the adjacency graph, where adjacent cells in the configuration space are linked to each other by connecting their corresponding nodes [42].

One of the simplest *exact cellular decomposition* techniques is trapezoidal decomposition [75]. In this approach a vertical line, named a *slice*, is passed left to right through a known environment which is occupied with polygonal obstacles. When a slice intersects a vertex (this is described as an *event*) *cells* are constructed via a sequence of open and close operations. There are three types of events: IN, OUT, and MIDDLE. At an IN event the current cell is closed and two new cells are opened. An OUT event is the reverse: two cells are closed, and a new one is opened. At a MIDDLE event, the current cell is closed, and a new one is constructed. The result of these events is a free space that is broken down into simple sub spaces called trapezoidal cells. Since each cell is a trapezoid, simple back-and-forth motions can be used to cover each cell as illustrated in Figure 2.2(a). A depth-first-like graph search algorithm is used to obtain a path list that represents an exhaustive walk through the adjacency graph such that the path visits each node at least once.

Unfortunately the trapezoidal approach can become very inefficient in terms of the number of back-and-forth motions required. In order to address this problem, the classical Boustrophedon[1]

[1]From the Greek for ox-turning.

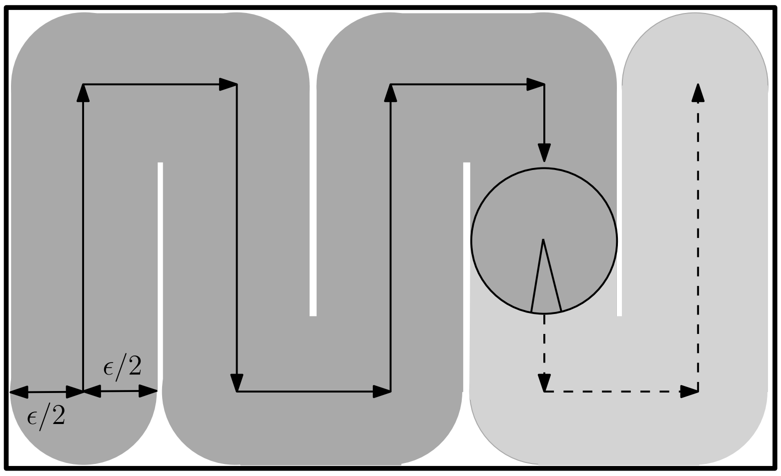
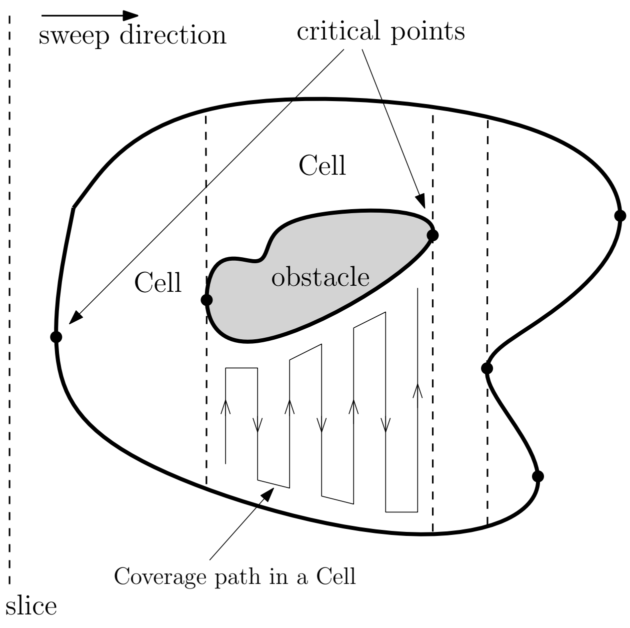


FIGURE 2.3: Boustrophedon motion. Here the robot carries a circular tool with diameter ϵ which extends the robot's sweep area over both of its sides. Darker paint area indicates the area covered already and the lighter paint area that will be covered following the back-and-forth path. Normally the robot would move so that adjacent swaths touch. In this figure space has been left between passes of the algorithm for purposes of exposition.

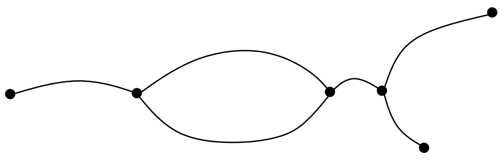
Cellular Decomposition (BCD) algorithm was introduced by Choset in [40, 42]. This decomposition is an enhancement of the trapezoidal decomposition that merges all cells between IN and OUT events into one cell to reduce excessive lengthwise motions as represented in Figure 2.2(b). The BCD relies on changes in connectivity of a slice to determine the existence of an event instead of exploiting the structure of polygons to determine IN, OUT, and MIDDLE events.

To use BCD for area coverage, an adjacency graph is created where a node represents each cell, and an edge is used to connect pairs of adjacent cells. Each node in the graph is visited following an exhaustive walk. At each node, the cell is covered by boustrophedon (simple back-and-forth) motions. Figure 2.3 illustrates sample back-and-forth motion. It is assumed that the robot is a point in $\mathcal{W} = \mathbb{R}^2$, and that the robot carries a circular tool with diameter ϵ which extends the robot's sweep area over both of its sides. This enables the robot to paint or mow part of the entire space up to distance $\epsilon/2$ from both sides of the robot through a continuous movement.

Building on work presented in [40, 42], Acar et al. [27] presented an exact cellular decomposition algorithm known as the *Morse decomposition algorithm* in which the cells have simple structure and can be defined for non-polygonal terrain. The BCD algorithm is a specific case of the Morse decomposition algorithm in which a straight line called a *slice*, $h(x, y) = x$, is swept from left to right over the entire terrain. The intersection of the slice and the area to be covered is followed by the robot. A *cell* is a region defined by the boustrophedon decomposition where slice connectivity does not change. In other words, no obstacle breaks the connectivity of the



(a)



(b)

FIGURE 2.4: (a) Boustrophedon cellular decomposition. (b) The corresponding Reeb graph.

slice inside each cell. A *critical point* is a point on an obstacle which causes a change in slice connectivity. Thus, the free space is divided into sub regions (cells) of constant slice connectivity. Each of the cells can be covered with a simple vertical back-and-forth sweeping motion [27].

Each of the decomposition algorithms result in a collection of regions that can be easily swept via motions of the robot using back-and-forth motion. All that remains is to develop a strategy for moving the robot from one cell to the other. A Reeb graph [53] represents the topology of the cellular decomposition that is dual to adjacency graph where the nodes of the Reeb graph represent the critical points and the edges represent the edges connect the neighbouring critical points, i.e., correspond to cells [26]. A BCD and its Reeb graph are depicted in Figure 2.4.

The cellular decomposition and Reeb graph may result in inefficiencies in terms of the direction of coverage, how to move through the Reeb graph, and also in terms of where the start and end points are in the individual cells of the decomposition. Mannadiar and Rekleitis [80] presented a new optimal algorithm based on BCD for the complete coverage of a known environment with obstacles. The algorithm leads a mobile robot through a sequence of areas to be passed without wasting energy or time. The optimal solution to the Chinese Postman Problem (CPP) [51] from graph theory is adapted for the calculation of cell ordering. The CPP is used to find an order for traversing the Reeb graph such that the robot traverses through all the edges at least once. When the graph has an Eulerian circuit [35], that circuit is an optimal solution. Otherwise, the optimization problem is to find edges with odd degree vertices to make the graph Eulerian by doubling of the edges so that the resulting multigraph does have an Eulerian circuit. The solution to the postman problem in the original graph is obtained by finding an Eulerian circuit for the new graph. By splitting selected cells into two components, the single cell coverage used in the BCD algorithm is modified in order to eliminate repeat coverage.

A critical underlying assumption for these algorithms is that the vehicle is holonomic (a holonomic vehicle can turn in place and move independently in all directions). When actually applying one of these algorithms using real robots it may be necessary to have the robot undergo additional motions in order to actually make the instantaneous motions assumed by the (theoretical) algorithm.

In order to deal with the potential time inefficiencies associated with the complete coverage of an unknown environment when the robot finishes one cell and has to move to another cell, an algorithm for online complete coverage for autonomous cleaning robots in an unknown environment, called BA* can be employed. BA* is presented in [104] and is based on boustrophedon motions and the A* search [64]. An unvisited cell is covered by the robot using a single boustrophedon motion until the robot reaches a critical point. To cover the next unvisited cell at the critical point, the best backtracking point (which is found by applying a greedy strategy and an intelligent backtracking mechanism) is determined and this is used as the starting point of the next boustrophedon motion.

Two algorithms: Straight Field Coverage Algorithm (SFCA) and, Curved Field Coverage Algorithm (CFCA) are presented in [62] and their efficiency is evaluated using a simple two-wheeled agricultural vehicle. The algorithms deal with the field bounding box instead of the field polygon itself. These algorithms are capable of handling fields with any shape in a relatively short computational time. In both SFCA and CFCA, initially n tracks parallel to the field edges from the inside are generated to be used as row headlands. In the next step, SFCA uses the longest edge of the field as the reference edge for generating parallel rows, while CFCA selects longest set of neighbouring edges as the reference rows and parallel rows are generated parallel to it.

Many of the algorithms described above assume point-robot like motion. Kinematic motion constraints are considered in the literature as well. In the motion paths identified in BCD-like algorithms frequent turning motions are assigned to the robots. Such motions may not be possible for certain classes of robot vehicles. In order to address this problem in non-holonomic robots, a continuous steering control is required to replace the zig-zag motion used by BCD-like algorithms. For example, complete coverage control for non-holonomic mobile robots is considered in [60]. In this approach, first, a rectangle is covered with non-overlapping disks, and a complete coverage path is planed passing through all centres of the the disks using a neural network. Using parametric polynomials, a smooth trajectory is generated following this path. Finally, continuous steering control is used to track the planned path based on the differential flatness of the system (a system is differentially flat if there exists a set of outputs, such that all states and inputs can be determined from these outputs and their finite-order derivatives).

In [105], the optimal coverage algorithm presented by [80] is extended for the general class of non-holonomic robots in the aerial robotics domain using a set of generated waypoints outlining the desired coverage path, which is then given as input to a robot motion controller. The authors also investigated the quality of the coverage path by changing the direction of coverage (sweep direction) before running the BCD algorithm, since fewer number of turns and longer straight path are more desirable for non-holonomic vehicles. They proposed three different strategies for sweep direction: setting it orthogonal to the dominant edge orientation for obstacle boundaries, aligning it directly with the distribution of the free space, and aligning it to be perpendicular to the dominant wind heading. Their experiments were performed over 100 km of successful coverage flights with a fixed-wing vehicle, also thousands of kilometers of flight in simulation. Their testing validated the robustness and efficiency of their proposed approach, and investigated the effects on the quality of coverage of different motion planners, the sweep direction, and environmental factors such as wind.

2.2.2 Multiple robot sensor coverage

Algorithms for single agent coverage can be adapted to the problem of multiple robot sensor coverage in a number of different ways. A good area coverage algorithm using multiple robots should fulfill the following criteria [67]:

- Completeness. The algorithm should provide complete coverage. As discussed below, many implementations employ assumptions for the environment and sensor that do not guarantee completeness.

- Robustness. The algorithm should be robust to failures of individual robots or communication between the robots.

- Efficiency. The algorithm should provide coverage in as short a period of time as possible.

There is, of course, a trade off between these three criteria. For the coverage task, the choice of multi-robot coverage approach strongly depends on the type of communication that exists between the robots. When the robots operate under a line-of-sight communication restriction, the robots must remain in close proximity. When communication between the robots is available without any restrictions, the robots can disperse through environment to cover the area in parallel and constantly update each other on their progress. Previous work in 2-Dimensional (2D) sensor coverage by a robot collective is described below.

Approximate cellular decomposition

In [66] Hazon and Kaminka, the single-robot STC algorithm presented in [56] was adapted to a multi-robot system and the *Multi-robot Spanning Tree Coverage (MSTC)* algorithm developed. Their offline algorithm (the robots have a map of the area) for multi-robot coverage of a terrain guarantees completeness, robustness and efficiency (see Figure 2.5). MSTC first builds a spanning tree S for the graph G which covers the environment. S is then divided into sections, $S_0, ..., S_n$, where n is the number of robots in the system. Each robot is then assigned an area to cover. Two versions of the MSTC algorithm are presented by Hazon and Kaminka [66]: non-backtracking MSTC, and backtracking MSTC. In the non-backtracking algorithm the robots simply move in a counter-clockwise direction along the spanning tree path until they reach the initial position of the next robot. In the backtracking algorithm the robots can backtrack over parts of their coverage path. If a robot does not finish or does not respond after a specified period, that robot is assumed to have failed and the robot in the section behind picks up that portion of the work.

In [67] Hazon and Kaminka present an *optimal polynomial time coverage algorithm.* Given an initial spanning tree and the initial locations of the robots, the algorithm is similar to the backtracking MSTC algorithm described above. If the robots go back and forth along the

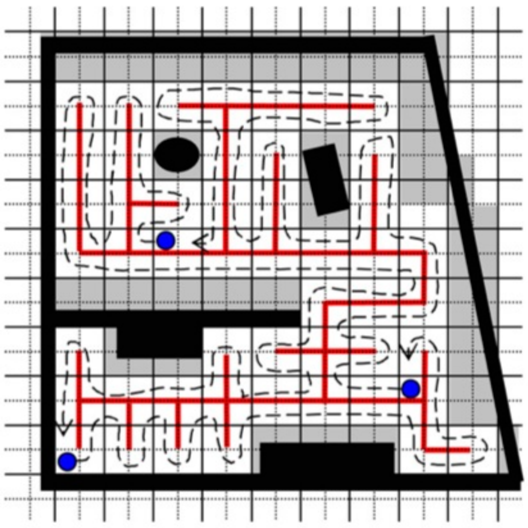


FIGURE 2.5: The grid, the spanning tree and the paths for three robots. Figure reprinted from [67].

given spanning tree, optimality is guaranteed. In [30], Agmon at al. focused on the challenge of constructing a coverage spanning tree for both online and offline coverage that minimizes the time to complete coverage. First, they investigated a polynomial time tree-construction algorithm for offline coverage. Second, an algorithm for online coverage of a finite terrain based on spanning-trees is provided. The algorithm presented in [30] guarantees completeness and linear time coverage with no redundancy in the coverage. In addition, this algorithm provides robustness to the failure of individual robots.

Exact cellular decomposition

In [96], Rekleitis et al. developed a family of coverage algorithms that employed two robots to cover an unknown environment using a visibility graph-like decomposition. In order to reduce odometry errors, the robots are also used as odometry beacons in the algorithm. However, the algorithms do not address the failure of a robot.

A multi-robot coverage method based on BCD is presented in [97] and [72]. The algorithm presented by [97] works even under the restriction that communication between two robots is available only when they are within the line of sight of each other. An improved algorithm for multi-robot coverage with unbounded communication is provided in [72]. In this algorithm, the robots are initially distributed through space and each robot is allocated a bounded area to cover. The area is decomposed into cells where each cell width is fixed. This algorithm has been demonstrated to be robust to failures, however, it has not been shown to be complete.

For the coverage task, since the choice of multi-robot policy strongly depends on the type of communication that exists between the robots, operating the robots as a team helps to minimize repeat coverage. In [98], Rekleitis presented two algorithms based on BCD for the restricted communication and the non restricted communication scenario.

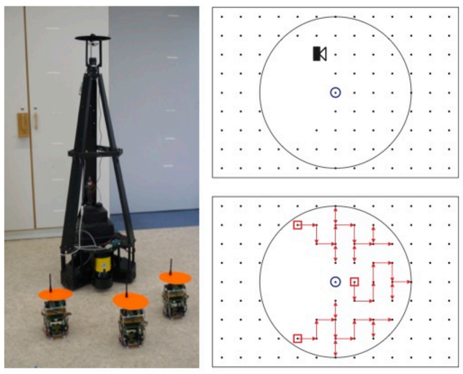
A 2D boundary coverage algorithm for multiple robots is presented in [50]. In this work, a team of robots must visit all points on the boundary of the 2D target environment. The boundary coverage problem is converted into an equivalent graph representation. To plan the visiting routes of every robot, a heuristic search is used. This algorithm provides complete coverage of the boundary by balancing the routes between the robots.

Teams of heterogenous robots

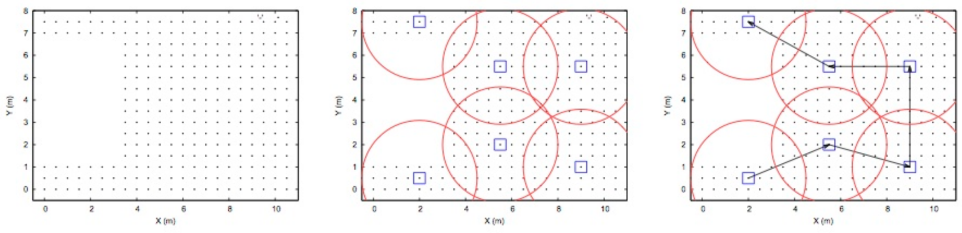
Parker [87] presented one of the earliest research demonstrations of heterogeneity in mobile multi-robot teams. She developed a number of algorithms for robot teams that were able to compensate for heterogeneity in task allocation and execution. Parker at al. presented an approach in [88] for heterogeneous mobile sensor network deployment using robot gathering and line-of-sight formations. In this approach, the deployment is accomplished through the collaboration of three types of robots. The first type is a mobile *Sensor Node* which has the ability of moving and acoustic sensing but cannot localize itself and avoid obstacles. Two types of *Helper* robots with more capability are used to herd *Sensor Node* robots through the environment: *Leader Helper* robots have the capability of localizing and path planning in the environment; *Follower Helper* robots have the capability of detecting the relative pose of other robot team members using a vision system. It is assumed that all robots can communicate with each other to share their information.

Later, in [89], Parker at al. presented the design of autonomous behaviours for tightly-coupled cooperation in heterogeneous robots team. This work was part of a wider set of experiments by Howard [70]. This work was specifically focused on robot exploration and deployment rather than on area coverage. A large number of robots were deployed as sensor nodes into an unexplored building guided and controlled by more capable robots which used them to map the building's interior to perform detection and tracking of intruders.

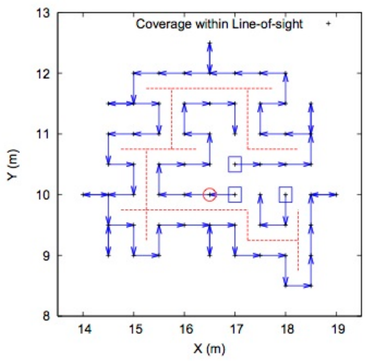
An approach to the area coverage problem using a team of heterogeneous mobile robots and considering line-of-sight conditions (LOS) is presented by Hofmeister at al. in [68]. This work extends the work presented by Hazon and Kaminka [67]. A large number of inexpensive and



(a)



(b)



(c)

FIGURE 2.6: (a) Heterogeneous team of mobile robots. Rectangles depict the starting positions of the child robots and arrows their roadmap graphs. The position of the parent robot is depicted by a circle. (b) Parent path plan (c) Child path plan within parent's LOS. Figures reprinted from [68, 69].

small child robots with restricted sensing and computation capabilities and a parent robot with state-of-the-art sensors and sufficient computation power are employed (see Figure 2.6(a)). They showed that area coverage can be performed in a fast and efficient way by using a larger number of inexpensive small child robots that lack sensors. In addition, the effect of the LOS radius of the parent robot on coverage time is discussed (see Figure 2.6). Later, in [69], Hofmeister et al. used the algorithms presented in [68] to build an image-based map of the environment using a team of heterogeneous robots with different capabilities.

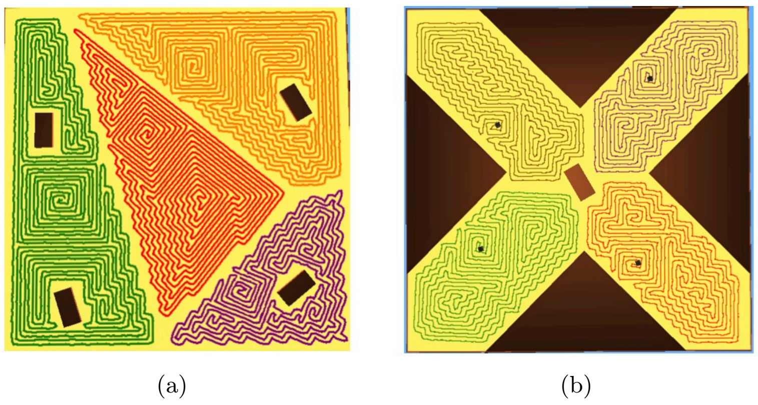


FIGURE 2.7: Robots cover (a) using Voronoi Partition-based Coverage (VPC) algorithm for a square environment, and (b) a X-shaped region within this environment using VPC algorithm. Figure reprinted from [61].

Distributed area coverage algorithms

The main assumption that underlies most multi-robot distributed coverage algorithms is that the environment can be decomposed into a cellular structure (exact or approximate decomposition) before deploying the robots. Such algorithms typically focus on coverage algorithms that allow the robots to visit these cells while trying to achieve desirable outcomes such as completeness and non-redundancy [67]. These decompositions are usually either assumed to be performed online or alternatively the space is decomposed into cells that are aligned with the footprint of a robot. There are other approaches that investigate distributed multi-robot coverage techniques that handle both decomposition and coverage of the environment in a distributed manner (e.g., [61, 82, 90]). Such approaches partition the space in the environment and allocate each robot to cover an allotted sub-region. Min et al. [82] partitioned the space into equal sub-areas and then assigns each robot to cover one sub-area. They used a limited motion and sensor model and assumed global communication. In order to facilitate cooperation, negotiation between agents was used.

In [61] Guruprasad et al. presented the Voronoi Partition-based Coverage (VPC) algorithm (see Figure 2.7) in which each of these regions are subsequently covered by a robot using any existing single-robot coverage algorithm (such as spanning tree coverage). This algorithm is robust to robot failures in that should a robot fail to report, the work of covering the missing area is assigned to another robot team member.

A sensor-based coverage planning for multi-robots considering the energy capacities of the mobile robots is presented by Parlaktuna at al. [90]. Initially, the environment is modelled using a

Generalized Voronoi diagram-based graph to guarantee complete sensor-based coverage. Chinese Postman Problem (CPP) [51] and/or Rural Postman Problem (RPP) [91] solving techniques are used for initial route planning. The initial route is partitioned amongst the robots by considering robot energy capacities. This work intends to increase the efficiency of the solution to the multirobot coverage problem; but the solution does not consider robot failures. Later in [86], they presented a fault-tolerant control architecture for sensor-based coverage using Multi-robots. A heartbeat strategy is used to detect the failed robots. The heartbeat strategy system for an n-robot team is implemented by sending and receiving messages via the communication agent that is responsible for interaction with the other robots. Each robot periodically sends heartbeat messages known as "I'm alive" messages to other robots in order to inform them about its aliveness. The time between two "I'm alive" messages is known as a heartbeat period. If the delay time (the difference between the control time and the time of the last received heartbeat message) is bigger than the predefined duration which is greater than the heartbeat period, a failure is reported.

2.3 2D pose estimation for a robot collective

Localization has always been a key problem for both indoor and outdoor mobile robots. For many autonomous navigation tasks the mobile robot must be able to localize itself relative to some external frame of reference. Localization is defined as estimating the pose of a robot with respect to its environment. For ground contact robots the pose consists of the robot's position (x, y) and heading (θ) in space. Here we are interested in the 2D version of the problem. For details on the general version of the problem see [49].

Localization for a team requires the localization of each robot in the team and hence the geometric relationship between team members within the same environment. Such localization could be performed using a number of different methods. Two extremes in this space are: independently, where each robot in a team determines its pose alone based on its own sensors; and cooperatively, which takes advantages of multiple robots to improve the positioning accuracy using other robots' sensors data. Under the cooperative localization (CL) model the sensor data from many robots is integrated to obtain a more precise localization of each robot.

Early work on cooperative localization was presented by Kurazume et al. in [74]. This work required coordination between the motions of the robot team for cooperative positioning. At least one member of the robot team remains stationary and acts as a landmark while the other team members are in motion. Improvements over this system are discussed and tested in [73].

Cooperative localization using a helical pattern was used by [99] to facilitate mapping. This approach deals with the problem of exploration of an unknown environment using two mobile robots. In order to reduce errors in odometry, one robot is equipped with a camera tracking system that allows it to determine its relative position and orientation with respect to a second robot carrying a helix target pattern that acts as a portable landmark. The accuracy of the vision based system is limited by discretization errors. Later work used a laser range finder and a three plane target produced estimates on the order of 0.01 m accuracy [95]. In even later work using distributed statistical estimators such as the maximum-likelihood Monte Carlo filters [55] the need for synchronized motion was relaxed. A KF-based distributed multi-robot localization approach is presented by Roumeliotis et al. [100]. Also, an EKF-based algorithm for outdoor use is described by Madhavan et al. in [78, 79] for the localization of a team of robots operating in unstructured environments.

When individual team members may not have absolute positioning capabilities, cooperative localization can be performed exploiting different sensors on board the team members. Martinelli et al.[81] extended the EKF approach introduced in [78] by considering the most general relative observation between two robots. In [31] Bailey et al. presented a distributed algorithm for performing joint localization of a heterogeneous team of robots. A central server is utilized to fuse measurements involving multiple platforms.

State estimation becomes a difficult problem when the communication channel for information exchange between all robots is not guaranteed. An algorithm is presented by Leung et al. that allows decentralized state estimation to be performed in a dynamic robot network when full connectivity is not guaranteed [76]. In this work, it is shown that in order to produce an estimate equivalent to the centralized state estimate, a robot only needs to consider its own knowledge of the network topology. However, the robot should ensure that the same task can be performed by all other robots in the network. In order to define when a state estimate equivalent to the centralized estimate can be made by the decentralized state estimator, checkpoints and partial checkpoints are defined. This ensures that the estimate is based on all past information. No knowledge of the number of robots in the network is necessary using this method. In addition, by exploiting the Markov property at partial checkpoints, the memory usage (although large compared to the centralized estimator) is limited.

Later, Leung et al., 2010 [77] extended their work on information flow in a robot network. This work also provided results on simulated localization for a broad range of network connectivity settings to give a comprehensive assessment on the performance of the approach.

A variety of different sensors have been considered for cooperative localization of a robot team. For example, [71] used robots equipped with omnidirectional vision cameras in order to identify and localize each other. While [47] used a pair of robots, one equipped with an active stereo vision and one with active lighting for localization. The various methods employed for localization use different sensors with different levels of accuracy. Some of these sensors are able to estimate accurately the distance between the robots, others the orientation of the observed robot relative to the observing robot, and some are able to estimate even the orientation of the observed robot. Sensors are not necessarily only on board the robot. For example, in [83], an overhead camera tracks the movement of the mobile robots in a workspace. Using feature recognition the pose of the robots are determined and localization performed.

2.4 Communication within the fleet of robots

One of the challenges in using multi-robot systems lies with dealing with the communication among them. Many multiple robot systems require wireless communication between the robots. One way of accomplishing communication in ROS [94] is to run a single ROS master (single roscore) on one of the robots and coordinating all communication through this one master. An active ROS system is organized as a collection of *nodes* that communicate using *messages*. Nodes advertise and receive messages through:

- A publish-subscribe mechanism, called *topics*, where multiple nodes can publish messages and multiple nodes can subscribe to receive messages
- A direct node to node communication mechanism called *services*.

The single master is used by all of the robots to initialize nodes and access parameters. Unfortunately, in a robot team with transient communication such an approach can lead to catastrophic failure of the robots. In a single ROS master model, when a robot loses communication with the master (the roscore) it cannot initialize new nodes, or messages, or services. In essence the system freezes. Clearly, a single ROS master (a single roscore) is inappropriate for a multi-robot system with the potential for communication failure. The obvious solution to this problem is a system with multiple ROS environments (there are multiple roscores) running, with one associated with each robot. In a multi ROS master model, robots retain complete independence (i.e., can initialize new nodes and wait for services even when disconnected), robots can change tasks, retrieve data and take corrective action (plan a path back to wireless range). Each robot acts as its own master, and the masters communicate with each other. Unfortunately, the various roscore's must deal with the potential for communication failures between them.

There exist ROS packages that support the communication of information between the multi roscore environments. The following is a brief summary of existing techniques [17]:

- **Master Sync.** [16] This approach synchronizes topics from one master to another. This package only works for two masters, and synchronizing topics. This is inappropriate for multi-robot systems with more than two robots.
- **Foreign Relays.**[15] This approach transfers topics to another master via an intermediary relay.
- **Multimaster FKIE.**[14] This approach synchronizes everything between multiple masters, not only two masters. This requires no or minimal configuration. The changes are automatically discovered and synchronized. This package consists of:
  - A *master_discovery* node, which connects to the ROS master to get/publish changes over the network.
  - A *master_sync* node, which connects to the discovered nodes to request the actual ROS state and registers the remote topics/service.
  - A *node_manager*, which manages the ROS multi-master system. It also manages nodes, topics, services, parameters and launch files.
- **Rocon Multimaster.** [21] The Robotics in concert (Rocon) project is trying to make multi-master ROS practical in order to solve multi-robot-device-tablet problems. In particular, it provides the gateway model, which is an upgrade the earlier *foreign_relay* and *master_sync* concepts.

While each robot in the fleet is in many ways independent from other members, it is important that each robot is aware of the current state of the other robots in the fleet, and most importantly, the current state of intra-fleet communication.

Several methods are proposed in the literature to deal with communication failure between elements of a group of robots. Two main approaches are the heartbeat and pinging strategies [34]. In the heartbeat strategy the agent periodically sends heartbeat messages known as "I'm alive messages" to other agents in order to inform them about its aliveness. In the pinging strategy, an agent periodically sends requests to the other agents and waits for their replies [43]. In terms of the work here, both mother and child robots need to know each other's state and the state of the communication status between the elements of the fleet. One particular property of the heterogeneous fleet being considered here, is that there is a single mother robot and multiple child robots. This does provide a number of simplifications over the more general problem with large numbers of different robots of different classes.

2.5 Summary

This chapter has provided a brief introduction to the problems of path planning, localization and communication for a team of heterogeneous aquatic surface robots. Coverage Path Planning is the task of determining a path that passes over all points of an area or volume of interest while avoiding obstacles. This task is integral to many robotic applications. In this chapter, we have reviewed coverage path planning and cooperative localization. Grid-based methods such the STC algorithm and its derivatives provide complete coverage on a discretized representation of the target environment. However, the grid representation of the environment used is highly sensitive to localization error.

For (2D) planar environments, the trapezoidal decomposition approach guarantees complete coverage for a known polygonal environment. An improvement to the trapezoidal decomposition is the boustrophedon decomposition, which generates shorter complete coverage paths for the same class of environments. The Morse-based cellular decomposition provides complete coverage paths for environments whose obstacle boundaries are differentiable. A method to detect the critical points that determine the cell boundaries using range sensor information allows a team of robots to perform Morse-based cellular decomposition coverage on-line.

Multi-robot coverage methods (either exact or approximate cellular decomposition) can reduce the time required to obtain sensor coverage by dividing the workload among the individual robot team members. By sensing individual robot failure and dynamically re-allocating robots, increased task robustness can be achieved.

For teams of robots, team pose estimation and communication/synchronization are key requirements. 2D pose estimation for a team can be performed in a number of ways. Although it is possible to have each robot estimate its pose independently, another approach is to have the robots rely on each other's sensors for pose estimation. In terms of ensuring reliable communication a common approach is to utilize a heartbeat protocol in which robots communicate not only their state but also liveliness information in order to know which robots are currently 'alive' and working on the coverage task.

Although there has been a number of advances in terms of coverage algorithms, a number of interesting problems remain. For example, how should non-holonomic motion properties be integrated within a coverage algorithm? For heterogeneous teams, how should localization and liveliness be operationalized? In the next chapter we outline an approach for addressing some of the problems associated with coverage path planning and cooperative localization for a heterogeneous fleet of ASVs. Specifically, a Boustrophedon decomposition approach is developed for a group of autonomous agents with holonomic and non-holonomic constraints.

Chapter 3

Area coverage using a heterogeneous fleet of robots

3.1 Introduction

A number of assumptions and simplifications underlie the work here in order to make area coverage problem with a heterogeneous fleet of robots tractable. First, the world is modelled as a plane with an exterior and interior polygon boundary. The flat plane assumption is a reasonable assumption for small bodies of water – we ignore effects related to the curvature of the Earth – and a polygonal approximation of the water boundary is sufficient for estimation of various water surface events. Individual robots are assumed to have specific sensor and communication capabilities. Although it would be possible to consider a wide range of different robot teams that might address this task, here we consider perhaps the simplest heterogenous robot team composed of two robot classes. The robot team is modelled as a heterogeneous team consisting of one well equipped robot (the *mother robot* – Eddy) and many smaller less equipped robots (*child robots* – Minnows). These platforms are described in detail in Appendices A and B. This team choice leverages existing hardware infrastructure and is a reasonable model for a wide range of applications in which command and control is centralized with a number of assistant vessels extending the sensing capabilities of the command ship. Unlike the case with many classes of terrestrial robots, here we are concerned with robots with dynamic motion constraints. Autonomous surface vessels are subject to complex dynamic processes related to wind and wave action, and many vessels utilize rudder/propeller structures for locomotion.

Any realistic solution must consider this complication. In terms of localization, we assume that the mother robot can solve the localization problem globally, but that collaborative and cooperative localization is required for the child robots. Therefore, the child robots must remain within sensor range of the mother robot. Building a system of autonomous surface robots to address the problem of sensor coverage requires a solution to a number of related problems in multi-robot systems. This chapter deals with the details of the various algorithms that have been developed in this work to address the following problems:

- A sensor-based area coverage algorithm for the various robots (mother and child robots) in a known body of water.
- A multi-robot localization system for the various robots with different localization capabilities.
- Ensuring robust communication among the fleet of robots.

3.2 Sensor-based area coverage

A wide variety of different solutions exist for the problem of sensor coverage, as described earlier in Chapter 2. In developing a coverage algorithm for a heterogeneous fleet, we begin by first considering a simplified version of the problem which is then extended to accommodate the realities of the fleet. In this simplified version:

- The environment is a known, 2D simple polygonal static environment containing simple static polygonal obstacles.
- The fleet consists of two types of point robots: a mother robot, and multiple child robots.
- Solutions to the localization process will require the child robots to remain within a maximum distance d of the mother robot, related to the communication/localization constraints of the fleet.
- Each child robot has a circular sensing footprint (f_c) while the mother robot has a circular sensing footprint (f_m).
- Initially the robots are deployed so that the children are within d of the mother robot.

Following the general approach of [26], we treat the problem of providing sensor coverage as one of decomposing the known environment into a sequence of areas that are then covered by the robots and their sensors. Here we adapt the BCD algorithm to the process of providing sensor coverage with a fleet of robots. Note that as the child robots must remain within a distance d of the mother robot. It is not possible to parallelize the coverage problem by allocating one robot to each cell in the decomposition as proposed by [98]. Rather, we seek efficiencies in terms of coverage by deploying the team of robots as a coordinated fleet within each cell. The following sections describe the process in detail. We begin by utilizing the BCD algorithm and construct a Reeb graph to generate an optimal and efficient path for the fleet of robots. Then, the organization of the fleet members in a rigid formation is addressee non-holonomic constraints are considered and an approach to deal with this constraint is proposed. Finally, we consider the problem of maintaining an effective communication channel between the mother and child robots.

3.2.1 BCD decomposition and Reeb graph

A brief explanation of the BCD algorithm is presented in Section 2.2.1. Here we provide details of the each step of the algorithm as used in this work. Figure 3.1 provides an overview of the various steps.

Input. The input to the BCD algorithm is a binary map separating obstacles from the free space that is to be covered.

Cellular decomposition.

- **Find the critical points.**
  - Sweep the environment along a *sweep direction* using a 1-pixel wide vertical strip named *slice*, $h(x, y) = x$. The sweep direction is along the x-axis (left to right) (see Figure 2.4(a)).
  - As the slice sweeps the space, it may intersect obstacles, which divides the sweep area into non-overlapping regions of free space. When the slice passes an obstacle, smaller slice pieces are merged into larger pieces. The total number of pieces in a slice in some column x is called the *slice connectivity count*.

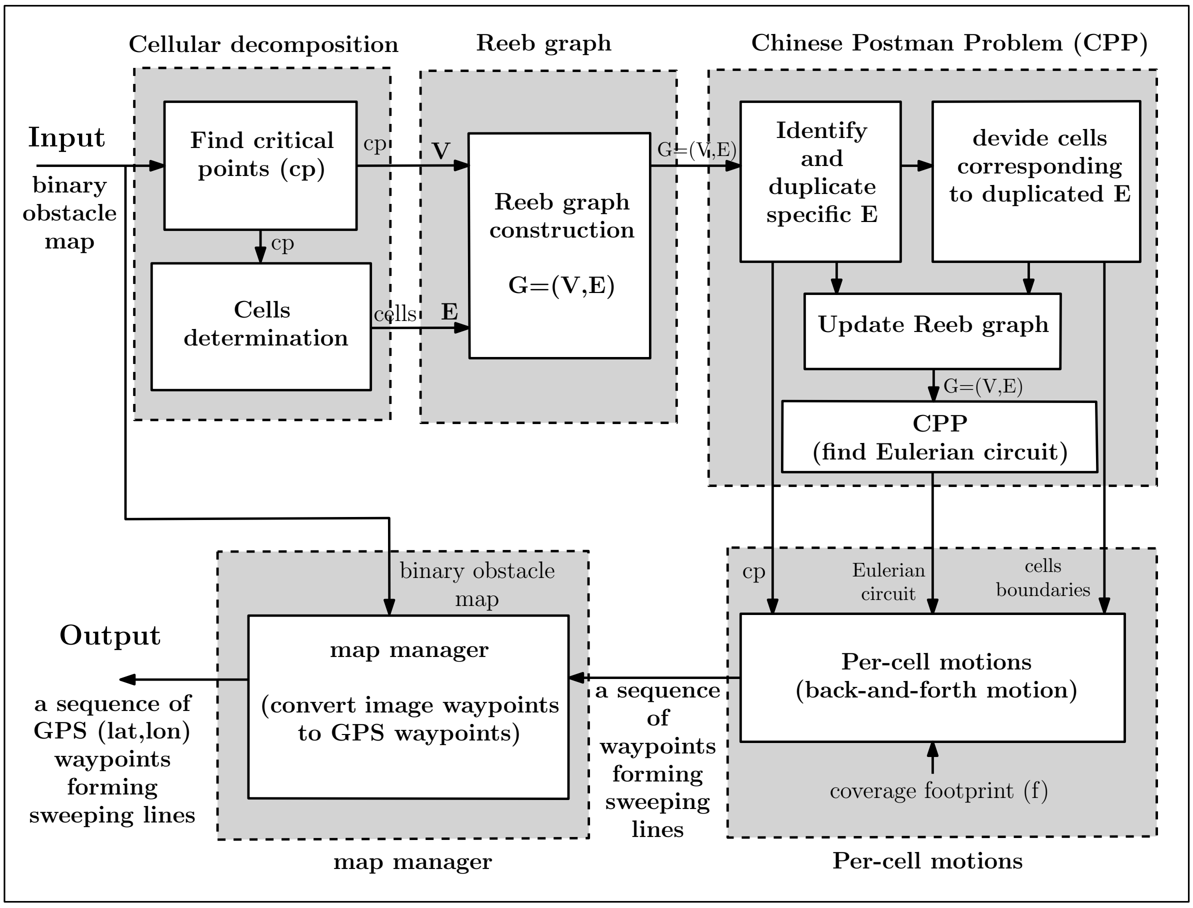
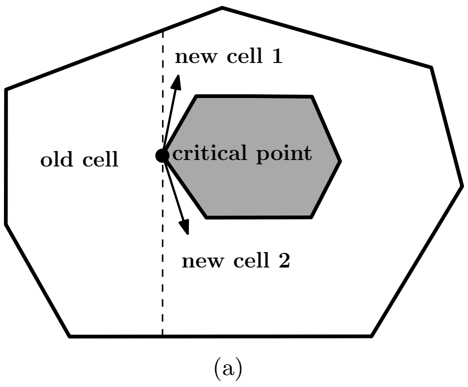

Cellular decomposition
Reeb graph
Chinese Postman Problem (CPP)
Input
binary obstacle map
Find critical points (cp)
cp
V
cp
Cells determination
cells
E
Reeb graph construction
G=(V,E)
G=(V,E)
Identify and duplicate specific E
devide cells corresponding to duplicated E
Update Reeb graph
G=(V,E)
CPP (find Eulerian circuit)
binary obstacle map
cp
Eulerian circuit
cells boundaries
Output
map manager
(convert image waypoints to GPS waypoints)
a sequence of GPS (lat,lon) waypoints forming sweeping lines
a sequence of waypoints forming sweeping lines
Per-cell motions (back-and-forth motion)
coverage footprint (f)
map manager
Per-cell motions

FIGURE 3.1: BCD decomposition and Reeb graph diagram.

– The points on obstacles which cause a change in the *slice connectivity count* are called *critical points*. The critical points are always located on the obstacle boundaries (see Figure 2.4(a)). There are two general types of critical points: critical points causing an increase in slice connectivity count and the critical points causing a decrease in slice connectivity count.

– The mid-point of vertical line segments are included as critical points as well, because they also causing change the connectivity of the slices.

- **Cell determination.** At critical points, *cells* are established by grouping connected regions of free space extending across multiple slices. The slice connectivity remains constant within a cell. Figure 3.2(a) shows how, at the critical point, the connectivity of the slice changes from one to two as the old cell is closed and two new cells are created. In Figure 3.2(b), at the critical point, the connectivity of the slice changes from two to one and two old cells are closed and a new cell is created.



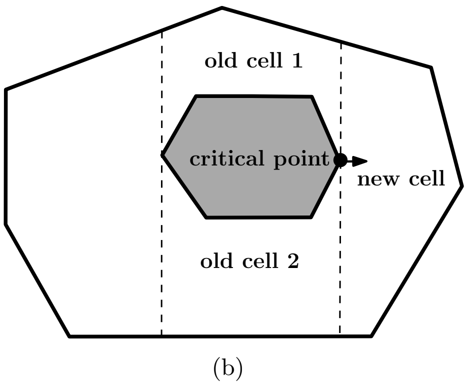


FIGURE 3.2: Cell determination with the BCD algorithm. (a) The connectivity of the slice changes from one to two. (b) the connectivity of the slice changes from two to one.

Reeb graph construction. Once the cell decomposition is constructed, critical points and cells boundaries are represented by the vertices V and edges E of the Reeb graph $G = (V, E)$ (see Figure 2.4(b)). Due to the nature of the BCD algorithm, every vertex V in the Reeb graph has a degree of one or three, i.e., each corresponding critical point is connected to exactly one or three cells (see Figure 3.3(a)).

Chinese Postman Problem (CPP). The Reeb graph is used as input to the CPP. The CPP is used to find an order for traversing the Reeb graph such that the robot traverses through all the edges at least once. When the graph has an Eulerian circuit [35], that circuit is an optimal solution [51]. An undirected graph has an Eulerian cycle if and only if every vertex has even degree. As constructed, the Reeb graph has vertices with degree 1 or 3. Revising this graph so that all vertices have an even degree allows for a more efficient traversal of the graph. Therefore, the solution to the CPP consists of the following:

- Identify and duplicate a certain set of edges to provide an even degree for corresponding vertices.
- To improve the efficiency of the coverage process (i.e., to prevent repeat coverage), cells corresponding to duplicated edges are divided into non-overlapping top and bottom cells. This strategy guarantees that no free space area will be covered more than once.
- The Reeb graph is updated with the new connectivity information.
- An Eulerian cycle through the updated Reeb graph is found.

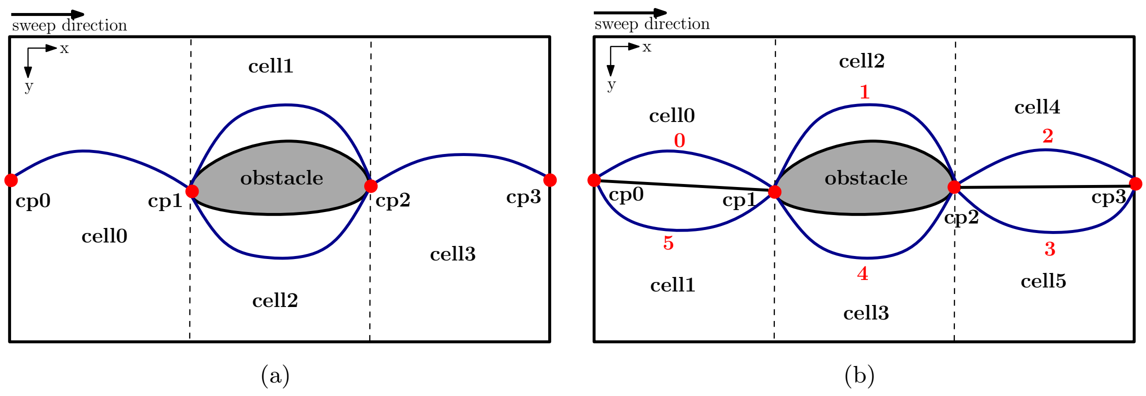


Figure 3.3: BCD, Reeb graph, and Eulerian circuit. (a) BCD and Reeb graph construction. (b) Optimal solution that corresponding numbers illustrate the Eulerian circuit, i.e., the order of coverage.

The output here is an Eulerian circuit that traverses through all of the connected cells in the environment. Figure 3.3(b) shows the decomposed cells and new cells for an environment. The corresponding numbers illustrate the order of cells coverage provided by Eulerian circuit.

Per-cell motions. After generating an Eulerian circuit to pass through all cells, the algorithm generates a path that covers each individual cell, following the order given by the Eulerian circuit. Since no obstacles exist in each cell, it is straightforward to cover them efficiently using a simple back-and-forth sweeping motions. This trajectory is represented by a sequence of waypoints. The coverage path within each cell has two parts: motion along a slice, and motion along the cell boundary as shown in Figure 3.4.

A critical parameter of the back-and-forth motion (Boustrophedon motion) is the *coverage footprint*, which measures the width between consecutive sweep lines. The footprint coverage width is dependent on the sensing range of the sensor that is used to achieve coverage.

Map manager. The waypoints provided by the last step (pixel coordinates) are given to a Google map manager, which utilizes the Google Static Maps API [24]. The Google Static Map service creates a map based on URL parameters sent through a standard HTTP request and returns the map as an image. The map manger also converts the waypoints (pixel coordinates (x, y)) to their corresponding GPS waypoints (latitude, longitude).

Output. The output is a sequence of GPS waypoints for a fleet of robots with *coverage footprint* f that should be traversed through all cells to cover the entire environment.

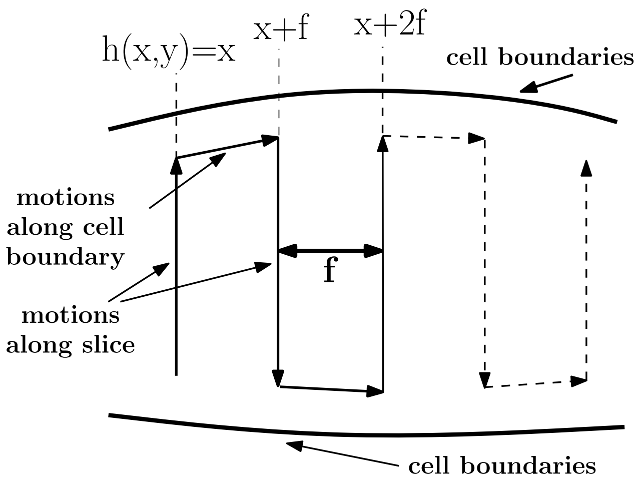


FIGURE 3.4: Boustrophedon path construction process, where f is coverage footprint and x is the slice parameter. The robot laps along the current slice, $h(x, y) = x$ or going along cell boundary by an f distance. x is also increased by this distance to form a new slice.

Figure 3.5 shows the optimal decomposed cells and Boustrophedon motions in each cell for a sample real world environment. The environment, its Reeb graph, the updated Reeb graph, new cell decomposition, and planned back-and-forth motion in each cell is shown in Figure 3.5(a)-(g). An Eulerian circuit is determined in order to identify the order in which the cells to be passed by the robot. The output from this process is a tour path connecting all of the environment cells as shown Figure 3.5(h)-(i).

3.2.2 Line abreast

The simplest way of utilizing the fleet to sweep out the space as they follow a Boustrophedon path within a given cell would be to organize the fleet members in a rigid formation such that their unified sensor coverage meets the requirements of the Boustrophedon algorithm. Two such configurations would be *line ahead* and *line abreast* as depicted in Figure 3.6 [32] although other strategies would also be possible. Line ahead is a regular naval line of battle, ships sailing head-to-tail in a single column. Line abreast involves the ships sailing side by side, the line being perpendicular to direction of the motion. Line abreast is useful for rapidly searching an surface area. Given the nature of the sensor footprint, line abreast provides a more efficient (but potentially less robust) sensing approach than a line ahead formation.

Individual robots are positioned line abreast with touching or intersecting sensor fields as shown in Figure 3.7. This provides the fleet with a combined sensor sweep a maximum $2nR_c+2R_m$ wide (where n is the number of child robots, R_c is the child robot sensing radius, and R_m is the mother

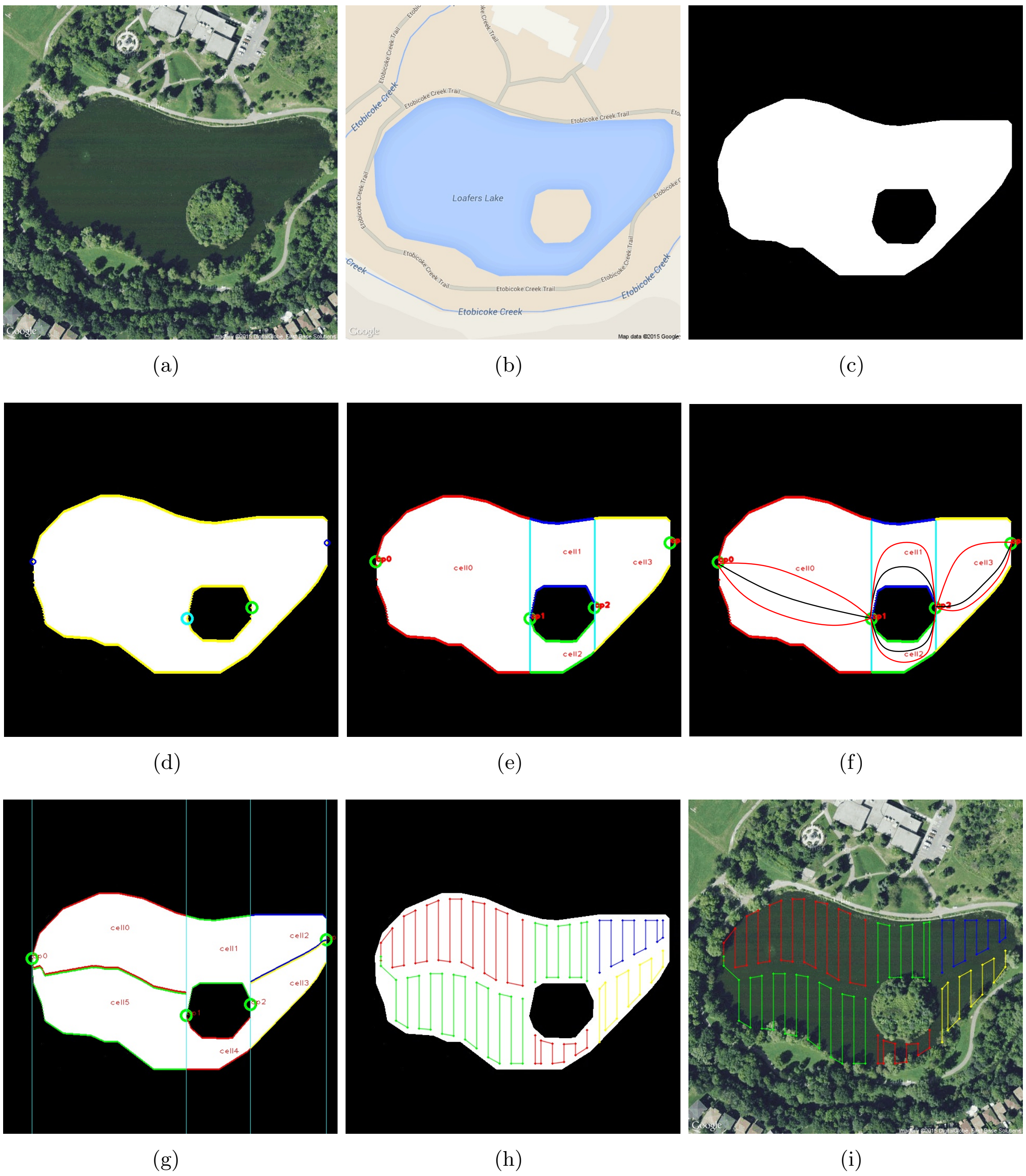


FIGURE 3.5: BCD and Reeb graph of an real environment. (a) Satellite map of "Loafers lake" located at Brampton, ON reprinted from Google. (b) Standard map of the lake. (c) The binary map that is the input of our algorithm. (d) and (e) depict the *critical points* and *cells* based on BCD. (f) The Reeb graph is shown in black lines and the updated Reeb graph is shown in red lines, which is constructed to have an Eulerian circuits. The formation of new cells are shown in (g). The corresponding cell numbers illustrate the order of cells coverage provided by Eulerian circuit. Planned motions are shown in (h) and (i).

robot sensing radius) and with a complex sensor footprint bow-to-stern. The Boustrophedon algorithm assumes that the sensor sweep contains no holes, so solutions in which the sensor footprints of the various robots do not touch would require significant modification to the basic

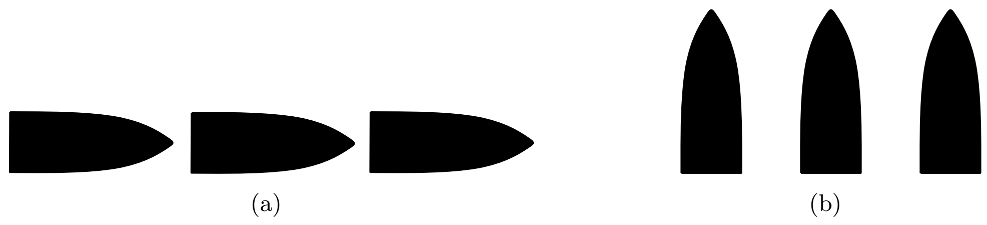


FIGURE 3.6: Formation of three robots. (a) *line ahead* formation. (b) *Line abreast* formation.

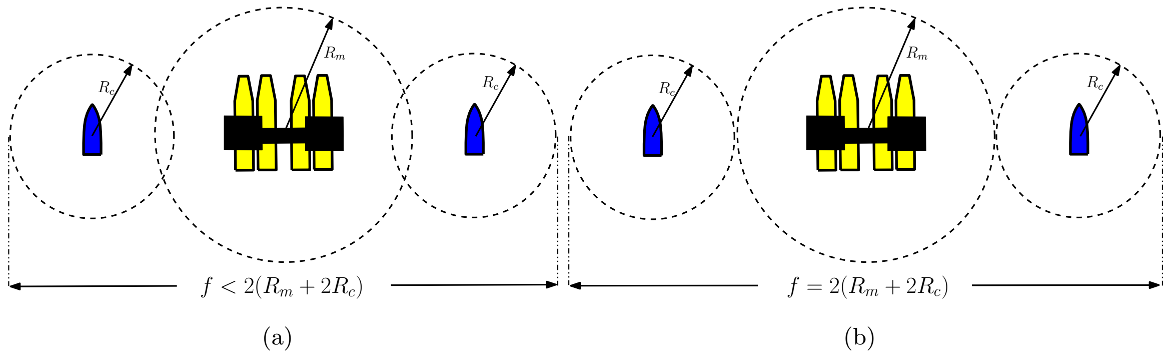


FIGURE 3.7: A heterogeneous fleet of robots in line abreast with sensor fields either (a) partially overlapping or just (b) touching, a combined sensor sweep (coverage footprint (f)) a maximum $2(R_m + 2R_c)$ wide.

sweep algorithm.

In terms of the sweep width (coverage footprint), the most efficient positioning of the elements of the fleet will be in *perfect line abreast* formation, thus providing a sweep width of $2nR_c+2R_m$. Note that solutions in which the formation of robots is on an angle, rather than at right angle to the direction of motion will also result in a reduced net sensor width.

3.2.3 Dealing with non-holonomic constraints

The solution developed in the previous section requires the robots to make very complex motions – motions that are not possible for non-holonomic members of the robot team. In addressing this issue, we first observe that the robots are capable of moving in line abreast formation, and this formation maximizes the coverage of the fleet's sensors in terms of the BCD algorithm. If we maintain line abreast formation but require the robots to move in their preferred direction, then the resulting motion of the robots within the fleet would be as depicted in Figure 3.8. Note that here we have not imposed non-holonomic constraints on any of the robots.

As the mother robot moves the child robots adjust their positions so that their relative position with respect to the mother robots remains unchanged. In this approach, the sensor footprint

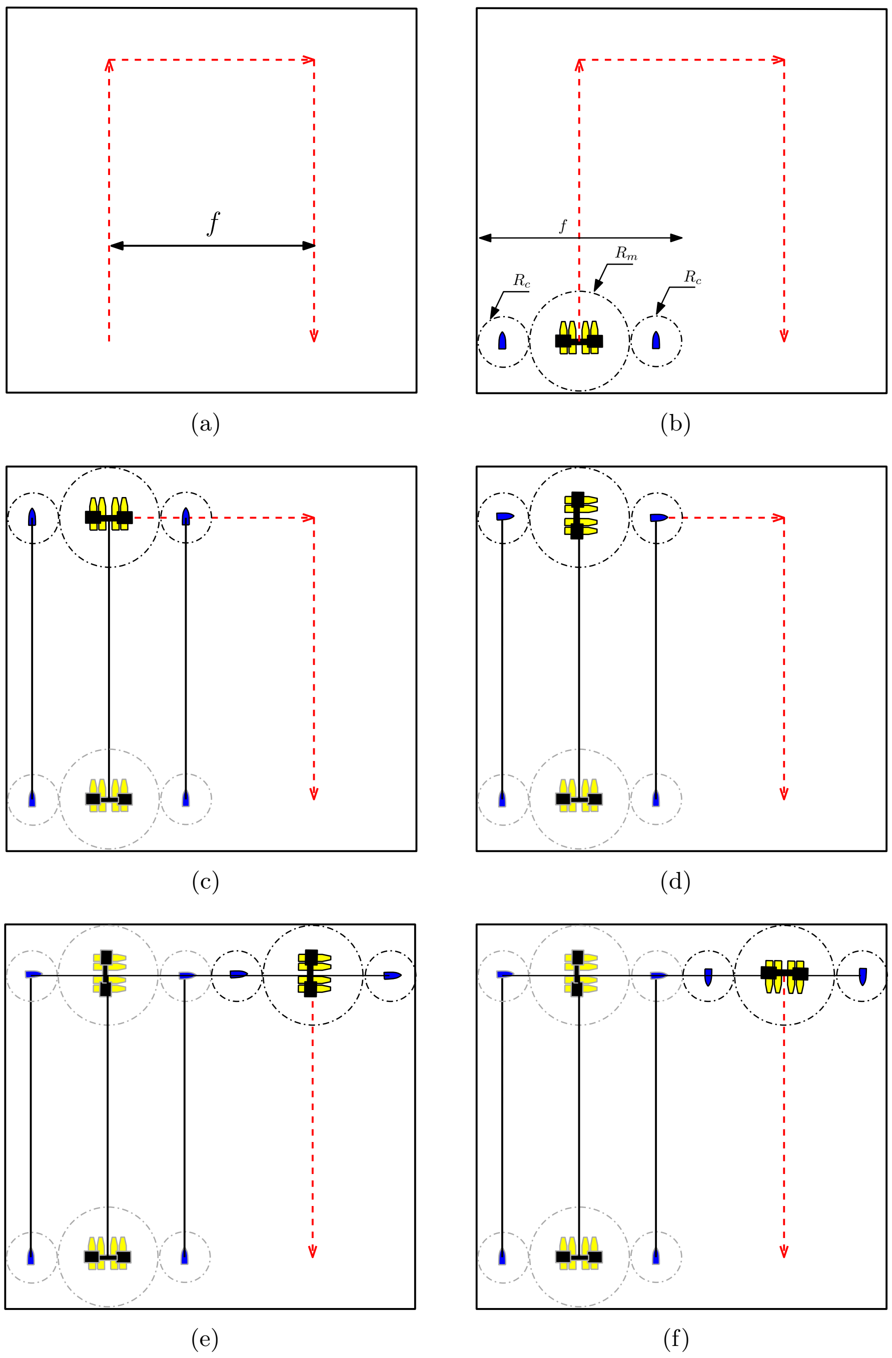


FIGURE 3.8: Line abreast approach without non-holonomic constraint. (a) depicts the fleet coverage path. (b) depicts the fleet members and their formation (a mother robot in the middle and two child robots in two sides of mother robot). Dotted circles denote the sensing area of each robots. (c)-(f) depict the mother and child robots motions in turns considering all robots a holonomic robot.

does not change as a function of time. In the BCD algorithm, individual cells are swept using back-and-forth motions of the sensor. For forward motions, the combined fleet sweeps out a complex region that is $2nR_c + 2R_m$ wide. Assuming that individual elements of the fleet are holonomic, the process of executing the turning maneuver at the end of a pass is a simple matter

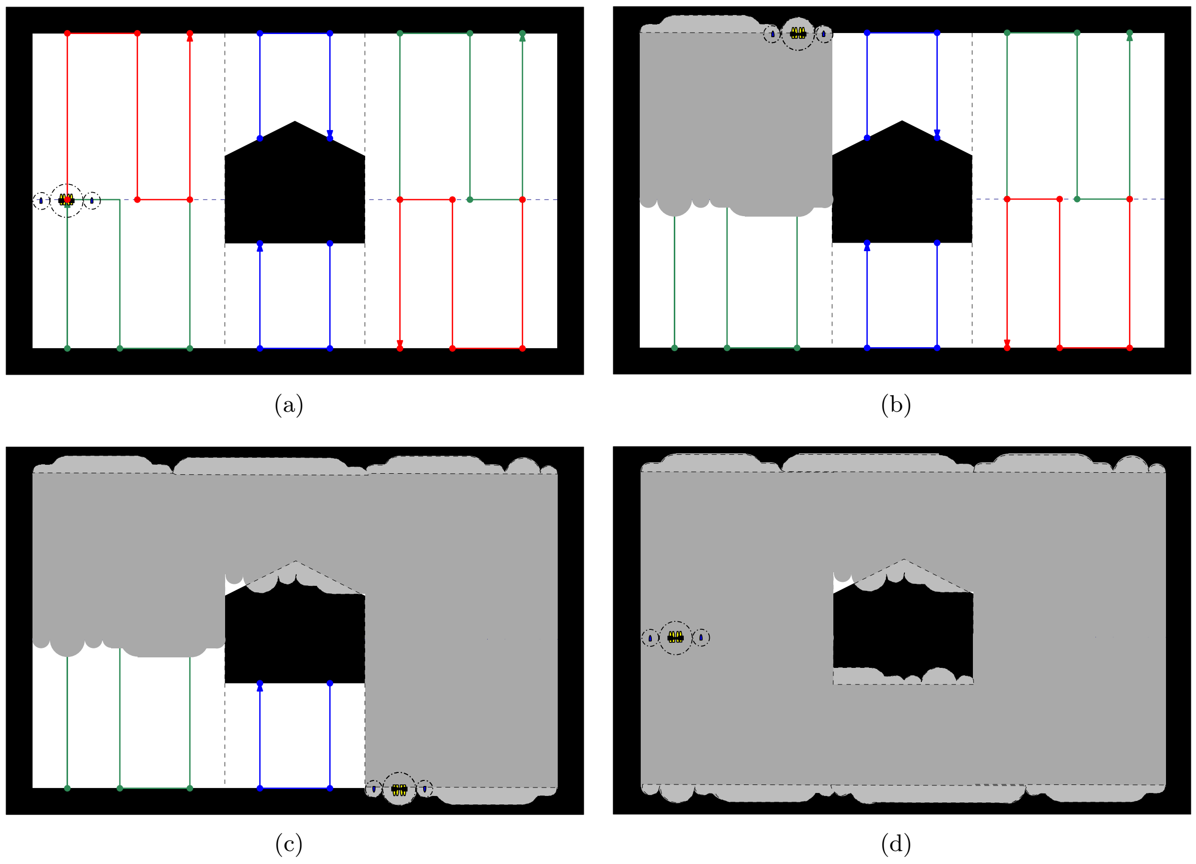


FIGURE 3.9: Fleet coverage with holonomic robots. (a) Boustrophedon cellular decomposition, the boustrophedon motion in each cell, and the fleet formation. (b)-(d) area covered in grey by traversing fleet following Eulerian order and path provided in each cell. The boundaries of the environment are dilated such that the robots can move to the absolute boundary of the swept space. Small “holes” may appear at the boundary of obstacles due to the large physical footprint of the combined fleet. Such holes are predictable and identifiable and can be covered by the mother robot operating in isolation or through some other method.

of moving the entire fleet over one sensor width and then moving down the next row, which is shown are Figure 3.8(c)-(f).

The basic sweep algorithm assumes that the sensor provides a single line of sensory information perpendicular to the direction of travel. Under this model the entire fleet must move to the actual boundary of the space being swept out. Practically, we dilate the boundaries of the environment slightly so that the robots can move to the absolute boundary of the “swept space”. Figure 3.9 shows a couple of snapshots of the fleet providing coverage of a polygon environment without considering non-holonomic constraints. Figure 3.9(a) depicts the BCD, the path should be covered by fleet in each cell, and the primary line abreast formation of the mother and child robots. Here the mother robot is located in the middle as the leader and two child robots are located on either side of the mother robot. In Figure 3.9(b)-(d), the area that has been covered

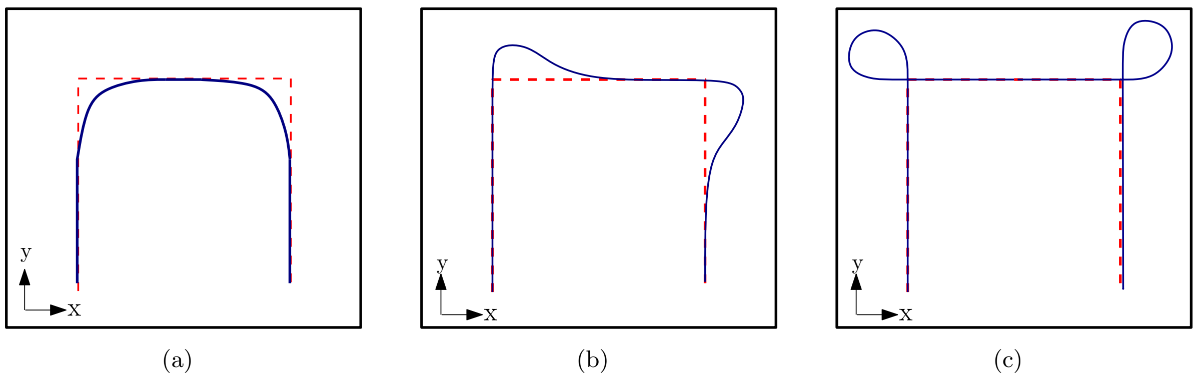


FIGURE 3.10: Three potential strategies for steering the non-holonomic child robots along the ideal path (shown in red). (a) is not desirable in that it will be difficult to ensure good coverage at the beginning and end of the motion along the y direction to the x direction. (b) has a similar issue but along the x direction and then along the y direction as the robot moves down the next leg of the path. Motion like (c) avoids both of these problems although it does require that there be free space for the robot to move in outside of the region being scanned.

by fleet is painted in grey. Due to the nature of the combined sensor footprint some slight *over scanning* of some areas is required in order to provide full coverage. Furthermore, if the cells are trapezoidal like the second cell, some other areas are missed as shown in Figure 3.9(c). Such regions are easily identified and can be covered later by the mother robot operating on its own.

Although the motions presented in Figure 3.8 and Figure 3.9 are possible for the mother robot, and this configuration is straightforward in terms of the BCD algorithm as the sweep region remains constant, it is problematic for the child robots. Specifically, it can cause elements of the fleet to execute motions that violate their non-holonomic constraints. As is the case with the mother robot, the child robots are non-holonomic. Unfortunately, unlike the mother robot, the child robots cannot change orientation independently of position. Therefore, an optimized and smooth continuous control trajectory for the child robots is required that takes into account the dynamic constraints of the child robots. Three potential strategies for steering the non-holonomic child robots along the ideal path are shown in Figure 3.10. Figure 3.10(a) is not desirable in that it will be difficult to ensure good coverage while navigating from along the y direction to the x direction (and vice versa). The strategy described in Figure 3.10(b) has a similar issue as the robot moves down the next leg of the path. Motion plans like that illustrated in Figure 3.10(c) avoid these problems although it does require that there be free space for the robot to move in outside of the region being scanned. Figure 3.11 illustrates this type of maneuver if we assume the child robots are non-holonomic and the mother robot is capable of holonomic motion. Figure 3.11(b) depicts the fleet members and their formation. Figure 3.11(c)-(f) depicts the mother and child robot motions during turns considering the

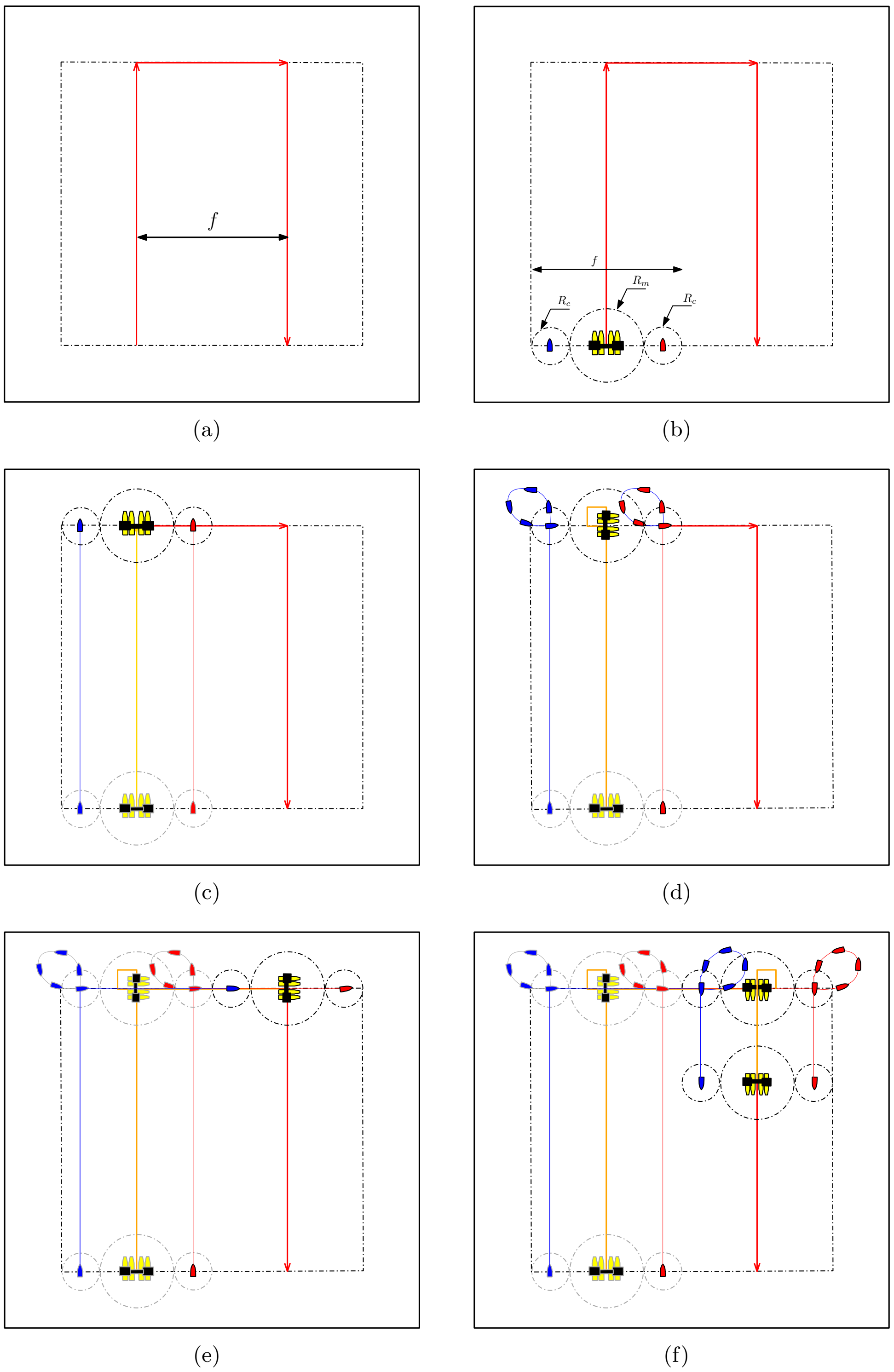


FIGURE 3.11: Line abreast approach with non-holonomic constraints. (a) depicts the fleet coverage path. (b) depicts the fleet members and their formation (a mother robot in middle and two child robots in two sides of the mother robot). Dotted circles denote the sensing area of each robots. (c)-(f) depict the mother and child robots motions in turns considering the mother robot as a holonomic robot and the child robots as a non-holonomic robot.

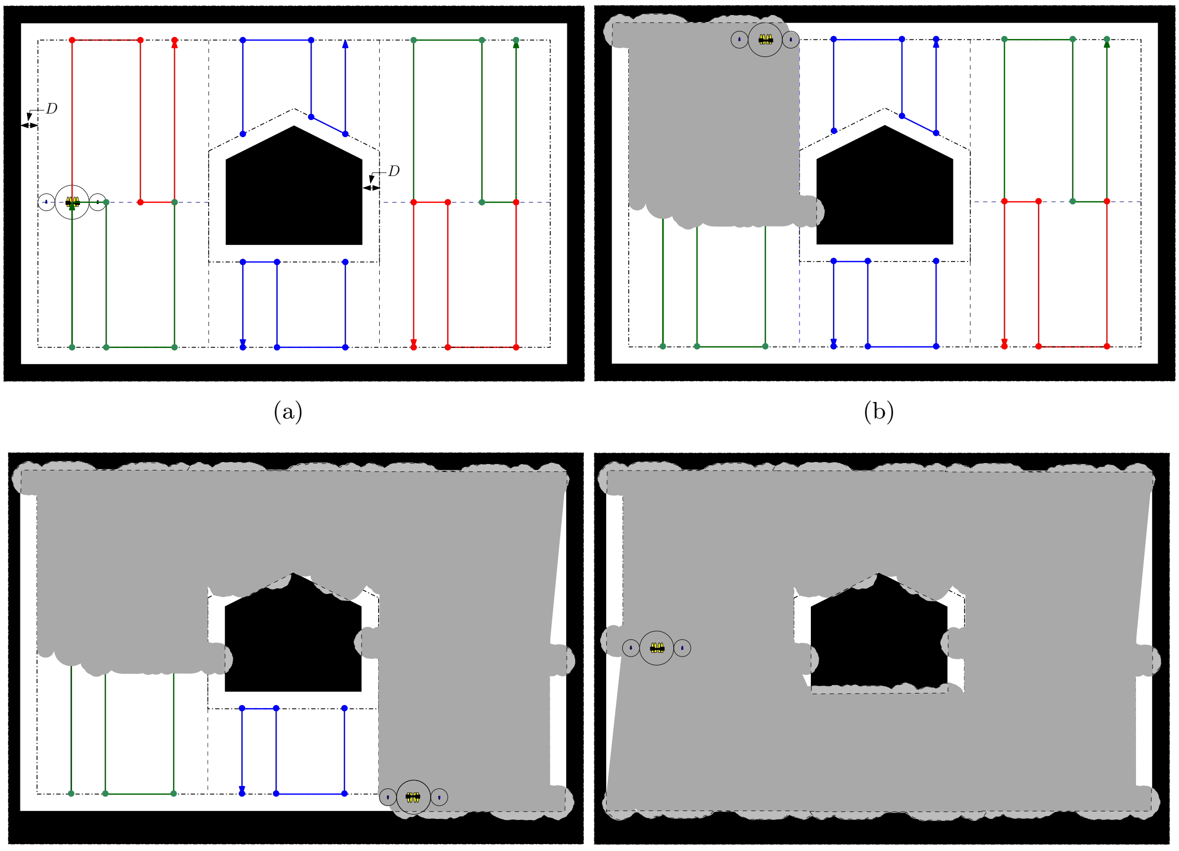


FIGURE 3.12: Fleet coverage with nonholonomic robots. (a) Boustrophedon cellular decomposition, the boustrophedon motion in each cell, and the fleet formation. (b)-(d) area covered in grey by traversing the fleet following Eulerian order and the path provided in each cell.

mother robot can change orientation independently of position and the child robots as a non-holonomic devices that must move in order to change direction

Using the full turn strategy for the child robots in turning motions requires the presence of a safe area around boundaries and obstacles which its size is determined from the child robot's safe turning diameter (D). Practically this means that the regions around obstacles and boundaries must be dilated so as to provide sufficient space for the fleet to maneuver. Figure 3.12 provides of snapshots of the fleet providing coverage of a polygon environment while incorporating the considering non-holonomic motion constraints associated with the child robots and $D = 2R_c$. The regions around obstacles have been dilated so as to provide sufficient space for the fleet to maneuver. Figure 3.12(a) depicts the BCD, the path that is to be covered by fleet in each cell, and the primary line abreast formation of the mother and child robots. In Figure 3.12(b)-(d), the area that have been covered by fleet are coloured in grey.

One final issue involves insuring that the motion of the mother robot does not intersect the motion of the child robots. If the mother robot executes the same motion trajectory as the child robots then this will be straightforward. Such an approach will also ensure that the fleet remains in close proximity to each other. Another approach is to have the mother robot execute simple rotation/straight line motions while the child robots execute their curved trajectories. The mother robot can then adjust its velocity along its straight line motion paths so that it remains away from the child robots. This is the approach followed in the experiments reported here, but either approach would be appropriate.

3.3 Multi-robot localization

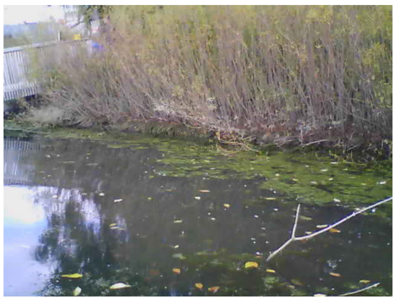
Localization for a set of robots in a common coordinate system requires the localization of each robot in the team and hence the geometric relationship between team members within the environment. Such localization could be performed using a number of different methods as described in Chapter 2. Here we utilize a cooperate localization approach. Under the cooperative localization model, the sensor data from the collection of robots is integrated to obtain a more precise localization of each robot. In the domain of the heterogenous system of surface robots used in this thesis, we assume that the mother robot (Eddy) is able to solve the localization problem via access to high resolution external sensors (e.g., DGPS, a tilt-compensated three axis digital compass and the like) but other solutions are required for the child robot localization. One possible solution to the localization problem for the child robots would be to equip the child robots with commercial GPS systems. Unfortunately such systems do not provide the necessary accuracy for localization of the child robots. Notwithstanding the accuracy and precision reported by the manufactures of such devices, positional errors in the range of 10's of meters are not uncommon, with considerable variability of the reported position. A more practical solution is to instrument the mother robot with sufficient sensing capability to determine the position (dx, dy) of each child robot relative to the mother robot and to utilize a compass on board each of the child robots to compute their absolute orientation (θ). Many solutions are possible for obtaining the position of the child robot relative to the mother. Here a visual system is employed. A 360 degree panoramic camera is mounted on the mother robot to track the movement of the child robots using coloured targets mounted on each of the child robots. Although commercial panoramic sensors exist, in order to deal with the wide range of visual conditions and in order to harden the sensor against environmental concerns, a custom sensor was constructed from a collection of eight networked web cameras. Figure 3.13 shows images from the individual cameras as the robot operates in the Stong Pond near York university, Canada. These eight images can be pasted together in order to provide a panoramic view of the robot's environment (Figure 3.13(i)) and the location of the child robots obtained relative to the mother robot using standard image processing techniques, as described below. The following sections describe the process in detail. We begin by defining the various coordinate frames used. Then, the camera model used for each camera is presented. Finally the localization algorithm and its accuracy are discussed. Note that although the eight cameras that make up the omnidirectional sensor can be thought of as a single video image, this is not necessary here. Rather, each camera can be processed independently and only the locations obtained by the individual cameras combined.

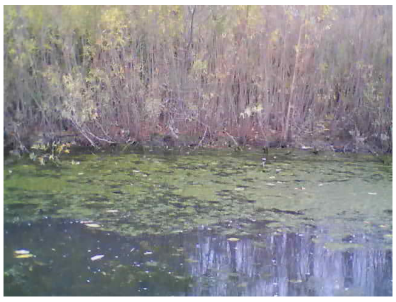
3.3.1 Coordinate frames

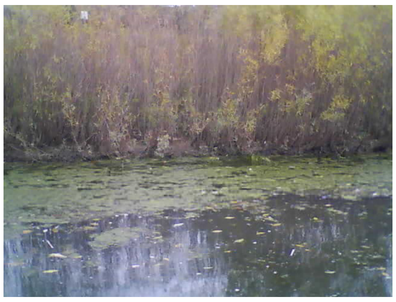
There are a number of coordinate frames used throughout this thesis:

- World frame
- Base station frame (local map frame)
- Mother robot frame (Eddy frame)
- Each child robot frame (Minnow frame, one per robot)
- Individual camera frames (eight)
- Individual image frames (eight)

These frames are illustrated in Figure 3.14. The world frame $(X_wY_wZ_w)$ is obtained by mapping latitude, longitude, and elevation onto the local level frame which is a fixed global reference frame and is the same as coordinate used by the global positioning system (GPS). The base station frame $(X_bY_bZ_b)$ is represented in the East-North-Up (ENU) cartesian coordinate system centred at the base station latitude/longitude/altitude. Due to the 2D nature of water surface, we eliminate the Z coordinate of elevation and refer to a location on the water surface and shoreline as the 2D point (X_w, Y_w) and (X_b, Y_b). The mother's robot body-fixed frame $(X_mY_mZ_m)$ and the child robot j's body-fixed frame's $(X_c^jY_c^jZ_c^j)$ have their origin located at the autonomous surface vessel's (ASV's) centre of rotation, which is assumed to be on the water's surface for


(a)

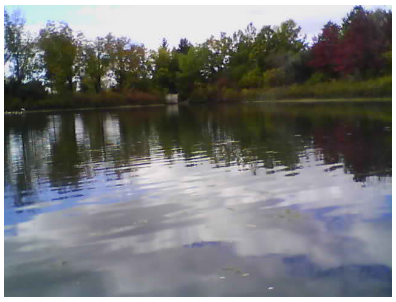

(b)

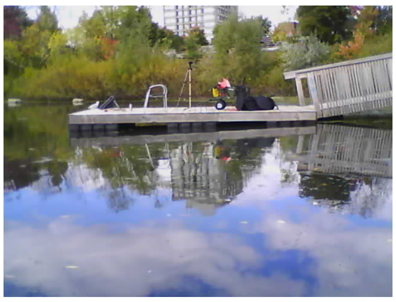

(c)

(d)

(e)

(f)


(g)


(h)

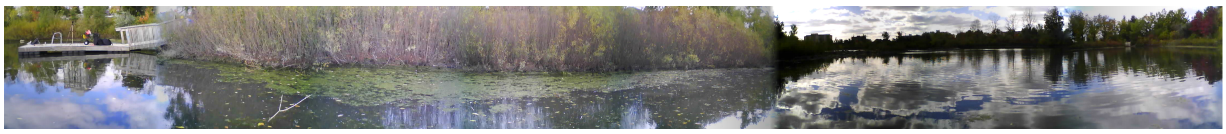

(i)

FIGURE 3.13: (a)-(h) Pictures of omnidirectional sensor. (i) Panoramic view. Note that there exist some regions without overlap, and no effort has been made here to correct the intensities returned from the individual cameras.

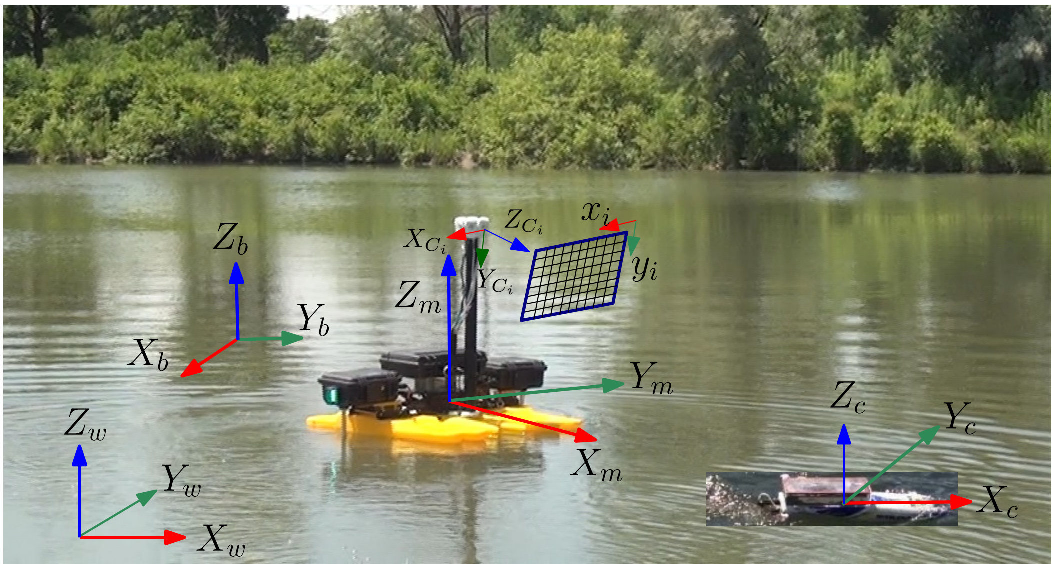


FIGURE 3.14: Coordinate frames. See text for details.

simplicity. Each camera frame $(X_{C_i} Y_{C_i} Z_{C_i})$ is fixed to camera i, with its origin at the centre of the camera. Each camera i's image frame (x_i, y_i) has its origin located at the upper left corner of the camera's image. Frame positions are expressed in meters except for the image frame which uses pixels.

3.3.2 Camera geometry

The cameras used by the omnidirectional sensor are modelled as pinhole cameras. Full details on pinhole camera geometry can be found in [54]. In this model, the geometric relationship between a 3D world point and its 2D corresponding projection onto the image plane is modelled using a perspective transformation. We exploit the fact that targets are constrained to the surface of the water which is assumed to be flat and perpendicular to the mast for the omnidirectional camera. The following sections describe the process of camera calibration, projection of planar points, and homography estimation.

Camera calibration

The material here follows the presentation given in [84]. Camera calibration includes the estimation of extrinsic and intrinsic camera parameters to establish a relationship between camera coordinate and a 3D world coordinate. A 3D point (X, Y, Z) is projected on the camera image

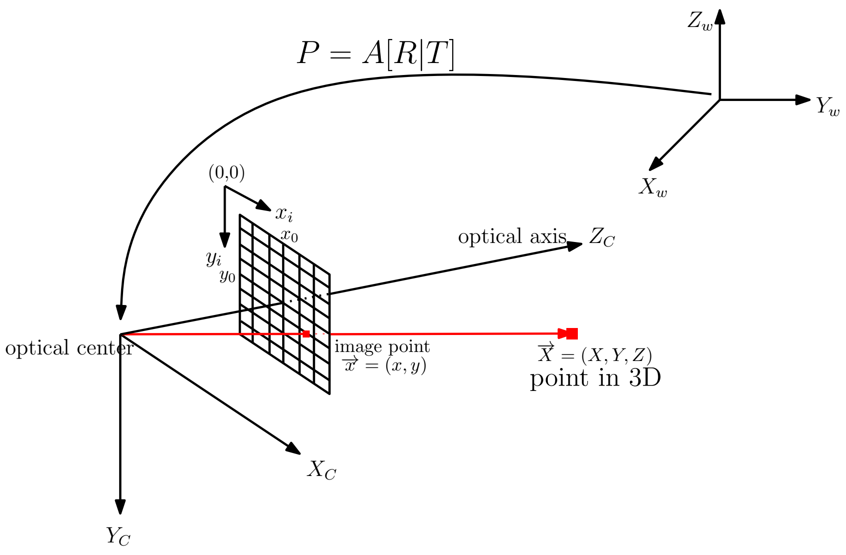


FIGURE 3.15: Camera geometry. Image reprinted from [84].

plane at the point (x, y) shown in Figure 3.15. The perspective transformation between the 3D world point and its corresponding image point is given by:

$$s\overrightarrow{x} = A[R|T]\overrightarrow{X} \tag{3.1}$$

Or less compactly,

$$s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.2}$$

where $\overrightarrow{X} = (X, Y, Z, 1)^T$ and $\overrightarrow{x} = (x, y, 1)^T$ are homogenous coordinates of a 3D world point and its corresponding image point, s is scaling factor, A is a camera matrix showing intrinsic parameters f_x, f_y, x_0, and y_0. f_x and f_y are the focal lengths expressed in pixel units, and (x_0, y_0) is the principal point (usually the image centre). The R and T parameters define the rotation and translation of the camera coordinates with respect to the world coordinates called the extrinsic parameters. The R values make up a rotation matrix and thus the camera model requires six extrinsic parameters (3 rotation and 3 translation), and 4 intrinsic parameters (two focal lengths and a principal point location).

In order to calculate intrinsic and extrinsic parameters, most calibration approaches establish correspondences between 3D world points on a calibration target and their 2D projections on the image plane of the camera. See [65, 102, 106] for some examples of camera parameter computation. Under the assumption that all points of interest lie on (or are very near to) the surface of the water and that the surface of the water is flat, a simpler solution is possible to compute the perspective transformation of planar points.

Projection of planar points

Due to the 2D nature of water surface, then without loss of generality we can treat the water surface as being the surface $Z = 0$ and we can eliminate the Z coordinate and refer to a location on the water's surface and shoreline as the 2D point (X_w, Y_w). For planar surfaces, the 3D to 2D perspective projection reduces to a 2D to 2D transformation. Then the 3×4 projection matrix ($P = A[R|T]$) reduces to a 3×3 matrix as follows:

$$s\overrightarrow{x} = P\overrightarrow{X} \tag{3.3}$$

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \tag{3.4}$$

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{14} \\ P_{21} & P_{22} & P_{24} \\ P_{31} & P_{32} & P_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.5}$$

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.6}$$

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.7}$$

The map between a world plane and a perspective image is an example of a Homography, and the matrix H is a planar homography transform (defined up to a scale factor, equation 3.7).

The calculation of the camera parameters requires the estimation of the homography transform H.

Homography estimation

Various approaches exist for computing the homography between a plane in the world and an individual camera [29, 48]. Typically more than 4 points are used in a least-squares approach. In this work, the normalized Direct Linear Transform (DLT) algorithm discussed in Section 2.1 of [48] is employed to estimate H given a set of point correspondences (correspondences between 2D points in the image and 3D points in world). From equation 3.7, we have:

$$\begin{cases} h_1X + h_2Y + h_3 - (h_7X + h_8Y + h_9)x = 0 \\ h_4X + h_5Y + h_6 - (h_7X + h_8Y + h_9)y = 0 \end{cases} \tag{3.8}$$

Equation 3.8 can be rewritten as:

$$\begin{bmatrix} X & Y & 1 & 0 & 0 & 0 & -Xx & -Yx & -x \\ 0 & 0 & 0 & X & Y & 1 & -Xy & -Yy & -y \end{bmatrix} h = 0 \tag{3.9}$$

where $h = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{bmatrix}^T$.

According to equation 3.9, each point correspondence provides two linearly independent equations. Because the homography transform is written using homogeneous coordinates, the homography H is defined using 8 unknown parameters. Therefore, at least 4 point correspondences providing 8 equations are required to compute the homography. Equation 3.9 can be written for n pairs of point correspondences as follows:

$$\begin{bmatrix} X_1 & Y_1 & 1 & 0 & 0 & 0 & -X_1x_1 & -Y_1x_1 & -x_1 \\ 0 & 0 & 0 & X_1 & Y_1 & 1 & -X_1y_1 & -Y_1y_1 & -y_1 \\ X_2 & Y_2 & 1 & 0 & 0 & 0 & -X_2x_2 & -Y_2x_2 & -x_2 \\ 0 & 0 & 0 & X_2 & Y_2 & 1 & -X_2y_2 & -Y_2y_2 & -y_2 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ X_n & Y_n & 1 & 0 & 0 & 0 & -X_nx_n & -Y_nx_n & -x_n \\ 0 & 0 & 0 & X_n & Y_n & 1 & -X_ny_n & -Y_ny_n & -y_n \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ 1 \end{bmatrix} = 0_9 \tag{3.10}$$

which has the form $Ah = 0$ where A is a $2n \times 9$ matrix. Solving this linear system involves the calculation of a Singular Value Decomposition (SVD). The coordinates of point correspondences should be normalized to avoid numerical instabilities. This is completely discussed in Section 2.1.1 of [48].

To calibrate each of the eight cameras that make up the omnidirectional sensor, first at least four pairs point correspondences must be established in each camera's view. The robust extraction of point correspondences can be performed by exploiting the topological structure of a checkerboard-like pattern. The image points are extracted using the Harris corner detector [63] library of OpenCV with sub-pixel accuracy. Figure 3.16 shows the procedure of image point detection and selection. An accurate large grid area are prepared as well as a 2×2 ($25.5\,cm \times 19\,cm$) chessboard-like pattern that is positioned at the same level as the surface of the water (mother robot frame) and perpendicular to the mast for the omnidirectional camera as shown in Figure 3.16. In Figure 3.16(c), the features detected by sub-pixel accuracy are shown. The four image correspondence points are selected and marked as shown in Figure 3.16(d) by blue circles. The corresponding points in 3D world are measured in a grid environment attached to this frame.

To compute the homography transform, OpenCV libraries [19] were used that takes a set of image-world correspondences and returns a homography matrix H. A unique homography matrix can be computed from four points provided that no three of them are collinear. Each of the eight cameras are calibrated in this common framework attached to the mother robot.

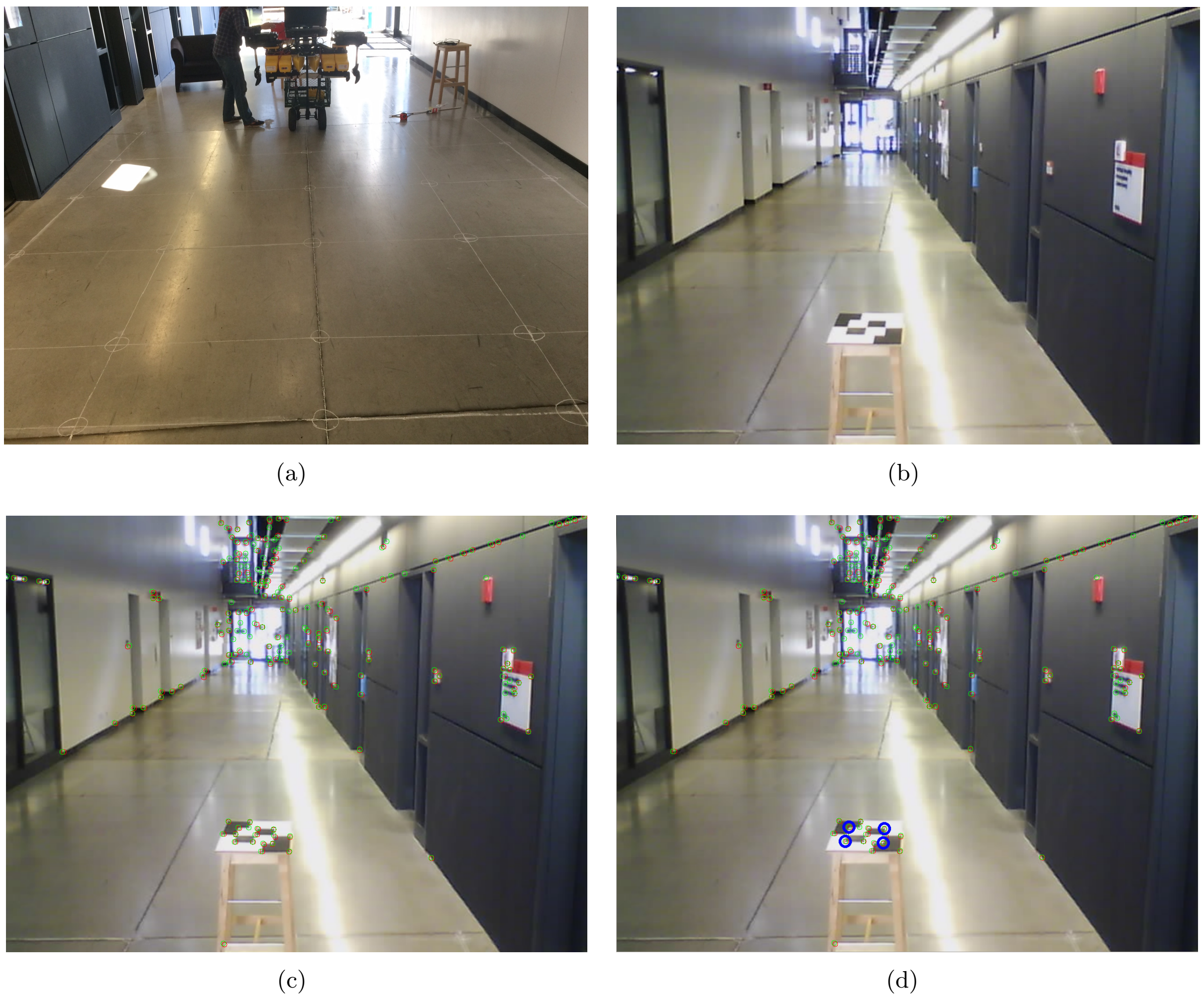


FIGURE 3.16: (a) A large grid area that is prepared by hand. (b) A 2×2 chessboard-like pattern used for calibration. Feature detection for calibration. (c) extracted corners are depicted. The red circle shows the corners detected using Harris corner detector and the green circles are the refined corners using sub-pixel accuracy. (d) four image points are selected from detected corners. The target is positioned so as to lie in the plane of the water when the robot is deployed on the water's surface.

3.3.3 Localization

The position and orientation of the mother robot are estimated using DGPS and a tilt-compensated compass respectively. The relative position and absolute orientation of each of the child robots are determined using the omnidirectional sensor mounted on the mother robot and the tilt-compensated compasses mounted on each of the child robots. Each of the child robots are equipped with a unique colour target. This target, which is large but of light weight, is detected and tracked using the colour blob detection library of OpenCV [18]. Figure 3.17 illustrates the colour detection and localization algorithm. As shown in Figure 3.17, each camera image is processed separately for both colour blob detection and image coordinates extraction.

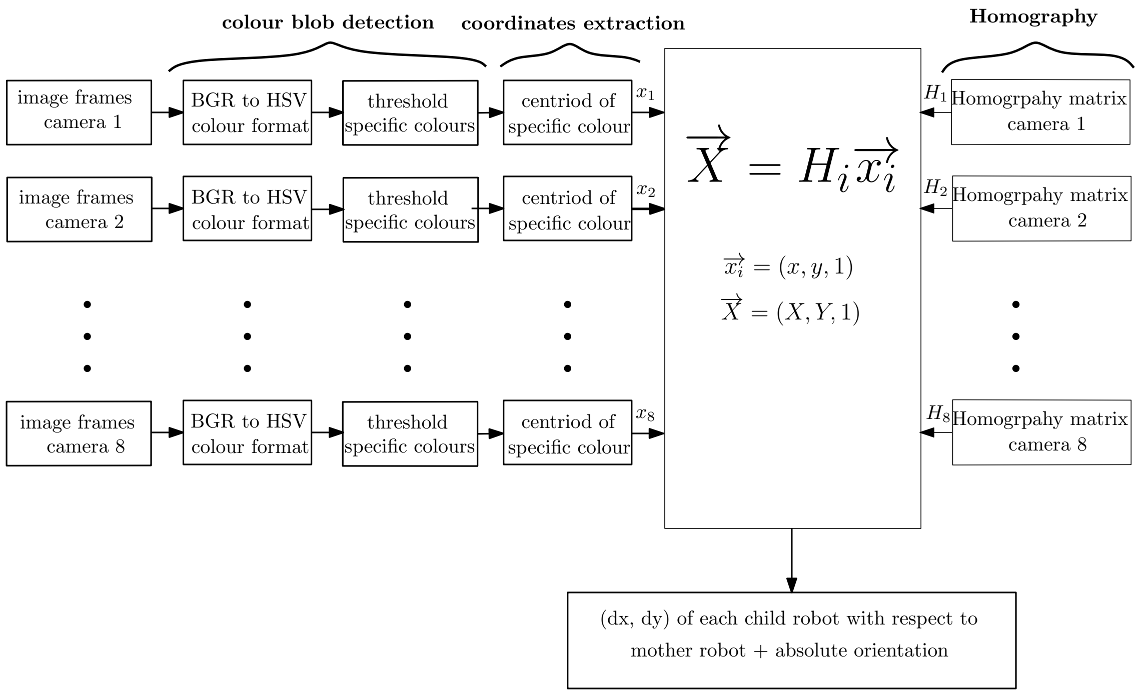

colour blob detection
coordinates extraction
Homography
image frames camera 1
BGR to HSV colour format
threshold specific colours
centriod of specific colour
x_1
image frames camera 2
BGR to HSV colour format
threshold specific colours
centriod of specific colour
x_2
image frames camera 8
BGR to HSV colour format
threshold specific colours
centriod of specific colour
x_8
$\overrightarrow{X} = H_i \overrightarrow{x_i}$
$\overrightarrow{x_i} = (x, y, 1)$
$\overrightarrow{X} = (X, Y, 1)$
H_1
Homogrpahy matrix camera 1
H_2
Homogrpahy matrix camera 2
H_8
Homogrpahy matrix camera 8
(dx, dy) of each child robot with respect to mother robot + absolute orientation

FIGURE 3.17: Child robot localization using the omnidirectional sensor.

Then, using the homography matrix of each camera, the position of each child robot with respect to the mother robot's frame is computed.

Localization accuracy

Accuracy is the uncertainty in the precise location of an object in an environment. The accuracy of the localization system used to localize the child robots relative to the mother robot is subject to errors associated with:

- **Camera precision.** The relation between the precision of the position estimation and the precision of the input image measurements is known as the Geometric Dilution Of Precision (GDOP) [49]. This is defined as:

$$GDOP = \frac{\Delta(\text{output measurement})}{\Delta(\text{input parameters})} \tag{3.11}$$

  GDOP can be expressed using the Jacobian of the measurement equation. The GDOP in our work is computed as follows:

$$s\begin{bmatrix}x\\y\\1\end{bmatrix} = H\begin{bmatrix}X\\Y\\1\end{bmatrix} \tag{3.12}$$

$$\begin{bmatrix}X\\Y\\1\end{bmatrix} = sH^{-1}\begin{bmatrix}x\\y\\1\end{bmatrix} \tag{3.13}$$

$$\begin{bmatrix}X\\Y\\1\end{bmatrix} = \begin{bmatrix}a & b & c\\d & e & f\\g & h & i\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix} \tag{3.14}$$

$$\begin{cases} X = (ax+by+c)/(gx+hy+i)\\ Y = (dx+ey+f)/(gx+hy+i)\end{cases} \tag{3.15}$$

$$GDOP = |J| = \left|\begin{bmatrix}\frac{\partial X}{\partial x} & \frac{\partial X}{\partial y}\\ \frac{\partial Y}{\partial x} & \frac{\partial Y}{\partial y}\end{bmatrix}\right| = \left|\begin{bmatrix}\frac{a}{gx+hy+i} - \frac{g(ax+by+c)}{(gx+hy+i)^2} & \frac{b}{gx+hy+i} - \frac{h(ax+by+c)}{(gx+hy+i)^2}\\ \frac{d}{gx+hy+i} - \frac{g(dx+ey+f)}{(gx+hy+i)^2} & \frac{e}{gx+hy+i} - \frac{h(dx+ey+f)}{(gx+hy+i)^2}\end{bmatrix}\right| \tag{3.16}$$

The geometric dilution of precision associated with one of the cameras are plotted in Figure 3.18. The pixel resolution of each of the cameras that makes up the omnidirectional sensor is 640×480 pixels although there are some pixels in each individual camera image that do not provide information about the water surface (they are viewing the sky). For a typical camera in the omnidirectional sensor, pixel row (y) values from 0 to 240 correspond to the sky, so we ignore them in the following. The magnitude of J is plotted in Figure 3.18. J is plotted over the entire image width ($x = 0..639$) and for that portion of the image that corresponds to the water's surface ($y = 240..479$). $|J|$ grows corresponding to points with small y values – points near the horizon.

- **Calibration accuracy.** In this work, the calibration accuracy depends on the homography matrix of two planar surfaces (image plane and ground plane). OpenCV is used to calculate the homography matrix. To find a good homography:

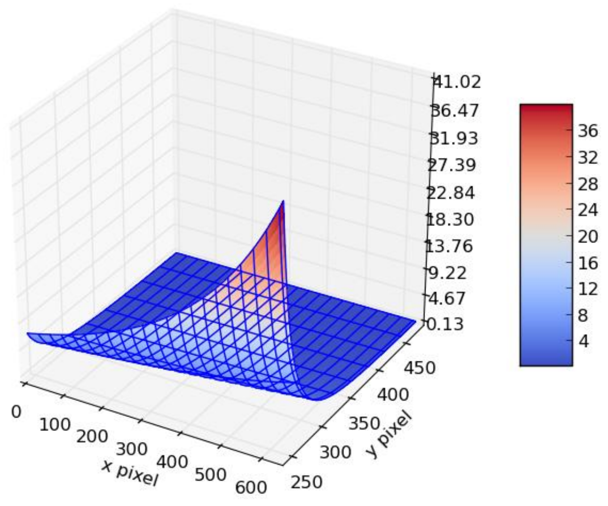


FIGURE 3.18: Geometric Dilution Of Precision (GDOP) of one of cameras. This figure shows the magnitude of J for a typical camera in the omnidirectional sensor. The magnitude of $|J|$ increases with lower y (row) values, corresponding to pixels closer to the horizon ($y = 240$ in this case).

  – Accurate matches of the key points are needed. At least four matches are required to compute the homography matrix. In this work, the four key points in image plane are extracted by corner detection with sub-pixel accuracy. In order to improve the accuracy, more matched key points could be used.
  – Key points distributed over the entire image are needed. If the four points are at the corners of the image, then a good homography matrix will be computed.

The best way to evaluate the accuracy of the homography is to evaluate reprojection errors [65]. The reprojection error is a geometric error corresponding to the distance between a pattern key point detected in a calibration image, and the corresponding world point projected into the same image.

- **Ground plane assumption errors.** In this work we assume that the water surface is flat even though the boat is subjected to wave action. The mast projects some height from the surface of the water and can thus sway with wave action potentially violating this assumption. These values can be computed using onboard compass to resolve ground plane assumption errors.
- **Blob detection accuracy.** Each child robot is augmented with a 33(length)×17(width)×25(height) cm colour target (see Figure 3.20). This target, which is a large but of light

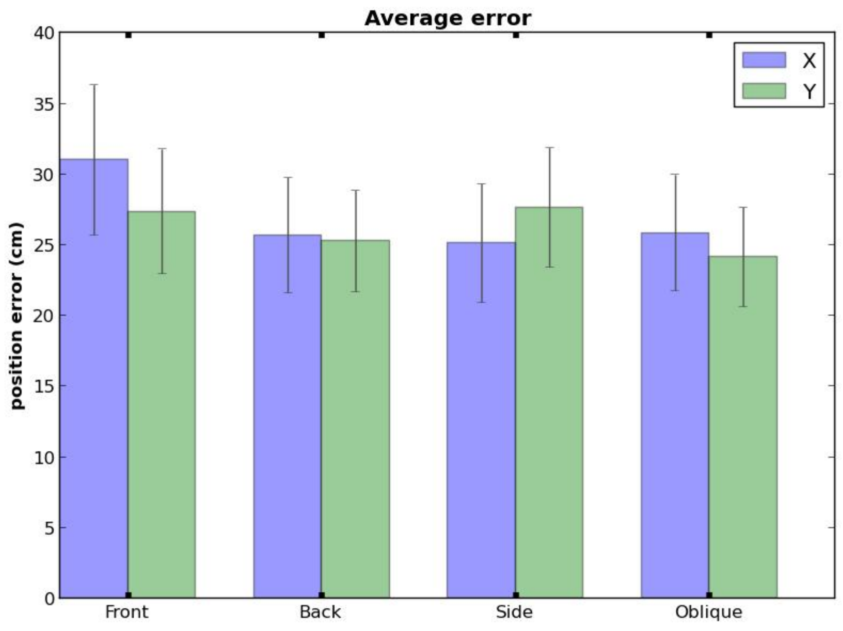


FIGURE 3.19: Average blob detection error in both X and Y direction at distance range 300 cm to 500 cm. X is the fore-aft direction of the mother robot while Y is the port-starboard direction. The ground truth measurements are done by hand (a large grid area). Error bars show standard errors. Views are of the child robot from the Front, Back, Side or Oblique.

weight, is detected and tracked to estimate the position of the child robots. Since this target should be detected in a larger distance from camera, it is designed to be large. The Blob detection accuracy is subject to:

– Where on the boat the blob is identified. Note that the boat position is not considered as the centre of detected blob, rather it is considered as the mid bottom point of the blob. Given the geometry of the boat and target, the worst case occurs when the boat is detected from the front or rear. In this case, the position error could be as large as one half of the length of the colour target. In order to evaluate this the blob detection algorithm was run on the boat in a number of different boat orientations relative to the camera at a range of distances from the sensor. Figure 3.19 depicts the position errors for different boat orientations in both the X and Y direction. Here X is the fore-aft direction of the mother robot while Y is the port-starboard direction. The Back and Side errors are slightly more than 1/2(length) and 1/2(width) respectively. Figure 3.20 shows the position of the blob with respect to the mother robot for various boat orientations.

– There is an error due to the bottom of the blob that is not at the surface of the water. Furthermore, the bottom of the front of the target is not at the same level on

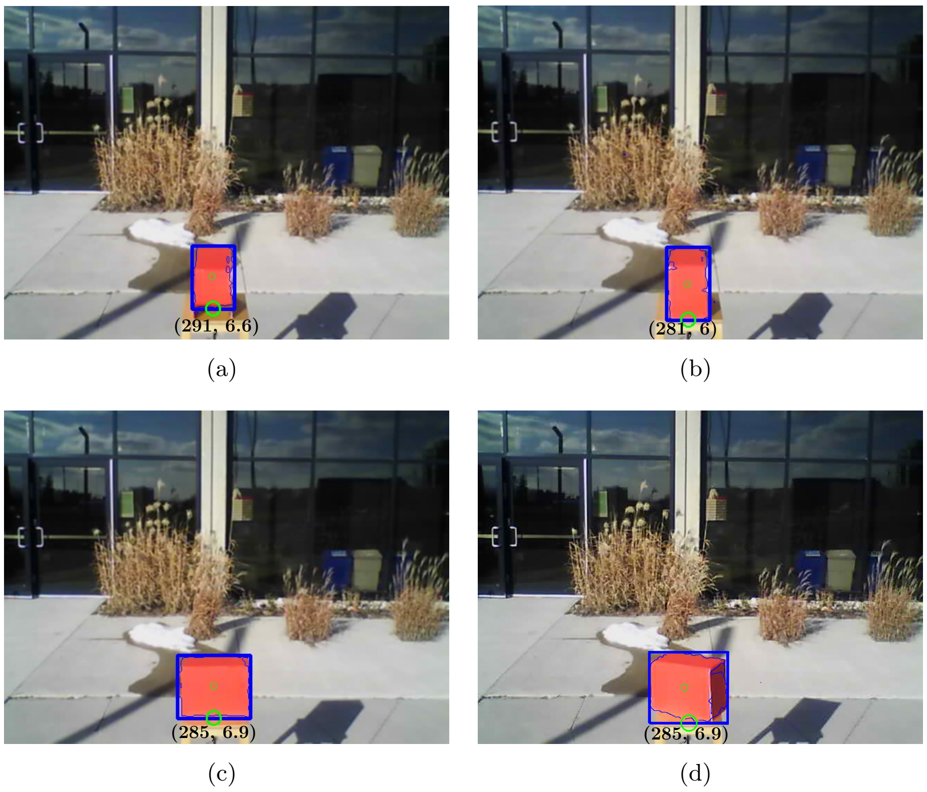
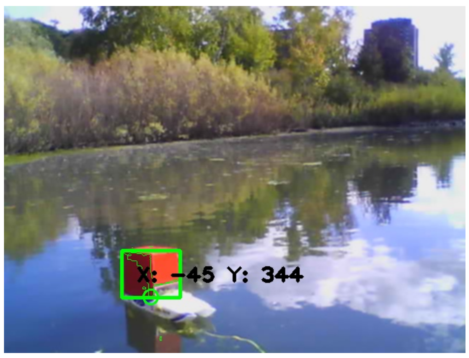


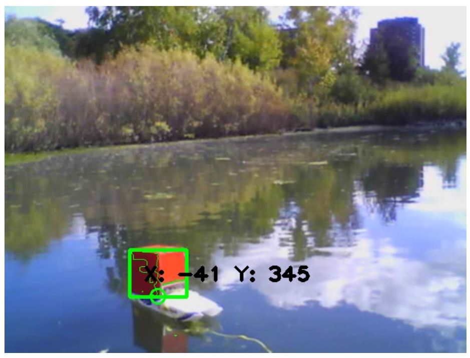
FIGURE 3.20: Blob detection at X = 300 cm and Y = 0.0 cm in the mother robot frame. The large green circle shows the image pixel of detected child robot. (a) Front side. (b) Back side. (c) Either right or left side. (d) Oblique view.

the boat as are the bottom of the back and side of the target. Figure 3.21 depicts a sequence of frames showing the detected colour box in real environment.

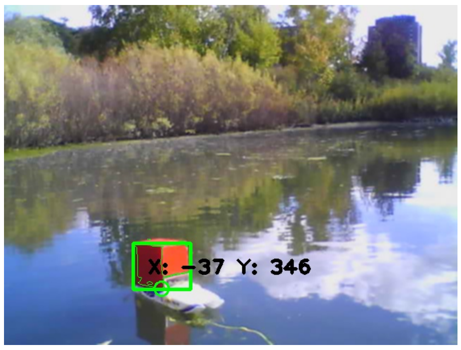
Each of the individual sources of error contribute to the overall system performance. The position of the robot measured by the localization system and with the real position of the robot were compared to show the accuracy of localization system used in this work (Figure 3.22). In this figure, the errors are caused by the combination of all errors explained above. The robot ground truth, the robot position estimation in various child robot orientations, and the maximum error in X and Y direction are depicted. Although one might have thought that there might have been a systematic error in the recovered position of the child robot that always placed it farther away from the mother robot – due to the geometry associated with identifying the blob location above the water's surface – this is not the entire story. Errors were typically in the range of 10-25cm, with a maximum error of 90cm. Position estimations were made at approximately 6hz.



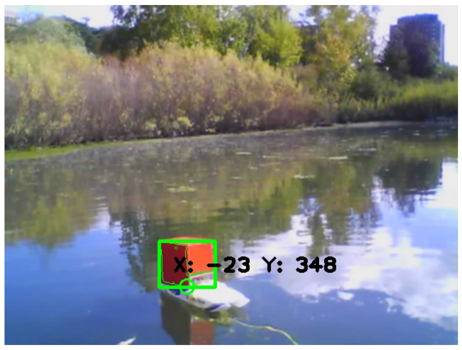
(a)



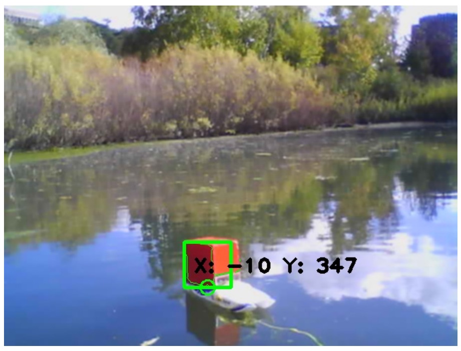
(b)



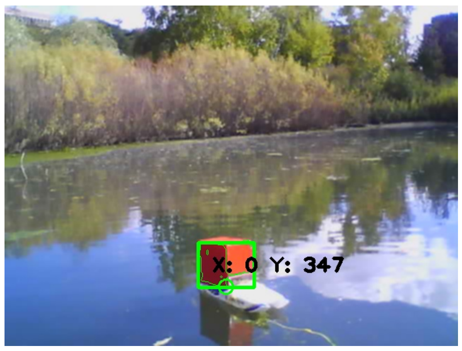
(c)



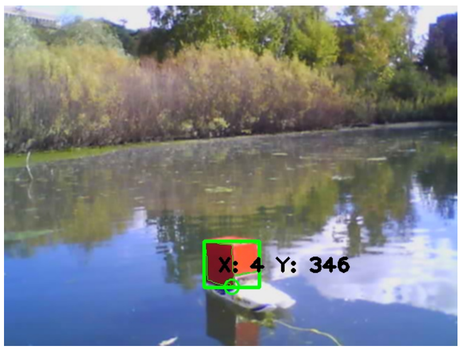
(d)



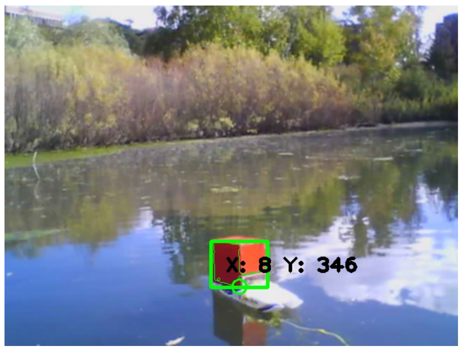
(e)



(f)



(g)



(h)

FIGURE 3.21: A sequence of frames captured by camera 3 that detects the child robot in real environment. The large green circle shows the image pixel of detected child robot. The position of the child robot with respect to the mother robot frame is shown as well.

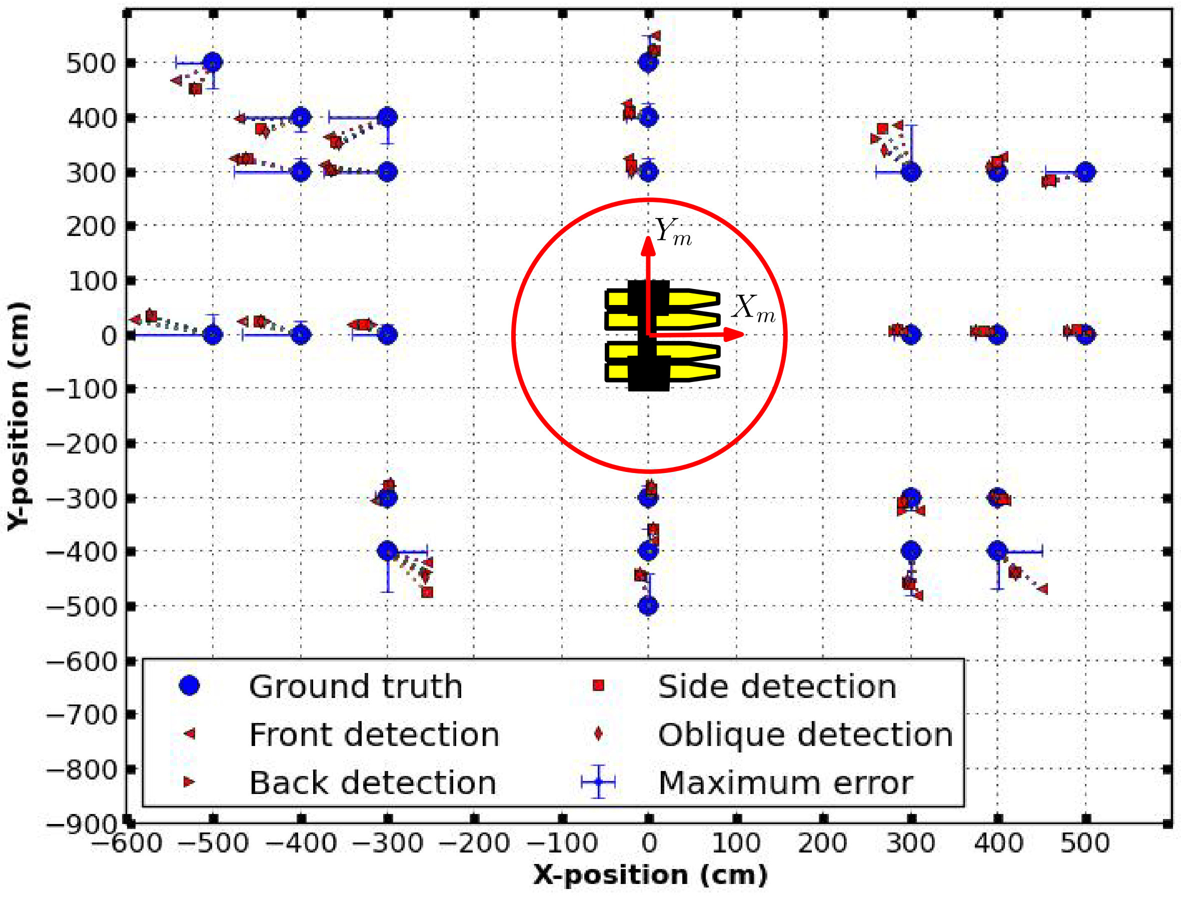


Figure 3.22: Child robot position in various blob identification at different positions. The robot ground truth, the robot position estimation in various side, and the maximum error in x and y direction are depicted. The ground truth measurements were done by hand.

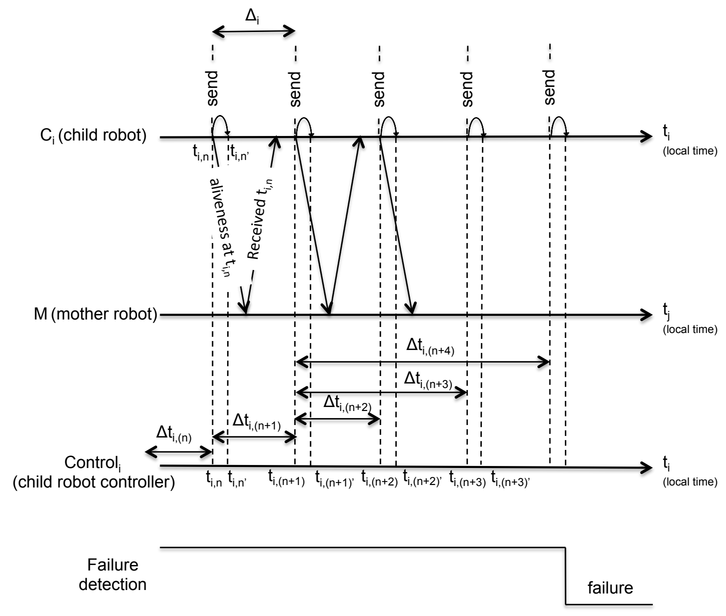
3.4 Intra-fleet communication

The fleet developed in this work requires wireless communication between the robots. Although we assume good communication between elements of the fleet, any communication strategy must deal with the reality that communication might fail. The ROS multi-master approach allows for the independent running of roscore masters on multiple entities, which allows for reduced network traffic, since the robots and computers exchange data only on selected topics and services. In this work, *foreign_relay* nodes are employed to transfer topics between the masters. When masters become disconnected, foreign topics die, but the individual robots remain functional and can take remedial action. For this to occur, however, it is essential that the various elements of the fleet are aware of the liveliness of the communication infrastructure.

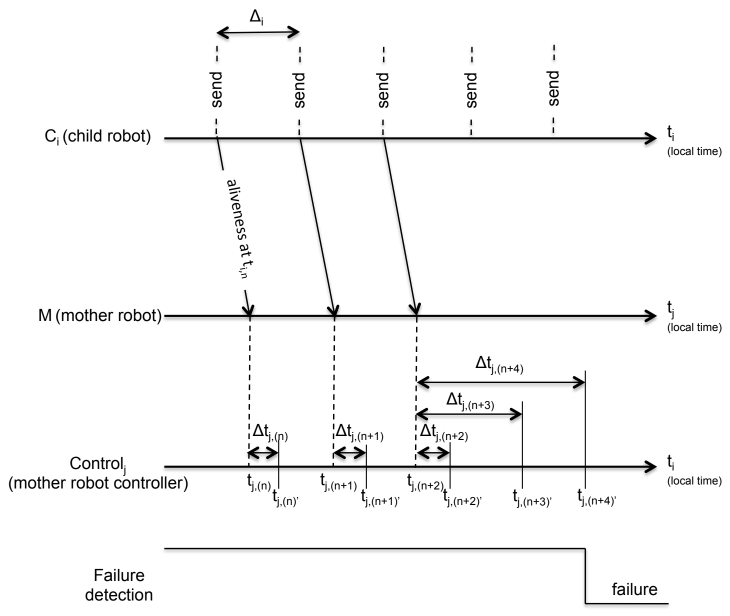
During early trials of minnow the need for a heartbeat (still alive) signal was made evident. The initial prototype would maintain its last command until completion, until the robot received another command or until it ran out of power. This command structure would become problematic if the boat for any reason lost contact with the base station due to driving too far away or due to a wave blocking the signal. Modifying the ROS command structure to require a heartbeat in the command structure as described in Chapter 2 allows for each robot to enable some default action (i.e. setting the throttle to zero) when communication is lost. Giving every robot its own master (multi-master roscore), allows individual robots to retain complete independence (individual robots can initialize new ROS nodes, transmit ROS messages, etc. even when disconnected from other robots in the fleet). Additionally, robots can change tasks, retrieve data and take corrective action (e.g., plan a path back to wireless range or backup action) when they detect that they are no longer in communication with the fleet.

The robots communicate in a star topology[1] with the mother robot as the central node. A standard heartbeat strategy is used to maintain the status of the communication structure (see [33]) between the mother robot and each child robot. In the heartbeat strategy, each child robot periodically (at 0.5 hz) sends a heartbeat message to the mother robot in order to inform the mother robot about its aliveness at a given time. Agents begin operation with a synchronized clock. When the mother robot receives this message, a message is sent back to the child robot. Message traffic between a child robot and a mother robot control (C_i and M respectively) is shown in Figure 3.23. Figure 3.23(a) illustrates the child robot's strategy to detect communication failure with the mother robot M. In this figure, the heartbeat period of the child robot, is Δ_i, the time between two heartbeat messages sent by the child robot. The aliveness time, $t_{i,n}$, is the time at which the heartbeat message $n$from the child robot i is published. The heartbeat message sent by a child robot consists of its aliveness time $t_{i,n}$. When the mother robot receives this message it echos it to the child robot. The delay time, $\Delta t_{i,n}$, is the difference between the last heartbeat message published and the last message received from mother (M). Each child robot concludes that its connection is lost with the mother robot when $\Delta t_{i,n} > (\Delta_i + T_{tolerance})$ where $T_{tolerance}$ is the tolerance time. The child robot detects that communication has been re-established when it once again receives an echoed heartbeat message with a valid communication delay time.

[1]Every robot connects to a central robot; the central robot acting as a server and the peripheral robots as clients.



(a)



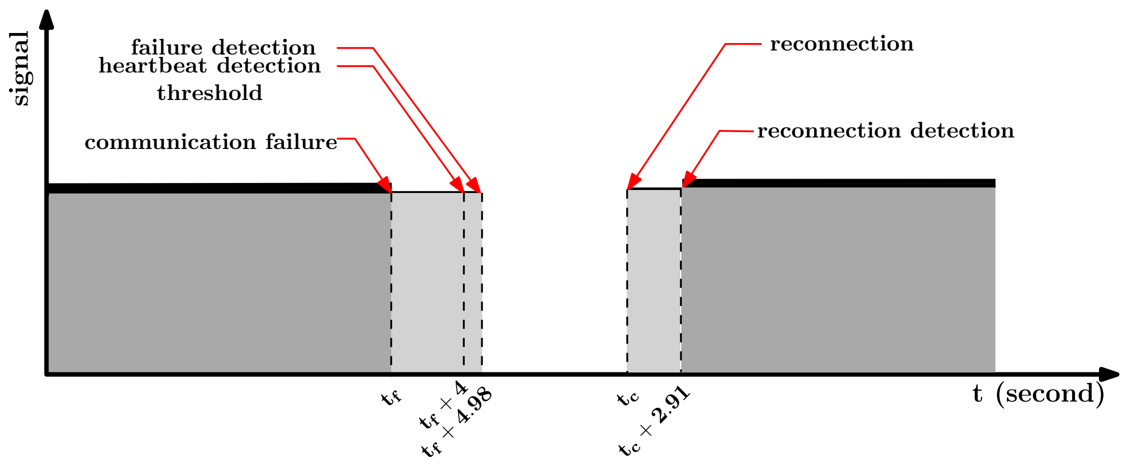
(b)

Figure 3.23: The strategies to detect communication failure. (a) The child robot's strategy to detect communication failure. (b) The mother robot's strategy to detect child robot failure. On a regular basis each child sends a message to the mother, which is then returned to the child. If the child does not receive the message within a predefined time interval, the child determines that it has lost communication with the fleet. Similarly, the mother robot expects to see a heartbeat message from the child every Δ_i seconds. If the message is not received in time $\Delta_i + T_{tolerance}$ then the mother declares that it has lost contact with the child. The mother claims that it has reconnected with the child once a heartbeat message is received again.

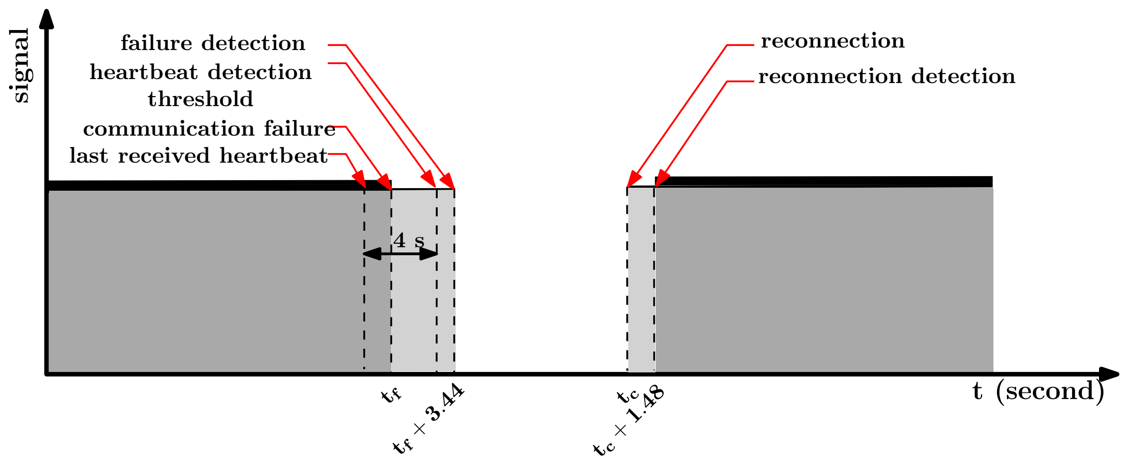
Figure 3.23(b) illustrates the mother robot's strategy to detect communication failure with a child robot. The received time, $t_{j,n}$, is the time at which the nth heartbeat message is received by the mother robot M. The mother robot should receive heartbeat messages every Δ_i seconds. Whenever the time between heartbeats, $\Delta t_{j,n}$, becomes greater than $\Delta_i + T_{tolerance}$, the mother robot declares that communication has been lost with child i. Similarly, whenever the time becomes less than $\Delta_i + T_{tolerance}$ the mother robot determines that it has reconnected with child i. This basic pinging strategy is augmented slightly to deal with multiple child robots. The echo ping from the mother to a given child robot also includes the current liveness status of each of the child robots. In this way each live child robot "knows" the liveliness of each of the other members of the fleet and is updated as to its true position from the mother robot.

Using the liveliness protocol described above, each of the elements of the fleet – here the mother and one child robot – monitor the state of the communication channel. Communication disconnect/reconnect events were repeated 15 times and the average time of detection of failure and detection of reconnection by the mother and child robot were recorded. Figure 3.24(a) illustrates the communication failure detection on a child robot. Here, the child robot sends a heartbeat message every 2s to inform the mother robot about its aliveness. At time t_f the communication channel was severed. The child robot detects its communication failure when the delay time is more than (2s + $T_{tolerance}$) (In this experiment $T_{tolerance}$ was set to 2s). Thus, the time period between the robot failure and the detection of this failure is defined by the heartbeat threshold (here set to four seconds). In this way, the full delay in the process due to OS overhead, ROS overhead, and the like could be properly examined. Approximately one second after this the 4s period the child robot detected this communication failure. At time t_c the communication channel was restarted. The child robot detects the reconnection roughly three seconds after reconnection. The dark grey region indicates that portion of time during which communication was possible and light grey indicated that portion of time which the communication failure and reconnection are detected by a child robot.

Figure 3.24(b) depicts the state of a child robot liveliness detected by the mother robot. The mother robot receives heartbeat messages sent by the child robot every 2s and checks the delay time (the time between current time and last received heartbeat message) periodically (every 1s). At time t_f the communication failure occurred. Then, the failure is detected when this delay time is more than (2s + $T_{tolerance}$). In this experiment $T_{tolerance}$ was set to 2s. Approximately 3.44 seconds after the communication failure, the mother robot detected communication failure. At this time, the mother robot adds the child robot to the list of failed child robots. At time



(a)



(b)

FIGURE 3.24: The status of the child robot aliveness detected by itself (Minnow) and the mother robot (Eddy). The dark grey region indicates that portion of time during which communication was possible and light grey indicated that portion of time which the failure and reconnection are detected. Here communication between the child robot and the mother was cut off at time t_f. The heartbeat period was set to 2 seconds, and $T_{tolerance}$ was considered as 2 seconds. (a) Failure and reconnection detection by the child robot. (b) Failure and reconnection detection by the mother robot. In the experimental runs averaged here, the child robot detected failure after 4.98s and the mother robot detected failure after 3.44s. At some time later t_c communication was re-established. Here the child robot detected that communication had been re-established after 2.91s while the mother robot detected this after 1.48s. Data was averaged over 15 trials. These experiments involved the real communication infrastructure, so times include time for the WIFI signal to be properly re-acquired by the OS and signals to be sent over the re-established link.

t_c the communication channel is re-stablished. The mother robot detects the reconnection of the child robot roughly 1.5 seconds after the reconnection. At this time, the mother robot removes the child robot from the failed list. The dark grey region indicates that portion of time during which communication was possible and light grey indicated that portion of time which the failure and reconnection are detected by the mother robot.

3.5 Summary

This chapter described the algorithms developed in this thesis for addressing the problems of area coverage, localization and communication for a team of heterogeneous aquatic surface robots. In this work, the world is modelled as a plane with an exterior and interior polygon boundary. First, a simplified version of solution to the area coverage problem is considered and then this solution is extended to satisfy the realities of the heterogeneous fleet. The Boustrophedon decomposition algorithm and a Reeb graph are employed to divide the known environment into cells. Each cell covered with a simple back-and-forth motion following an Eulerian circuit through all cells. A line abreast formation for the fleet members is utilized such that their unified sensor coverage meets the requirements of the Boustrophedon algorithm. Finally, a full turn strategy is developed for the child robots so that their turning motions meet the Minnow and Eddy non-holonomic motion constraints while still satisfying the requirements of the Boustrophedon decomposition algorithm.

For localization, a cooperative pose estimation approach is employed. Since the mother robot is able to robustly localize itself within the environment, the child robots rely on the mother robot for localization. An omnidirectional camera is mounted on the mother robot to track the movement of the child robots using coloured targets mounted on each of the child robots. The position of each child robot is determined relative to the mother robot and the orientation of them are determined utilizing on-board compass.

Communication and synchronization are key requirements in a heterogeneous team of robots. In terms of ensuring reliable communication, a heartbeat strategy is utilized in which robots communicate not only their state but also liveliness information in order to remain aware of the current state of the other robots in the fleet, and the current state of intra-fleet communication.

Chapter 4

Experimental validation

4.1 Introduction

This chapter provides a validation of the algorithms proposed in Chapter 3 through both simulation and operation of robots in the real world. The algorithms developed in this work are implemented in python and C++ under a Linux environments and ROS middleware. To visualize the results, *Rviz (ROS Visualization)* tools and Google Earth images are used.

The robots employed in this work are described in Appendices A and B. Here, we describe the structure of the fleet, its communication setup, as well as the localization process of the Minnows and Eddy in both simulation and field trials. The localization process for each robot is explained and the results are provided. The validation of point-to-point navigation of two kinds of robots is provided through simulation and validation of our sensor-based area coverage algorithm are provided through simulation as well.

4.2 Structure of the fleet

The fleet used in the testing described here consists of two Minnow robots and one Eddy. A single wireless network is used to communicate among the elements of the fleet and with a shore-based base station which provides both operator control and also communication with a fixed base station for DGPS localization of Eddy. Figure 4.1 provides an overview of the communication structure of the fleet in this work. The robots communicate in a star topology with Eddy as

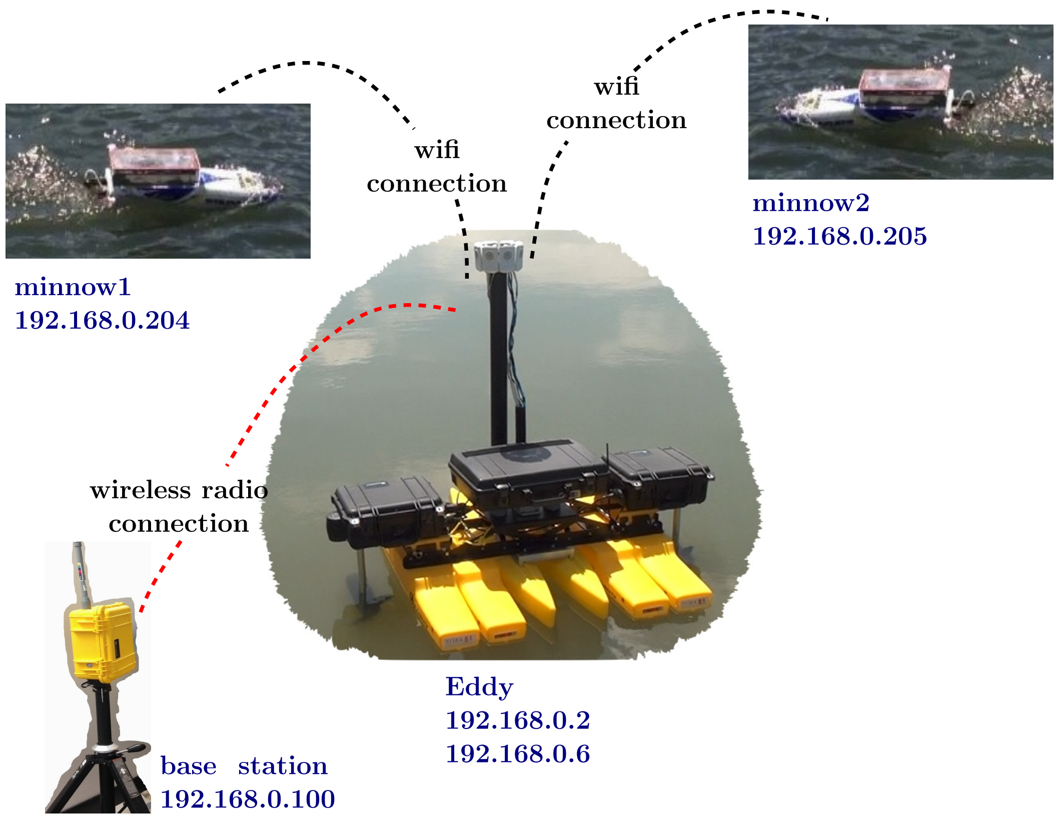

wifi connection
wifi connection
minnow1 192.168.0.204
minnow2 192.168.0.205
wireless radio connection
Eddy 192.168.0.2 192.168.0.6
base station 192.168.0.100

FIGURE 4.1: Structure of the fleet. The fleet used in the testing described here consists of two Minnow robots and Eddy.

the central node. A WIFI base station on board Eddy provides communication within the fleet. Eddy is equipped with a wireless radio transmitter that provides reliable communication with a shore-based operator. The WIFI system on Eddy and the radio communication channel to offshore users is addressable as a single TCP/IP network to all of the operating agents. Furthermore, the network allows for the establishment and maintenance of a common clock among all of the agents. Figure 4.1 also shows the IP addresses defined for each robot in the fleet.

4.3 Communication

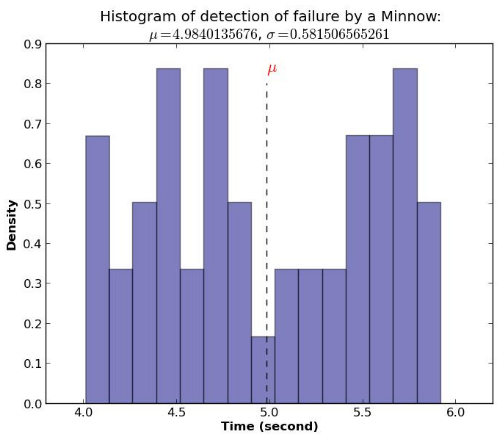
Communication between the element of the fleet is key in order for them to operate as an intelligent collection of robots. In addition, the DGPS technology that Eddy relies on for localization requires communication. Communication is also essential for the human operator to be able to control the operation of the fleet.

Each robot runs its own roscore. This enables the individual robots to operate independently of the other members of the fleet. On each robot a *foreign_relay* node is used to share published topics across multiple ROS masters (individual robots). Topics published from remote roscores are registered with the local ROS master with the same name. Heartbeat messages from the Minnow robots are exchanged periodically with Eddy to maintain the status of their connections with Eddy and to ensure that pose estimates are refreshed across the network. When communication failure is detected the fleet ceases movement and awaits operator input. Several pool and pond trials were performed to validate the multi-master and the heartbeat strategy functionality. In the current implementation a Minnow robot detects that its connection is lost with Eddy, it sets its speed to zero.

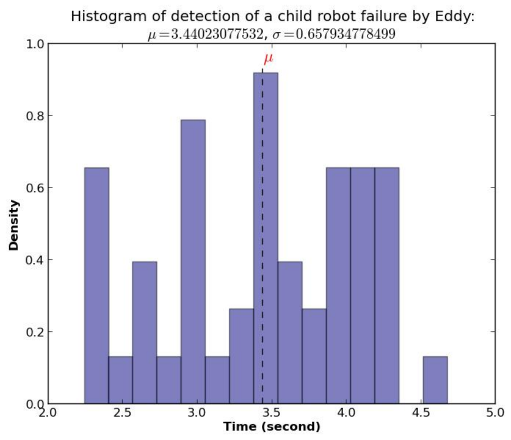
Beyond the detection of communication failure there is also the issue of communication reacquisition. Figure 4.2 shows a histogram of detection delay of 50 communication failure and reconnection data by a child robot and the mother robot.The actual distribution of time to detect communication failure is far from a normal distribution as should be expected given the discrete nature of the various underlying processes. At time t_f the communication link was severed. The heartbeat period was set at 2 seconds. Figure 4.2(a) shows the histogram of time to failure detection by a Minnow robot. Here, the heartbeat communication failure timeout was set to 4 seconds. Mean time for the minnow to detect failure was 4.984 seconds, with a maximum detection time of 5.721 seconds. Figure 4.2(b) shows the histogram of failure detection delay by Eddy. Here, the heartbeat communication failure threshold was set to 4 seconds from the last received heartbeat message time. The mean time for Eddy to detect communication failure was 3.44 seconds after communication failure, with a maximum detection time of 4.677 seconds.

At time t_c the communication link is reconnected. The heartbeat period was set at 2 seconds. Figure 4.2(c) shows the histogram of reconnection detection by the Minnow. Mean time for the Minnow to detect reconnection was 2.911 seconds, with a maximum detection time of 3.9 seconds. Figure 4.2(d) shows the histogram of reconnection detection by Eddy. The mean time for a Minnow robot to detect reconnection was 1.485 seconds, with a maximum detection time of 2.918 seconds.

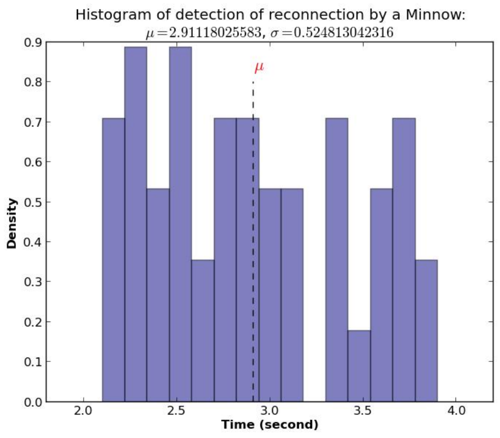
In any robot collective in which communication is essential, as is the case for the fleet here, the ability to detect and respond to communication failures is a key requirement. The multi-roscore/foreign_relay approach proved to be a highly successful and predictable mechanism for



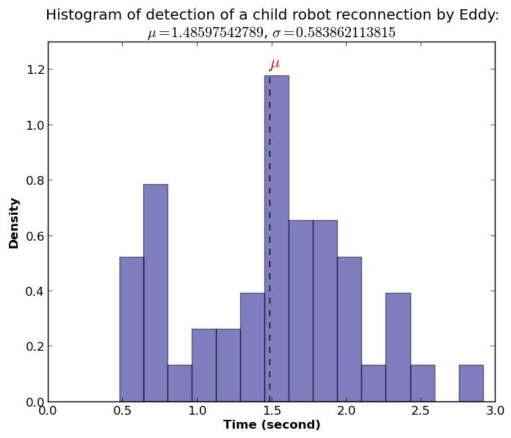
(a)



(b)



(c)



(d)

FIGURE 4.2: The graphs above show the distribution of 50 communication failure and reconnection detections by a Minnow robot and Eddy. The heartbeat period was set at 2 seconds. (a) Histogram of failure detection by a Minnow robot. Heartbeat communication failure threshold was set to 4 seconds. (b) Histogram of communication failure detection by Eddy. Heartbeat communication failure threshold was set to 4 seconds after the last received heartbeat message. (c) Histogram of reconnection detection by a Minnow robot. (d) Histogram of communication reconnection detection by Eddy.

detection communication failure/reconnection between Eddy and a Minnow robot. Communication failure detection was without exception detected within the small number of seconds threshold used here. For extremely short time thresholds, this is unlikely to be true but for thresholds in the small numbers of seconds internal timeouts related to the underlying OS and communication structures are not relevant.

4.4 Localization

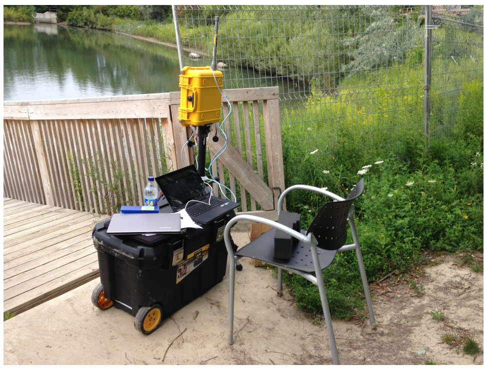
4.4.1 Eddy localization

Localization of Eddy is accomplished via DGPS and an on-board compass. Determining the absolute accuracy of the DGPS system is not possible as we lack a sufficiently accurate ground truth model. That being said, it was possible to characterize the precision of the DGPS sensor.

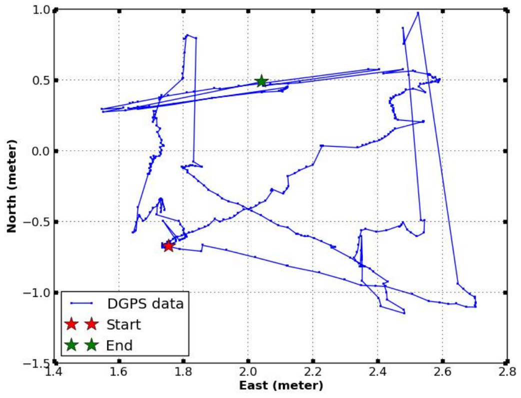
In order for the DGPS system to obtain good correction data, it requires a clear view of the sky. This is dependent upon the weather and the local geography. Many of the tests conducted during this work were performed on Stong pond on the campus of York University. Stong pond is surrounded by trees, and therefore the satellite coverage is not necessarily as good as at might be the case in wider open areas. The real-time DGPS used in this work requires that the base station be stationary and have access to satellite signals for 30-40 minutes prior to obtaining good GPS correction terms. Although this delay was recommended by the manufacturer, the number of trees obstructing the clear view of the sky has affect on the time needed for the DGPS correction term (see [59]). The base station was typically left for 50-60 minutes prior to testing in order for the solution to converge to a coherent value. The base station setup near Stong pond is shown in Figure 4.3(a).

In order to determine the precision of the DGPS sensor, Eddy's GPS sensor was set up as a stationary antenna over a period. Figure 4.3(b) plots the reported position of Eddy with respect to the base station position over the period while the robot remained stationary on land. As can be seen from the figure, the robot's reported GPS position varies in both x and y. These results are taken under particularly poor weather conditions. Figure 4.3(c) provides a histogram of deviation from the origin as well as the distribution curve over the same period. Mean deviation from the origin was one meter, with a maximum deviation of 2 meters. Since normally Eddy is moving, the precision when it moving is also of interest. In order to estimate this, Eddy was moved back and forth along a known path. Figure 4.4 plots the reported position of Eddy with respect to the base station position moving back and forth along a known path of ten meters. Positional precision while under motion seems consistent with that obtained while the robot is stationary.

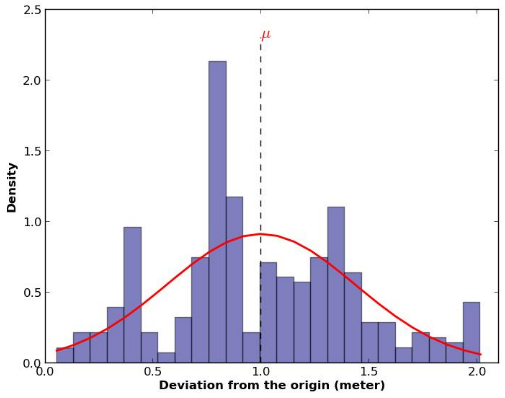
Several pond trials were performed to verify the DGPS system in practice. Figure 4.5(a) shows the latitude/longitude of Eddy's position as recorded by the DGPS system as it was driven



(a)



(b)



(c)

FIGURE 4.3: Eddy localized using DGPS. This requires a fixed base station (shown in (a)). Performance of the DGPS system can be quite variable. (b) plots the reported position of Eddy over about 10 minutes while the robot remained stationary. As can be seen from the figure, the robot's position can vary in both x and y. (c) provides a histogram of deviation from the origin over the same period.

around the safe boundary of Stong pond. Figure 4.5(b) overlays this data on Google Earth imagery of the pond. Figure 4.5(c) adds the on-board compass orientation data when the Eddy was driven around the boundary of the safe area. Figure 4.5(d) shows a section of the motion track to show the heading of Eddy. The direction of markers depicts the heading of Eddy. In practice, the DGPS signal available to Eddy can be expected to drift up to two meters although mean drift is closer to 1m.

4.4.2 Minnow localization

Localization of a Minnow is accomplished via omnidirectional sensor mounted on Eddy and an on-board compass. The position of the Minnows are determined with respect to Eddy frame.

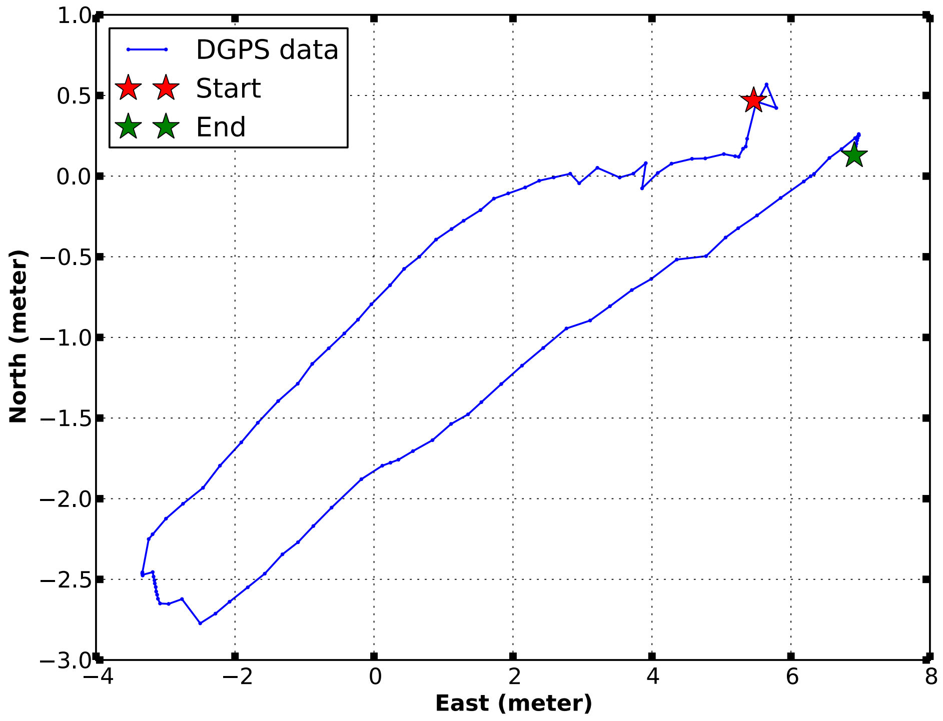


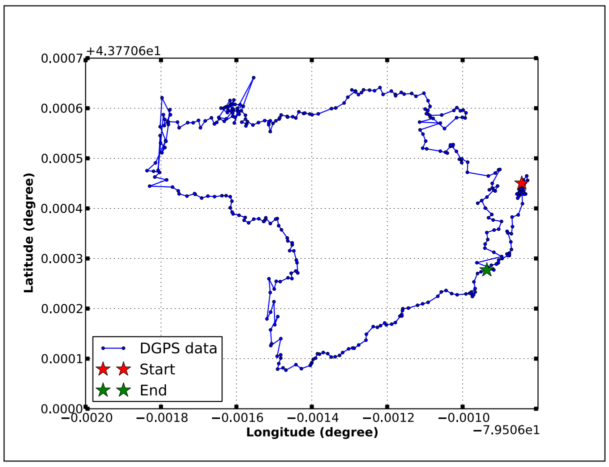
FIGURE 4.4: The positions of Eddy with respect to the base station position passing back and forth along a known path of ten meters.

First we did calibration for each camera separately to determine the transformation matrix between world frame and each camera frame.

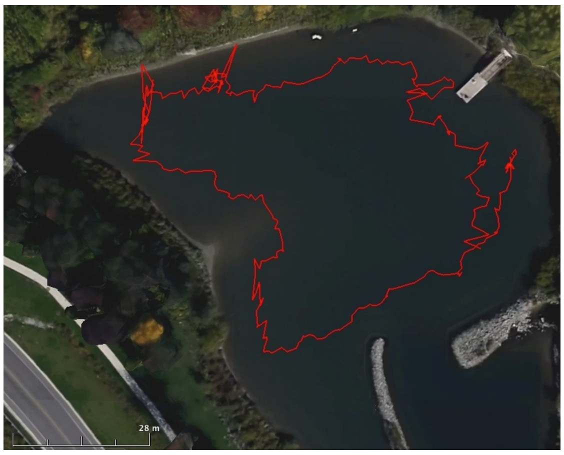
Several bench and pond tests were performed to evaluate the visual positioning system used to localize a Minnow relative to Eddy. Figure 4.6 shows the tracking data of a Minnow when it was moving in a straight line along different distances in different directions relative to Eddy. Dotted lines indicate ground truth measured by hand. Figure 4.6 also shows the Kalman tracker results of each trajectory (solid lines). Figure 4.7 plots the reported raw data and a Kalman tracker results when the Minnow is moved on a circular trajectory. The Kalman filter used was an EKF with good knowledge of the commanded motion of the vehicle (straight line at a known velocity or circular motion with known turning angle and velocity).

4.5 Point to point navigation

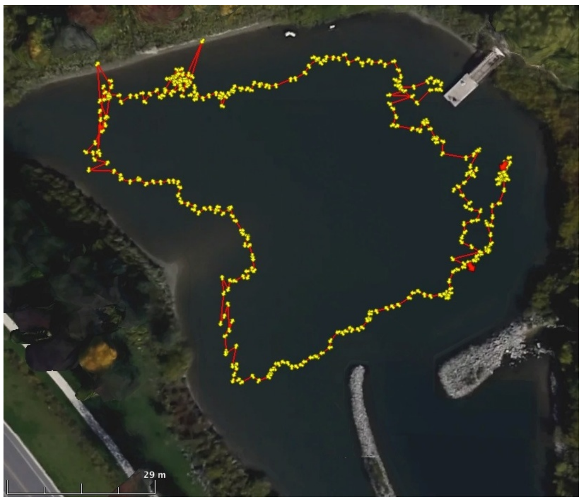
Given an appropriate mechanism to measure absolute pose data from elements of the fleet, it becomes possible to conduct point to point navigation of the various fleet elements. Given the



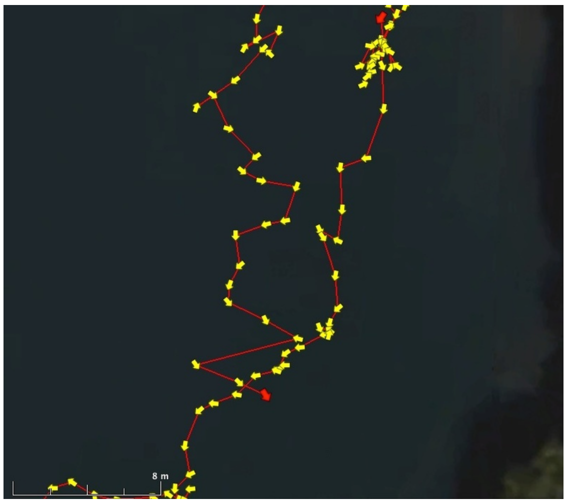
(a)



(b)



(c)



(d)

FIGURE 4.5: (a) Results of Eddy positioning as it is driven around Stong pond near York university. (b) A map of Eddy's track across Stong pond. There is some offset of the boundary in the google map imagery because of trees in northwest of Stong pond. (c) and (d) show a map of Eddy's track across Stong pond showing compass orientation information as well. Red markers indicate the start and end of the track.

difficulty in obtaining ground truth data with the real robots, results here are presented in simulation only.

Eddy

Simulation results of Eddy following a sequence of waypoints is depicted in Figure 4.8. This simulation was implemented using the 3D visualizing package *rviz* of ROS. Control of the Eddy robot is accomplished by providing the robot with a sequence of waypoints $wp = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$ given a starting state (x, y, θ) for the robot. Driving the robot through these waypoints involves

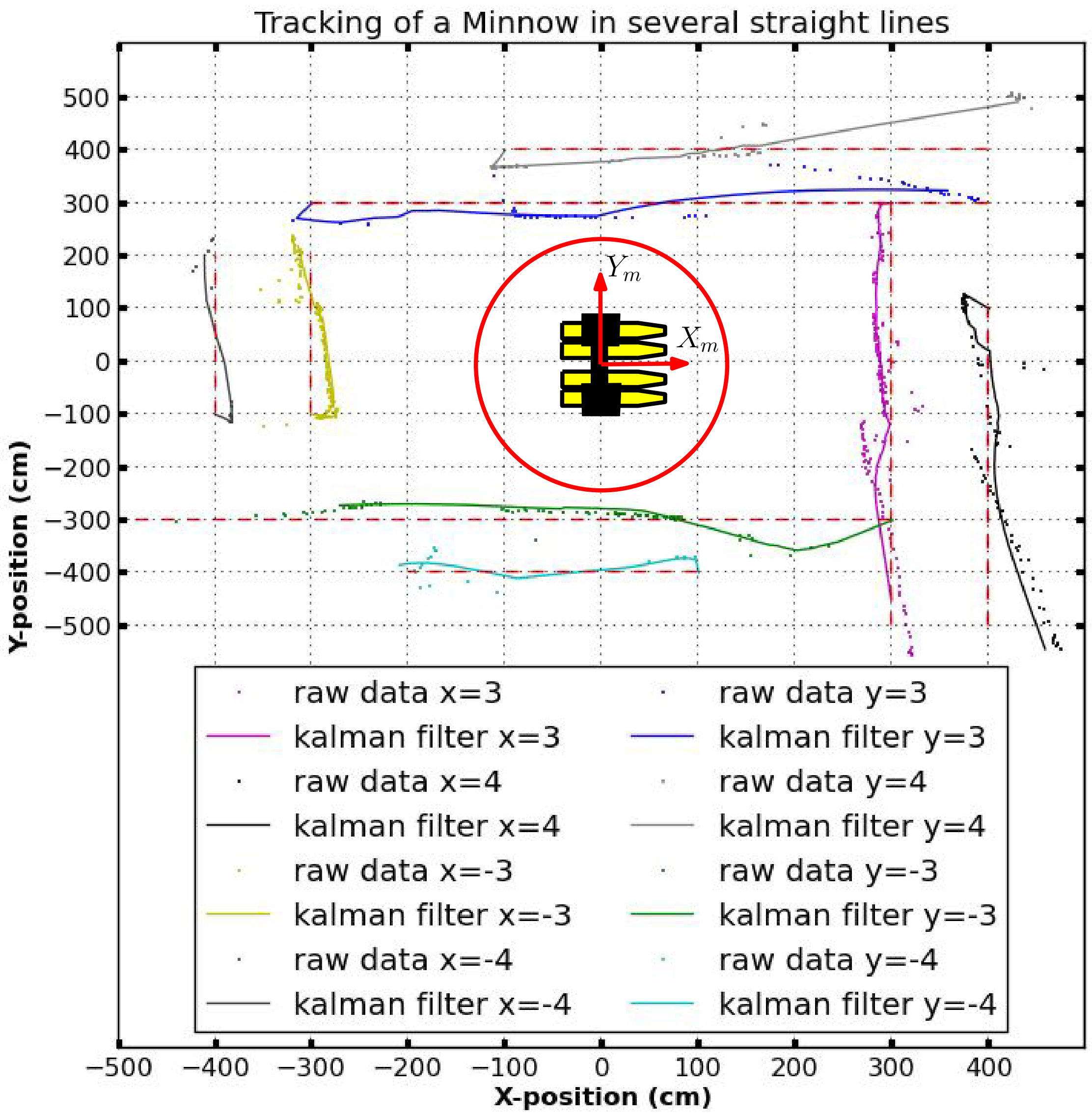


FIGURE 4.6: The tracking raw data of a Minnow when it was moving on a straight line in different distances as well as Kalman tracker results. The dashed lines depict the hand measured ground truth.

choosing $v(t)$ and $\omega(t)$ in order to make this happen (see more details in Appendix A). We assume that the robot has access to $(x, y, \theta)(t)$ of the vehicle using differential GPS and on-board compass data. Figure 4.8(a) shows a screen-shot of the *rviz* of Eddy and the waypoints. Figure 4.8(b)-(e) are screenshots from *rviz* simulation when Eddy is following a sequence of waypoints. The black squares depict the waypoints, the yellow box with a circle around it represents the Eddy with its circular footprint, and red line shows the path that Eddy has passed to reach each waypoint. The Algorithm 4.1 is used by Eddy to reach the target waypoint pose

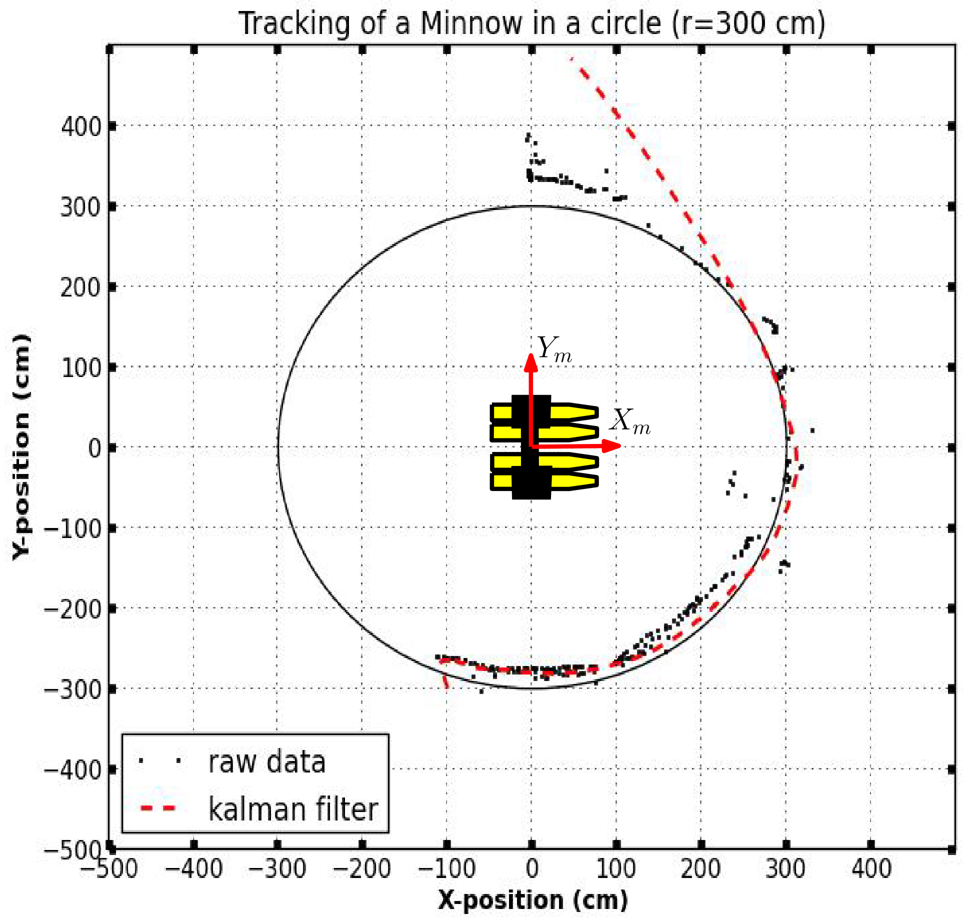


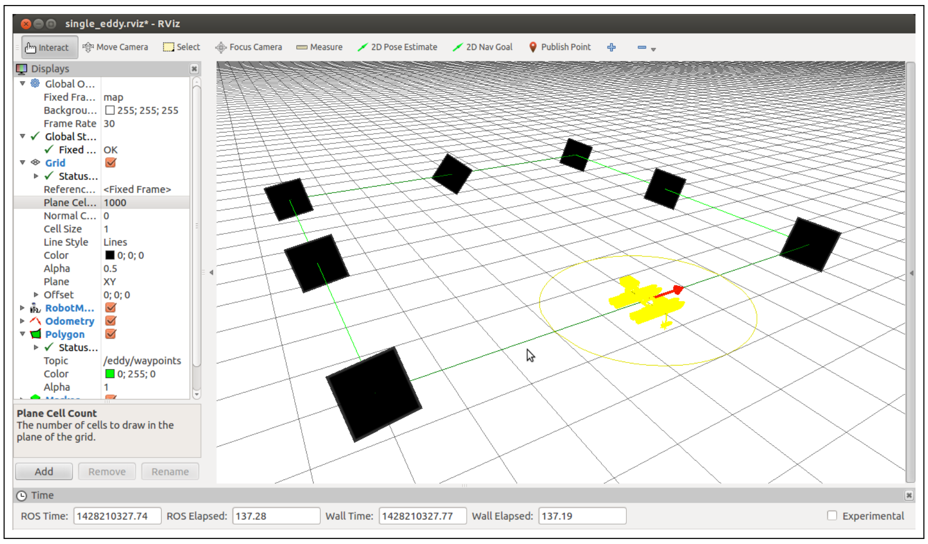
FIGURE 4.7: The tracking data of a Minnow when it was moving on a circular arc trajectory.

from its starting pose. This solution takes advantage of Eddy's ability to turn in place and to move in straight lines.

As shown in Figure 4.8(b)-(e), to drive Eddy to some waypoint, first it is rotated in place until it is heading to goal. Here an orientation tolerance is considered and hence the heading is not the exact heading. Then it is driven straight until reaches of the goal waypoint. If Eddy deviates too far from the heading to the goal, it corrects its heading. When it reaches the target or gets within a pre-defined distance from the target, it moves on to the next target in its waypoint list.

Minnows

Simulation results of minnow following a sequence of waypoints are depicted in Figure 4.9. Designing controllers for surface vessels subject to non-holonomic kinematic constraints associated

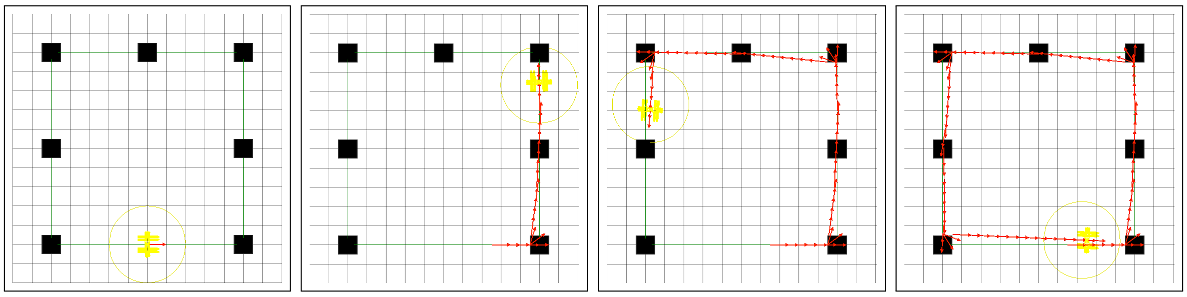


```
Global O...
  Fixed Fra...     map
  Backgrou...      255; 255; 255
  Frame Rate       30
Global St...
  Fixed ...        OK
Grid
  Status...
  Referenc...      <Fixed Frame>
  Plane Cel...     1000
  Normal C...      0
  Cell Size        1
  Line Style       Lines
  Color            0; 0; 0
  Alpha            0.5
  Plane            XY
  Offset           0; 0; 0
RobotM...
Odometry
Polygon
  Status...
  Topic            /eddy/waypoints
  Color            0; 255; 0
  Alpha            1
```

Plane Cell Count
The number of cells to draw in the plane of the grid.

ROS Time: 1428210327.74 ROS Elapsed: 137.28 Wall Time: 1428210327.77 Wall Elapsed: 137.19

(a)



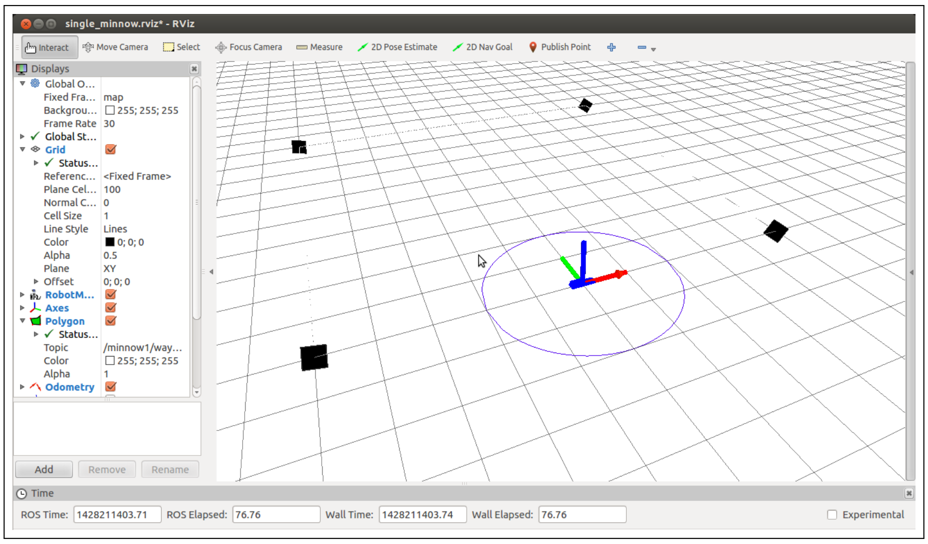
(b) (c) (d) (e)

FIGURE 4.8: (a) A screen-shot of the *rviz* visualization of Eddy and the waypoints. (b)-(e) Simulation results of Eddy following a sequence of waypoints. The black squares depict the waypoints, the yellow box with a circle around it represents Eddy with its circular footprint, and red line shows the path that Eddy followed to reach each waypoints. See text for details of the controller.

Algorithm 4.1 Eddy waypoint controller procedure

```
1: while $|(x, y) - (x\_goal, y\_goal)| > tolerance\_pos$ do
2:     desired_heading = atan(y_goal - y, x_goal - x)
3:     if $|desired\_heading - heading| > tolerance\_orientation$ then
4:         rotate robot in place by delta_theta towards desired_heading
5:     else
6:         drive robot forward at some velocity $v$
7:     end if
8: end while
```

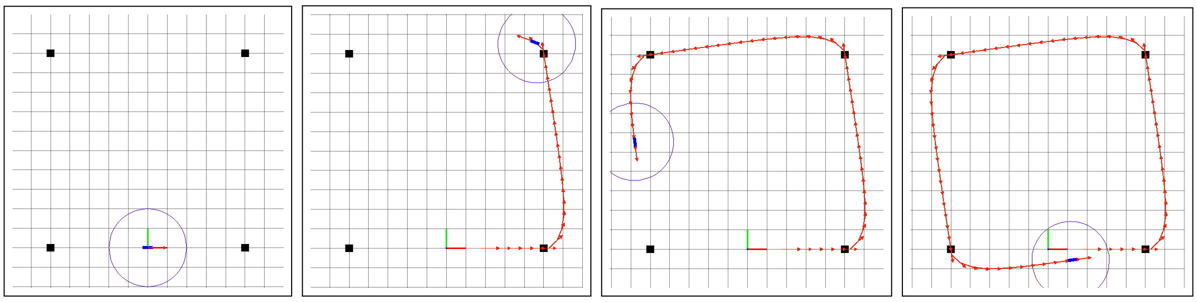
with rudder/thruster plants is more challenging. As with Eddy, control of a minnow robot is accomplished by providing the robot a sequence of waypoints $wp = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$ given a starting state (x, y, θ) for the robot. Driving the robot through these waypoints involves choosing vessel speed $u(t)$ and ruder angle $\delta(t)$ in order to make this happen (see more details in Appendix B). Here we assume that the robot has access to $(x, y, \theta)(t)$ of the vehicle through



Global O...
Fixed Fra... map
Backgrou... 255; 255; 255
Frame Rate 30
Global St...
Grid
Status...
Referenc... <Fixed Frame>
Plane Cel... 100
Normal C... 0
Cell Size 1
Line Style Lines
Color 0; 0; 0
Alpha 0.5
Plane XY
Offset 0; 0; 0
RobotM...
Axes
Polygon
Status...
Topic /minnow1/way...
Color 255; 255; 255
Alpha 1
Odometry

ROS Time: 1428211403.71 ROS Elapsed: 76.76 Wall Time: 1428211403.74 Wall Elapsed: 76.76 Experimental

(a)



(b) (c) (d) (e)

FIGURE 4.9: (a) A screen-shot of the *rviz* visualization of minnow and the waypoints. (b)-(e) Simulation results of minnow following a sequence of waypoints. The black squares depict the waypoints, the blue box with a circle around it represents the minnow with its circular footprint, and the red line shows the path that minnow executed to reach each waypoints. See text for details of the controller.

some combination of internal (e.g., onboard compass) and external (e.g., state estimation from the mother robot using an omnidirectional sensor) data. Figure 4.9(a) shows a screen-shot of the *rviz* visualization of minnow and the waypoints. Figure 4.9(b)-(e) are screenshots from *rviz* when the Minnow is following a sequence of waypoints. The black squares depict the waypoints, the blue box with a circle around it represents the Minnow with its circular footprint, and the red line shows the path that the Minnow has travelled to reach each waypoint. The Minnow robots use the same basic controller as Eddy, although due to the different kinematic/dynamic properties of the robot the effect on the behaviour of the robot is quite different. Specifically, it is not possible to change the heading of a Minnow robot without driving it forward (see Algorithm 4.2).

Algorithm 4.2 Minnow waypoint controller procedure

```
1: set robot velocity to $v$
2: while $|(x, y) - (x\_goal, y\_goal)| > tolerance\_pos$ do
3:     desired_heading = atan(y_goal - y, x_goal - x)
4:     a PID controller is used to correct the rudder angle so that the robot follows its desired
       heading
5: end while
```

4.6 Area coverage

This section presents a complete coverage solution for sensor-based area coverage using a hertegtenous fleet of ASVs. The validation of our algorithm is provided through simulation in a large-scale real body of water. First, the full turn strategy for the child robots in turning motion is validated. Then, the line abreast approach with non-holonomic constraints is simulated and visualized for the environment.

4.6.1 Waypoint simulation showing minnows executing their complex motions

As discussed in Section 3.2.3, given the kinematic constraints of the minnow robots they must execute complex synchronized motions in order to make the 90 degree turns required by the area coverage algorithm. Knowing that Minnow's safe turning diameter (D) in the absence of wind is approximately 4m, the simulation results for executing a Minnow are shown in Figure 4.10. Figure 4.10(a) shows the sequence of waypoints required for a Minnow to move in a required circle-like motion in turning motions. The created waypoints can not be too close to each other since the Minnow can not rotate sharply. Figure 4.10 shows hand-crafted relative waypoints that can be used to drive a Minnow robot through the necessary turn. The green squares depict the waypoints. Figure 4.10(b)-(d) illustrate the motion behaviour executed by a Minnow robot when following the grid-like motions required by the BCD algorithm. When the robot approaches a 90 degree turn, it passes through the point and follows a curved trajectory in the opposite direction of the intended new heading so that it can re-aquire the required BCD path shown in green in the Figure. In order for the robot to be able to follow this trajectory there must exist clear space outside of the free space covered by the BCD algorithm. In order to accomplish this, the boundary obstacles are dilated by the necessary turning radius of the Minnow robot.

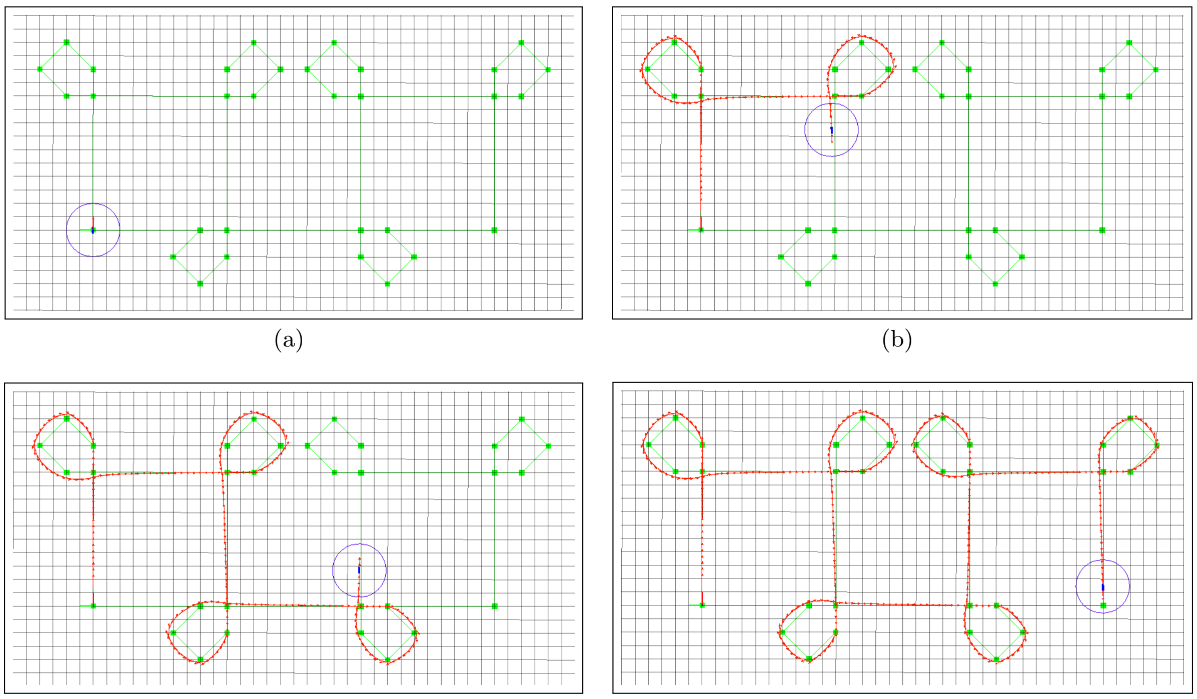

(a) (b) (c) (d)

FIGURE 4.10: Motion of a Minnow robot given a set of waypoints (shown as green squares). Here each green square represents a square of n meters in width. A Minnow robot cannot change orientation without moving, and so looping maneuvers are required to make the 90 degree turns required by the BCD algorithm. The red set of arrows show the position and orientation of the Minnow robot as it moves through the waypoints. In terms of the BCD algorithm, the Minnow executes the necessary straight line portions of the required path. The blue box with a circle around it represents the minnow with its circular footprint, and the red line shows the path that minnow executed to reach each waypoints.

Figure 4.11 shows the synchronized motion of two Minnow robots as they follow this turning trajectory. The structure of this turn is such that the two space-time curves followed by the Minnows do not intersect. The red and green arrows show the position and orientation of the two robots as they move in concert. When operating as part of the fleet, Eddy would be positioned mid-way between these two robots. Figure 4.12 shows a sequence of fleet members motions. When the Minnow robots execute their turning circles, Eddy can either follow the same type of path or should communications/localization restrictions permit, remain near the location where it should turn 90 degrees and then re-join the moving elements of the fleet as they pass by. In this later case, Eddy must adjust its speed along its trajectory so as to avoid the motion of the Minnow robots.

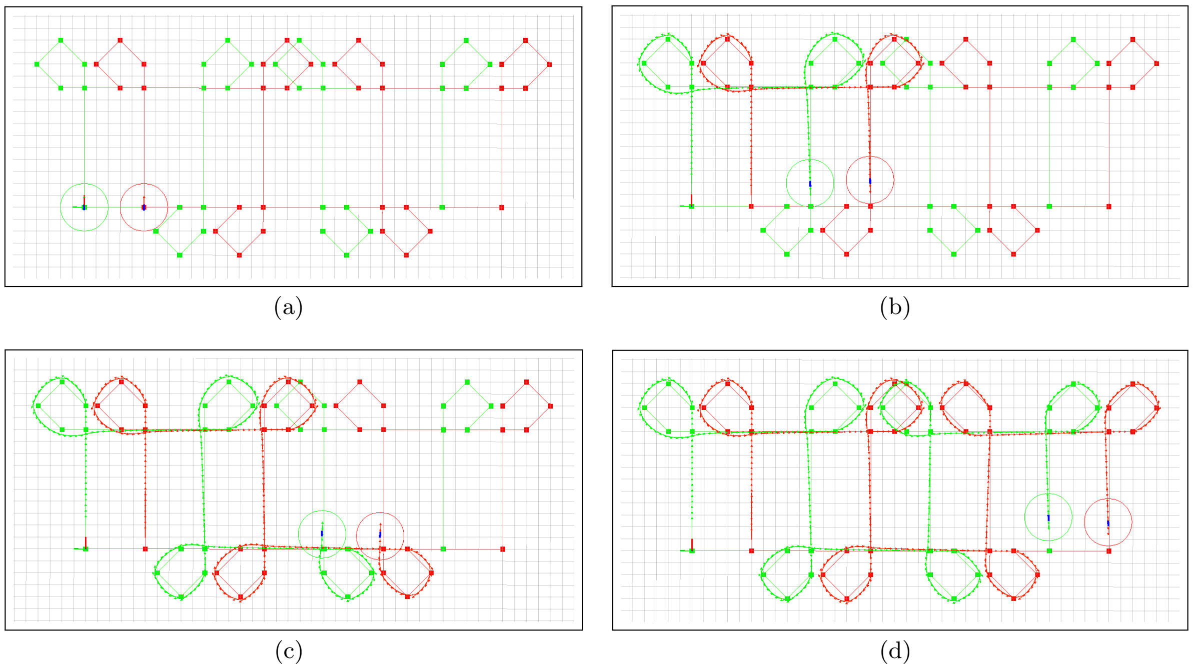
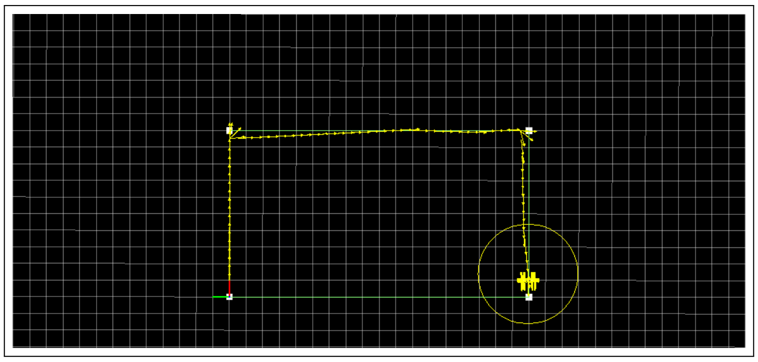

(a) (b) (c) (d)

FIGURE 4.11: Motion of a pair of Minnow robots given a set of waypoints. The red and green arrows show the position and orientation of the two robots as they move in concert. The blue boxes with circle around it represents the minnows with their circular footprint.

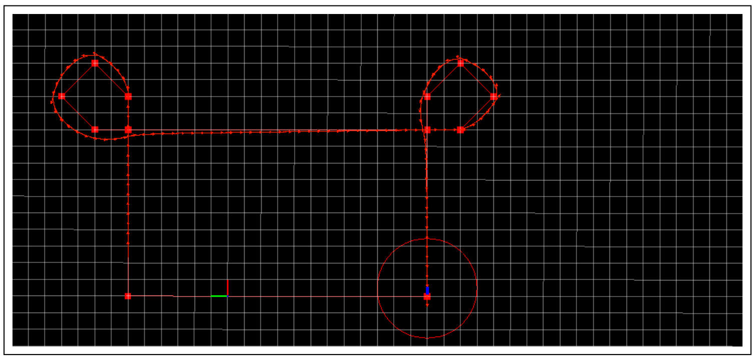
4.6.2 Waypoint simulation showing fleet executing some cell coverage

In order to represent our area coverage algorithm for a heterogeneous fleet of ASVs, the fleet simulated a complete coverage on a large 250m×150m real environment, with several curved and polygonal obstacles. Figure 4.13 shows the operation of the modified BCD algorithm and the waypoints for each member of the fleet considering their non-holonomic constraints. A safe area is introduced around boundaries and obstacles for turning the Minnow robots as shown in Figure 4.13(a). The environment, its Reeb graph and planned back-and-forth motion in each cell is shown in Figure 4.13(a). An Eulerian circuit is determined in order to identify the order in which the cells are to be scanned by the robot. The output from this process, a tour path connecting all of the environment cells, is also shown. The fleet waypoints are shown in Figure 4.13(c) and (d). The green, orange, and red lines show the waypoints of minnow1, Eddy, and minnow2 respectively. As a result, the waypoints for each member of fleet are created in order to provide a complete coverage of the entire environment.

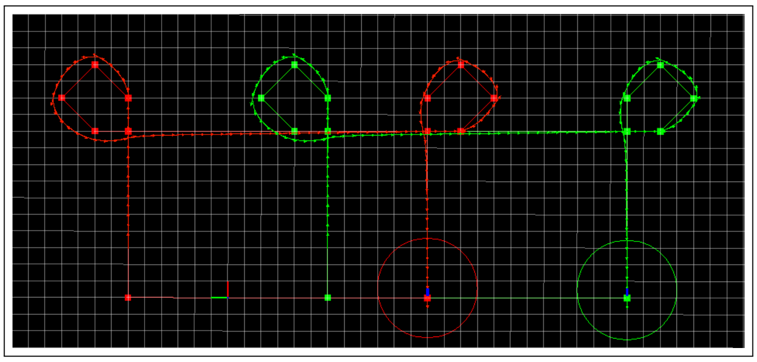
The results shown in Figure 4.13 were carried out in simulation. Here, the fleet footprint are considered as 12m. The linear speed of each robot was set to 0.5m/s. Also, the child robots position are set to +/- 4m of Eddy position. Figure 4.14 shows a detailed view of



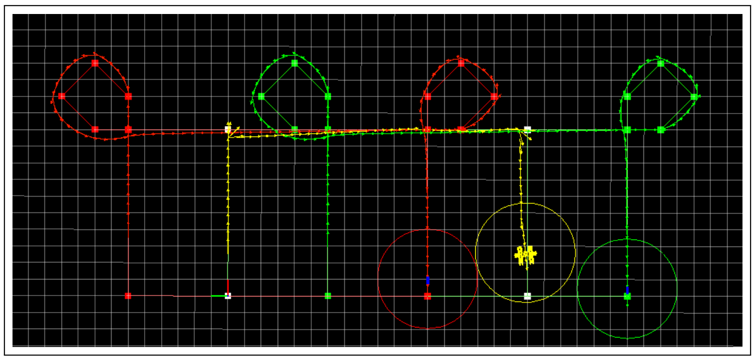
(a)



(b)



(c)



(d)

FIGURE 4.12: Motion of each member of fleet given a set of waypoints. (a) Eddy's motions. (b) Trajectory of a single Minnow. (c) Trajectory of a pair of Minnow. (d) Eddy and two Minnows in motion.

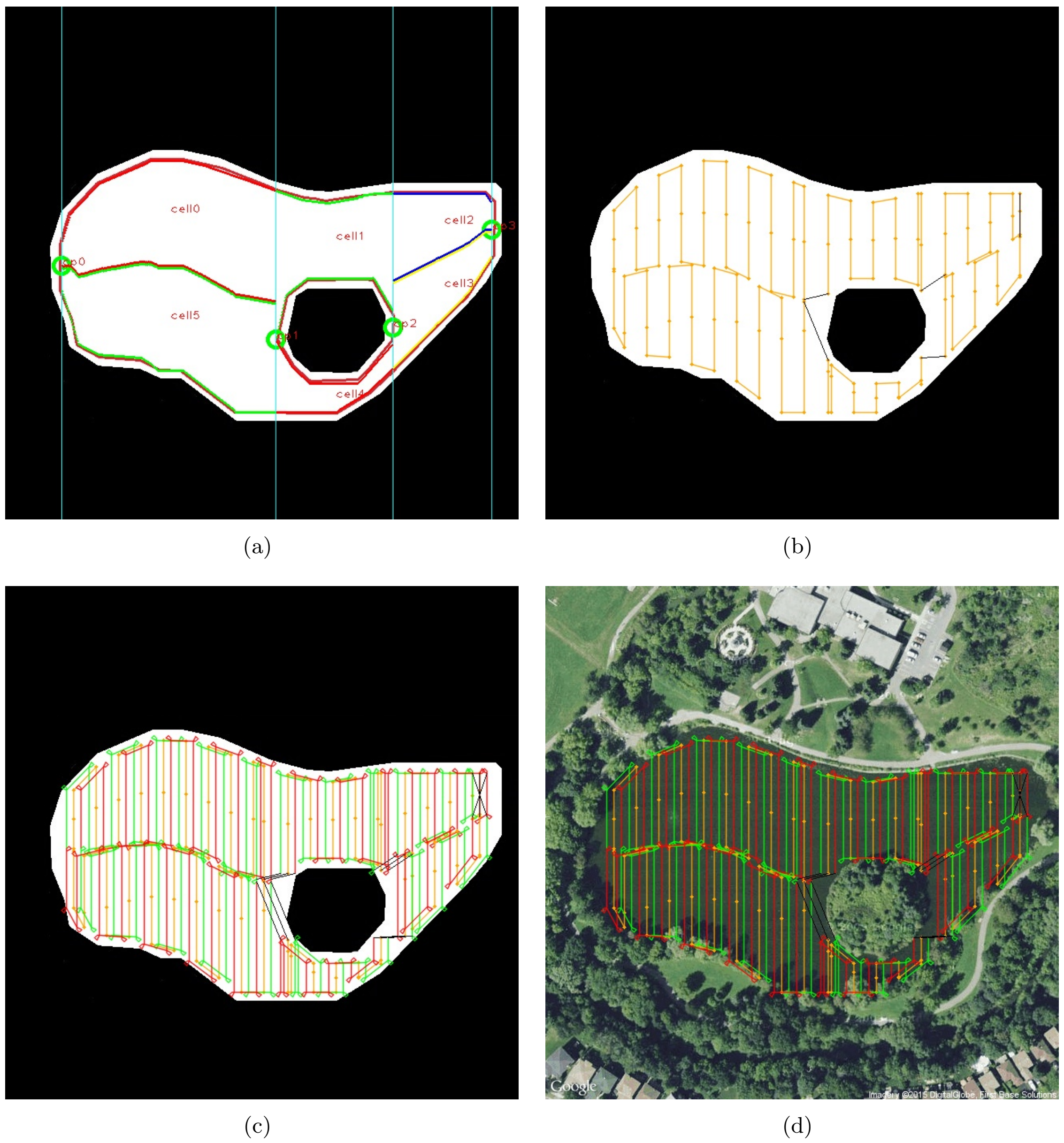
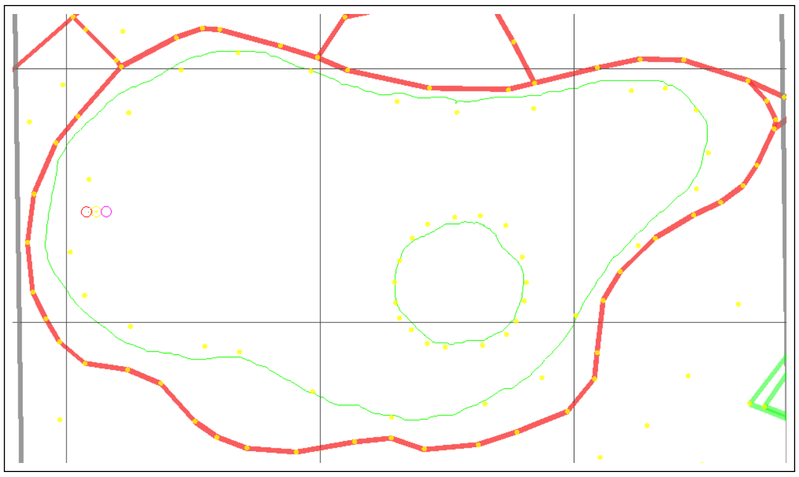
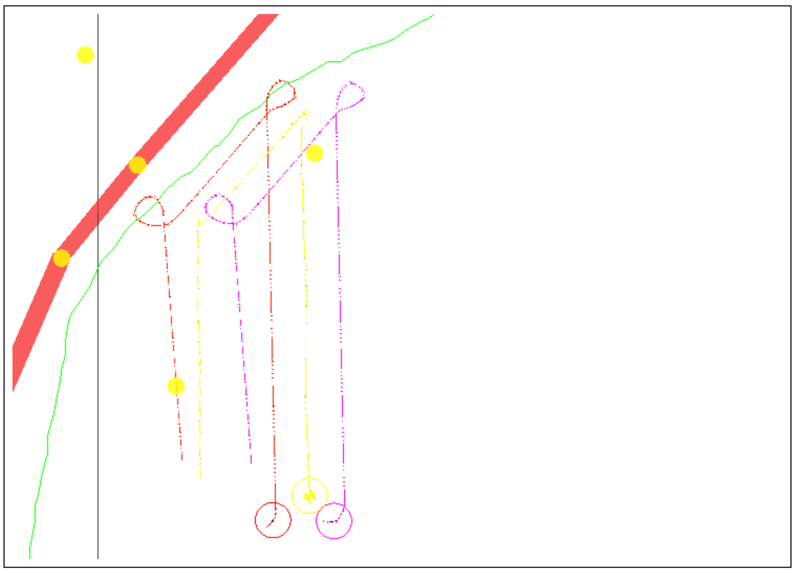


FIGURE 4.13: Motion coverage of Loafers Lake in Brampton, Ontario, Canada incorporating vessel non-holonomic constraints. The boundary is dilated ensuring that the robots can move outside of the coverage area. (a) The BCD algorithm is used to decompose the space into cells. (b) General fleet motion through the cells is planned as is motion between them. (c) Individual fleet element motions are planned based on this. (d) The final motion plan is overlaid on Google Earth imagery of the pond surface.

part of the motion plan of the fleet. The map of the environment are visualized in rviz using *osm_cartography* ROS package by translating OSM data (Exported form Open Street Map [20]) for a region into rviz markers. The visualized map in rviz is shown in Figure 4.14(a). It is assumed that the communication/localization radius is sufficiently large that the Minnow robots can execute their turning behaviour without Eddy remaining midway between them. Eddy moderates its speed as it approaches the waypoint at the end of each back and forth


(a)


(b)

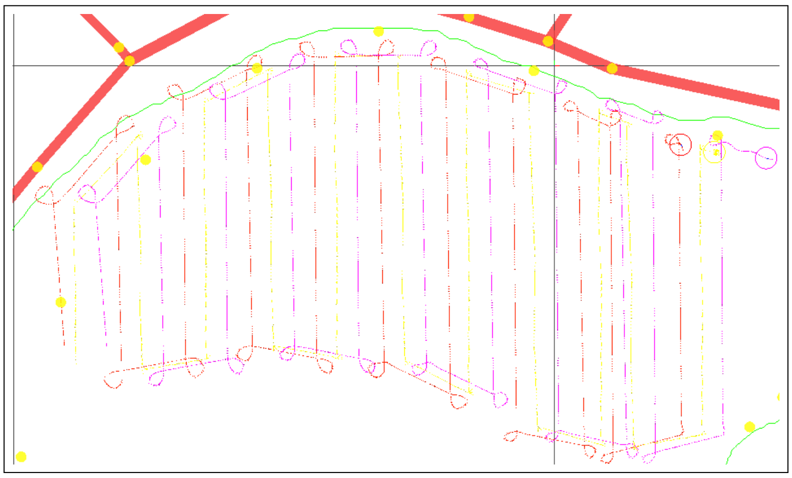

(c)

FIGURE 4.14: Simulated coverage paths for a 250m×150m region without considering environment conditions.

motion as necessary in order to remain out of the way of the Minnow robots as they execute their turns.

4.7 Summary

Sensor coverage with fleets of robots is a complex task requiring solutions to localization, communication, navigation and basic sensor coverage. The fleet used in this work consists of two Minnows and Eddy. The multi-roscore and foreign_relay approaches proved to be a highly successful and predictable mechanism for detection communication failure/reconnection between Eddy and the Minnow robots.

To localize Eddy, data from a DGPS and a compass is used. The precision of the DGPS employed in this work in a worst situation is between 0.2-2 meters. Better performance can be expected under better weather conditions. To localize the Minnows with respect to Eddy, an omnidirectional sensor is mounted that is proved to have a better results if a Kalman tracker is employed. Positional errors from a Minnow relative to Eddy can be expected to be under 0.9m.

Eddy and Minnow following a sequence of waypoints is simulated as well. Control of the Eddy robot (providing $v(t)$ and $\omega(t)$) is accomplished by providing the robot a sequence of waypoints given a starting state. The Minnow and Eddy robots use the same basic motion controller, although due to the different kinematic/dynamic properties of the robots the effect on the behaviour of the robots is quite different. This requires the waypoints for the Minnow robot's to be chosen with great care in order to be reachable.

Here we have demonstrated the overall algorithm in simulation. The individual robots have been tested in the field where communication, motion and localization strategies have been evaluated.

Chapter 5

Summary and future work

5.1 Summary and conclusions

This thesis investigated the development of a system for a complete sensor coverage of a known environment using a heterogeneous fleet of ASVs. The successful deployment of heterogeneous fleets of autonomous surface vessels requires solutions to a wide range of problems related to sensing, communications, command and control. This work addressed three of these problems: the development of a sensor coverage algorithm that considered the non-holonomic constraints associated with surface vessels, cooperative localization of elements of a surface fleet, and a suitable communication infrastructure. Beyond this, this work addressed a number of issues associated with the design and operation of the autonomous surface vessels themselves. Solutions were validated in the real world with real robotic platforms and with realistic simulation of the same devices on a real body of water.

The general problem of sensor coverage with an autonomous surface fleet would involve all possible combination of vessels, with all combinations of sensor and communication strategies. Clearly, such an problem is beyond the work of a single thesis. Rather than addressing the problem in general, in this work the problem of considering sensor coverage was limited to that involving a single kind of fleet: a fleet consisting of two classes of robots, one with considerable capabilities in terms of sensors and communication structure, coupled with a number of less well equipped robots that must rely on the more well equipped robot for localization and other tasks.

Specific issues addressed in developing this single fleet example, and surface coverage for it was included.

Development of the Minnow robot platform. Dealing with the area coverage problem related for this heterogeneous fleet involved developing solutions to a range of problems related to the actual design and construction of the fleet. A radio controlled (RC) hobby boat was repurposed as each Minnow (the child robot). The vessel is powered by a single DC motor driving a prop while an RC servo motor is used to provide positional control over the rudder. Standard electronics components are used to interface with the RC boat electronics, and the vessel was augmented with GPS, vision, and a tilt-compensated compass to provide the necessary onboard sensing capabilities. Each robot is self-contained for power and computation and communicates with other robots via a standard WIFI network. Each Minnow runs its own roscore, and communicates with other elements of the fleet via the wireless access point mounted on Eddy. A ROS-based control and sensing infrastructure is used to operate the vehicle on-board.

Modification of the Eddy platform. A Kingfisher M100 platform developed by Clearpath Robotics was modified as Eddy (the mother robot). The Kingfisher M100 is essentially a differential-drive surface vessel powered by two outboard motors. As shipped this robot is equipped with a differential GPS as well as a long-range radio communication channel for off-board command and control. Eddy is self-contained for power, computation, and communication with other robots. The vehicle has been augmented with additional onboard computation and sensors as well as through additional flotation support. Eddy is equipped with a wireless radio modem to communicate with its base station equipment and an additional onboard wireless router is used to communicate with the child robots operating in proximity to the Eddy platform. In order to increase the robot's sensing abilities, the Kingfisher was also modified through the development of a 360 degree video sensor. Eddy – like all of the robots in the heterogenous fleet – operates its own ROS master with communication between fleet elements provided through a multi-master ROS framework that supports a heartbeat signal to detect communications failure between Minnows and Eddy. This heartbeat signal is straightforward given the star-topology nature of the fleet with one central mother node (Eddy) and multiple child nodes (Minnows).

Localization algorithm for the minnow robot. For teams of robots, team pose estimation and communication/synchronization are key requirements. 2D pose estimation for a team can be performed in a number of ways. Here we explored the relatively simple task of providing elements of the fleet complete pose estimates not only of themselves, but also of other elements of the fleet. In the work presented here the fleet is modelled in a heterogeneous fashion in which one element of the fleet – the mother robot – has access to full pose estimation through an onboard compass and access to a DGPS receiver. Other elements of the fleet are equipped with a compass for orientation, but rely on the mother robot to provide good positional estimates. The process of obtaining these estimates is performed through a relatively straightforward omnidirectional vision system and the use of pre-positioned targets mounted on each of the child robots. Notwithstanding the simplified planar water surface model and the simple projection model of the visual target on the water plane, positional accuracy is quite good, certainly less than one meter for targets in proximity to the mother robot.

The other aspect of providing coordinated localization of the fleet involves the development of an appropriate software/communications strategy to monitor communications among the fleet elements. Here we utilize a star topology and a heartbeat strategy, which are appropriate given the structure of the fleet.

Motion controllers for the Minnow and Eddy platform. Motion control is accomplished by providing the robot a sequence of waypoints given a starting state. Eddy first spins in place until the robot's orientation coincides with the line from the starting position to the target position considering some orientation tolerance. Then, Eddy drives forward until the position coincides with the target waypoint position considering some distance tolerance. In this step, the robot frequently stops to correct its heading based on the current position and the compass data. Minnow robots use the same basic motion controller as Eddy, although due to the different kinematic/dynamic properties of the Minnow robots the effect on the behaviour of the robots is quite different. Specifically, it is not possible to change the heading of a Minnow robot without driving it forward.

Adaptation of standard coverage algorithms to meet the special needs of the robot fleet. Given a heterogeneous collection of robots that must remain in close proximity, a BCD-based approach will obtain a basic plan for the motion of the fleet. Within the fleet it is then necessary to ensure that the individual robots can actually execute the paths that have been generated. Many aquatic robots rely on a propeller/rudder strategy. This imposes kinematic/dynamic constraints on the robot which precludes the sharp turns that are assumed by the

BCD algorithm. A turning maneuver is used to allow the robots to meet the requirements of the BCD algorithm. We have demonstrated the algorithm in simulation. The robots have been tested in the field where communication and localization strategies have been evaluated.

5.2 Future work

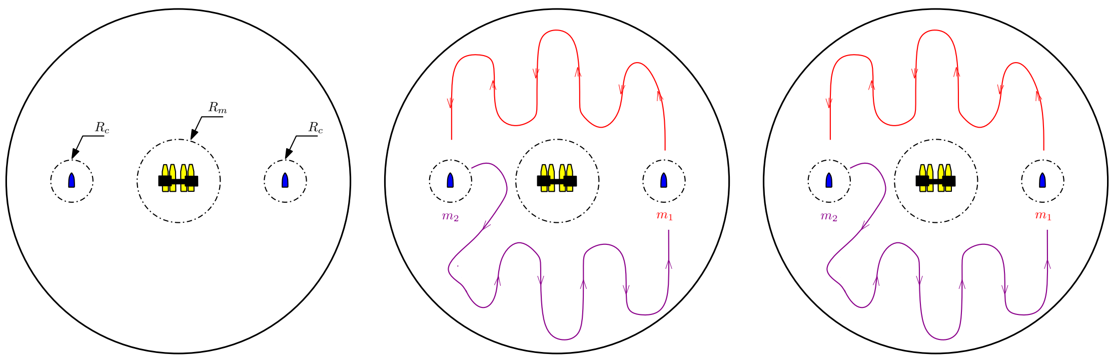
Some interesting areas remain for potential future investigation and they are described in the following paragraphs.

Chapter 4, developed in part, a point-to-point navigation algorithm for Eddy and the Minnow robot classes. The motion control algorithms were developed in the absence of environmental factors (e.g., wind, wave action and the like). As the robots are commanded to move, such environmental factors will result in systematic biases in the motion executed by the robots. Is it possible to exploit this information and tune (or adapt) the motion controller to account for these external effects? Also, we are interested to tune the PID controller for the choppy waters as well.

Another interesting area to explore is the use of the fleet to provide complete sensor coverage of small bodies of water and other liquids in order to provide a geotagged representation of water surface events. Full field tests of the fleet performing sensor coverage is intended to be implemented in the near future.

The robot fleet tested in this work consisted of a single mother robot (Eddy) and a very small number of Minnow robots. This fleet size was a consequence of the small number of robots available for testing and the effort involved in keeping even this small number of robots running in the field. Another potential area of interest involves the number of robots in the fleet and then evaluating the communication/synchronization system developed in this work and its performance for a larger fleet. Although in theory the algorithms should just scale, there exist real-life considerations in terms of the machines themselves, and the communication infrastructure that could and should be evaluated.

Finally, another area worthwhile investigating is to design a dynamic formation instead of rigid formation for fleet members. A critical issue for the rigid formation fleet is that when the mother robot executes a motion that requires the child robots to violate their non-holonomic constraints, the entire fleet must pause while the child robots manoeuvre so as to maintain their



(a) (b) (c)

FIGURE 5.1: Dynamic configuration approach. (a) depicts the fleet members. In (b), construction of coverage path of mother and child robots are illustrated. (c) shows the area covered by child robots in mother's LOS.

rigid position relative to the mother. This requirement for maneuvering must also be integrated into understanding that coverage of the robot fleet during this motion. In contrast, in the dynamic formation, child robots would not necessarily remain in a rigid constellation around the mother robot. In this approach, the fleet coverage footprint changes dynamically as the fleet moves and hence a better complete coverage can be provided. For area coverage following this method, the mother robot stops at predefined stations along its waypoints and the child robots move in a predefined waypoints so that the total area in LOS of the mother robot is covered at each station. Figure 5.1 shows a dynamic configuration approach and a coverage example at a station. When the coverage around the mother's station is complete, the robots cooperatively move following the defined path for the motion of fleet of robots, until the entire path is visited. This configuration requires a more sophisticated version of BCD but may provide efficiencies in terms of coverage. In addition, by sensing individual robot failure and dynamically re-allocating robots, increased task robustness can be exhibited.

Bibliography

[1] Clearpath Robotics Company, 2011. URL http://www.clearpathrobotics.com.

[2] Honeywell, Sep. 2011. URL http://www.honeywell.com.

[3] Arduino, Sep. 2011. URL http://www.arduino.cc.

[4] ROS, meet Arduino, Sep. 2011. URL http://www.ros.org/news/2011/02/ros-meet-arduino.html.

[5] FitPC, 2011. URL http://fit-pc.com/web/products/fit-pc2/.

[6] PandaBoard, June 2012. URL http://pandaboard.org.

[7] Rosserial, Sep. 2012. URL http://www.ros.org/wiki/rosserial?distro=electric.

[8] Differential GPS, 2013. URL http://en.wikipedia.org/wiki/Differential_GPS.

[9] Ocean Server Thechnology INC., 2013. URL http://www.ocean-server.com/compass.html.

[10] Ublox LEA-6T module with Precision Timing, 2013. URL http://www.u-blox.com/en/gps-modules/u-blox-6-timing-module/lea-6t.html.

[11] Video4Linux, July 2013. URL https://en.wikipedia.org/wiki/Video4Linux.

[12] Deepwater Horizon oil spill, December 2013. URL http://en.wikipedia.org/wiki/Deepwater_Horizon_oil_spill.

[13] D-Link WIFI camera, 2014. URL http://ca.dlink.com/products/connected-home/cloud-camera-1000-day-network-cloud-camera-2/.

[14] Multimaster fkie, 2014. URL http://wiki.ros.org/multimaster_fkie.

[15] Foreign relay, 2014. URL http://wiki.ros.org/foreign_relay?distro=groovy.

[16] Multimaster, 2014. URL http://wiki.ros.org/multimaster.

[17] Multimaster Special Interest Group, 2014. URL http://wiki.ros.org/sig/Multimaster.

[18] Image processing in OpenCV, 2014. URL http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html.

[19] Camera Calibration and 3D Reconstruction, 2014. URL http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.

[20] Open Street Map, 2014. URL http://www.openstreetmap.org.

[21] Rocon multimaster, 2014. URL http://wiki.ros.org/rocon_multimaster?distro=indigo.

[22] RTKLIB, 2014. URL http://www.rtklib.com.

[23] RTKLIB ROS, 2014. URL https://github.com/ethz-asl/rtklibros.

[24] Static Maps API V2 Developer, 2015. URL https://developers.google.com/maps/documentation/staticmaps/.

[25] E. U. Acar and H. Choset. Robust sensor-based coverage of unstructured environments. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2001*, volume 1, pages 61–68, Maui, HI, 2001.

[26] E. U. Acar and H. Choset. Sensor-based coverage of unknown environments: Incremental construction of morse decompositions. *The International Journal of Robotics Research*, 21(4):345 – 366, 2002.

[27] E. U. Acar, H. Choset, A. Rizzi, N. Atkar, and D. Hull. Morse decompositions for the coverage task. *The International Journal of Robotics Research*, 21(4):331 – 344, 2002.

[28] E. U. Acar, H. Choset, and J. Y. Lee. Sensor-based coverage with extended range detectors. *IEEE Transactions on Robotics*, 22(1):189 –198, 2006.

[29] A. Agarwal, C. V. Jawahar, and P. J. Narayanan. A survey of planar homography estimation techniques. Technical report, Centre for Visual Information Technology, IIIT Hyderabad, 2005.

[30] N. Agmon, N. Hazon, and G. Kaminka. The giving tree: Constructing trees for efficient offline and online multi-robot coverage. *Annals of Math and Artificial Intelligence*, 52 (2-4):143 – 168, 2008.

[31] T. Bailey, M. Bryson, H. Mu, J. Vial, L. McCalman, and H. Durrant-Whyte. Decentralized cooperative localization for heterogeneous teams of mobile robots. In *IEEE International Conference on Robotics and Automation, ICRA 2011*, pages 2859 – 2865, Shanghai, China, 2011.

[32] T. Balch and R.C. Arkin. Behaviour-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):462 – 472, 1998.

[33] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *the IEEE International Conference on Dependable Systems and Networks (DSN 2002)*, pages 354 – 363, 2002.

[34] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *IEEE International Conference on Dependable Systems and Networks (DSN 2002)*, page 354 363, 2002.

[35] N. L. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory 1736-1936.* Clarendon Press, Oxford, 1976.

[36] B. Bishop. Design and control of platoons of cooperating autonomous surface vessels. In *7th Annual Maritime Transportation System Research and Technology Coordination Conference*, November 2004.

[37] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14 – 23, 1986.

[38] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrus. Collaborative multi-robot exploration. In *Proceedings of IEEE International Conference on robotics and Automation, ICRA 2000*, San Francisco, AC, April 2000.

[39] A. Calce, P. Mojiri Forooshani, A. Speers, K. Watters, T. Young, and M. Jenkin. Autonomous aquatic agents. In *Proceedings of International Conference on Agents and Artificial Intelligence, ICAART 2013*, Barcelona, Spain, February 2013.

[40] H. Choset. Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots*, 9:247 – 253, 2000.

[41] H. Choset. Coverage for robotics - a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113 – 126, 2001.

[42] H. Choset and P. Pignon. Coverage path planning: The boustrophedon cellular decomposition. In *International Conference on Field and Service Robotics*, Canberra, Australia, 1997.

[43] A. L. Christensen, R. OGrady, and M. Dorigo. From fireflies to fault-tolerant swarms of robots. In *IEEE Transactions on Evolutionary Computation*, volume 13, pages 754–766, 2009.

[44] Clearpath Robotics. *KINGFISHER M100 Overview*, 2011. URL `http://www.clearpathrobotics.com/kingfisher`.

[45] Clearpath Robotics. *KINGFISHER M100 User Manual*, 2011.

[46] J. Colegrave and A. Branch. A case study of autonomous household vacuum cleaner. In *AIAA/NASA Conference on Intelligent Robots for Factory Field, Service, and Space (CIRFFSS)*, 1994.

[47] A. J. Davison and N. Kita. Active visual localization for cooperating inspection robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2000*, volume 3, pages 1709–1715, Takamatsu, Japan, 2000.

[48] E. Dubrofsky. Homography estimation. Computer Science, MSc thesis, University of British Columbia, 2009.

[49] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics.* New York, NY: Cambridge University Press, 2010.

[50] K. Easton and J. Burdick. A coverage algorithm for multi-robot boundary inspection. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2005*, pages 727 – 738, Barcelona, Spain, 2005.

[51] J. Edmonds and E. L. Johnson. Matching euler tours and the chinese postman problem. *Mathematic Programming*, 5(1):88–124, 1973.

[52] M. Drorand A. Efrat, A. Lubiw, and J. Mitchell. Touring a sequence of polygons. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, STOC 2003*, pages 473 – 482, New York, NY, USA, 2003.

[53] A. Fomenko and T. L. Kunii. *Topological modelling for visualization.* Tokyo: Springer-Verlag, 1997.

[54] D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Prentice Hall of India, 2002.

[55] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8:325 – 344, 2000.

[56] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31:77 – 98, 2001.

[57] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry: Theory and Applications*, 24(3):197 – 224, 2003.

[58] E. Gat and G. Dorais. Robot navigation by conditional sequencing. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 1994*, San Diego, CA, 1994.

[59] D. Grieneisen. Real time kinematic GPS for micro aerial vehicles. Electrical Engineering, semester-thesis, ETH, Zurich, Swiss, 2012.

[60] Y. Guo and M. Balakrishnan. Complete coverage control for nonholonomic mobile robots in dynamic environments. In *Proceedings of IEEE International Conference on Robotics and Automation, ICRA 2006*, pages 1704 – 1709, Orlando, FL, 2006.

[61] K. R. Guruprasad, Z. Wilson, and P. Dasgupta. Complete coverage of an initially unknown environment by multiple robots using voronoi partition. Technical report, Department of Computer Science University of Nebraska at Omaha, 2011.

[62] I. A. Hameed, D. D. Bochits, K. C. Swain, O. Green, C. G. Sorensen, and N. Micheal. Evaluation of field coverage algorithms for agricultural machines. In *International Agricultural Engineering Conference*, Bangkok, Thailand, 7 - 10 2009.

[63] C. Harris and M. Stephens. A combined corner and edge detector. In *In Proceeding of Fourth Alvey Vision Conference*, pages 147 – 151, 1988.

[64] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100 – 107, 1968.

[65] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision: Second edition.* Cambridge University Press, 2004.

[66] N. Hazon and G. A. Kaminka. Redundancy, efficiency and robustness in multi-robot coverage. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2005*, pages 735 – 741, April 2005.

[67] N. Hazon and G. A. Kaminka. On redundancy, efficiency, and robustness in coverage for multiple robots. *Robotics and Autonomous Systems*, 56(12):1102 –1114, 2008.

[68] M. Hofmeister and M. Kronfeld. Multi-robot coverage considering line-of-sight conditions. In *7th IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*, pages 121 – 126, Lecce, Italy, 2010.

[69] M. Hofmeister, M. Kronfeld, and A. Zell. Cooperative visual mapping in a heterogeneous team of mobile robots. In *IEEE International Conference on Robotics and Automation, ICRA 2011*, pages 1491 – 1496, Shanghai, China, 2011.

[70] A. Howard, L. E. Parker, and G. S. Sukhatme. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *International Journal of Robotics Research*, 25:431– 447, 2006.

[71] K. Kato, H. Ishiguro, and M. Barth. Identifying and localizing robots in a multi-robot system environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 1999*, volume 2, pages 966 – 971, Kyongju, South Korea, 1999.

[72] C. S. Kong, A. P. New, and I. Rekleitis. Distributed coverage with multi-robot system. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2006*, Orlando, FL, 2006.

[73] R. Kurazume and S. Hirose. An experimental study of a cooperative positioning system. *Autonomous Robots*, 8:43 – 52, 2000.

[74] R. Kurazume, S. Nagata, and S. Hirose. Cooperative positioning with multiple robots. In *Proceedings of IEEE International Conference on Robotics and Automation, ICRA 1994*, volume 2, pages 1250 – 1257, 1994.

[75] J. C. Latombe. *Robot Motion Planning: Edition en anglais.* Springer, 1991.

[76] K. Leung, T. D. Barfoot, and H. H. T. Liu. Decentralized localization for dynamic and sparse robot networks. In *IEEE International Conference on Robotics and Automation, ICRA 2009*, pages 3135 – 3141, Kobe, Japan, 2009.

[77] K. Leung, T. D. Barfoot, and H. H. T. Liu. Decentralized localization of sparsly-communicating robot networks: A centralized-equivalent approach. *IEEE Transaction on Robotics*, 26(1):62 – 77, 2010.

[78] R. Madhavan, K. Fregene, and L. E. Parker. Distributed heterogeneous outdoor multi-robot localization. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2002*, volume 1, 2002.

[79] R. Madhavan, K. Fregene, and L. E. Parker. Distributed cooperative outdoor multirobot localization and mapping. *Autonomous Robots*, 17:23 – 39, 2004.

[80] R. Mannadiar and I. Rekleitis. Optimal coverage of a known arbitrary environment. In *IEEE International Conference on robotics and Automation, ICRA 2010*, pages 5525 – 5530, Anchorage, AK, 2010.

[81] A. Martinelli, F. Pont, and R. Siegwart. Multi-robot localization using relative observation. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2005*, pages 2797 – 2802, Barcelona, Spain, 2005.

[82] T. Min and H. Yin. A decentralized approach for cooperative sweeping by multiple mobile robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 1998*, volume 1, Victoria, BC, 1998.

[83] A. Mourikis and S. Roumeliotis. Performance analysis of multirobot cooperative localization. *IEEE Transactions on Robotics*, 22(4):666 – 681, 2006.

[84] F. Ntawiniga. Head motion tracking in 3d space for drivers. Electrical Engineering, MSc thesis, Laval University, Quebec, Canada, 2008.

[85] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press New York, 1987.

[86] M. Ozkan, G. Kirlik, O. Parlaktuna, A. Yufka, and A. Yazici. A multi-robot control architecture for fault-tolerant sensor-based coverage. *International Journal of Advanced Robotic Systems*, 7(1), 2010.

[87] L. E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220 – 240, 1998.

[88] L. E. Parker, B. Kannan, X. Fu, and Y. Tang. Heterogeneous mobile sensor net deployment using robot herding an line-of-sight formations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2003*, Las Vegas, Nevada, 2003.

[89] L. E. Parker, B. Kannan, F. Tang, and M. Bailey. Tightly-coupled navigation assistance in heterogeneous multi-robot teams. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems , IROS 2004*, volume 1, Sendai, Japan, 2004.

[90] O. Parlaktuna, A. Sipahiohlu, G. Kirlik, and A. Yazici. Multi-robot sensor-based coverage path planning using capacitated arc routing approach. In *IEEE International Symposium on Intelligent Control part of 2009 IEEE Multi-conference on Systems and Control*, Saint Petersburg, Russia, 2009.

[91] W. L. Pearn and T. C. Wu. Algorithms for the rural postman problem. *Computers and Operations Research*, 22(8):819 – 828, 1995.

[92] E. Prassler, A. Ritter, C. Schaeffer, and P. Fiorini. A short history of cleaning robots. *Autonomous Robots*, 9:211 – 226, 2000.

[93] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing (3rd ed.).* New York: Cambridge University Press, 2007.

[94] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *Open-Source Software workshop at the International Conference on Robotics and Automation (ICRA)*, 2009.

[95] I. Rekleitis and G. Dudek. Multi-robot collaboration for robust exploration. *Annals of Mathematics and Artificial Intelligence*, 31:7 – 40, 2001.

[96] I. Rekleitis, G. Dudek, and E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 1997*, volume 2, Nagoya, Japan, 1997.

[97] I. Rekleitis, V. Lee-Shue, A. Peng New, and H. Choset. Limited communication, multi-robot team based coverage. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2004*, volume 4, pages 3462 – 3468, 2004.

[98] I. Rekleitis, A. Peng New, E. Rankin, and H. Choset. Efficient boustrophedon multi-robot coverage: an algorithmic approach. 52:109 –142, 2008.

[99] I. M. Rekleitis, G. Dudek, and E. E. Milios. On multiagent exploration. In *Proceedings of Vision Interface*, Vancouver, Canada, 1998.

[100] S. Roumeliotis and G. Bekey. Distributed multi-robot localization. *IEEE Transactions on Robotics and Automation*, 18(5):781 – 795, 2002.

[101] S. W. Ryu, Y. h. Lee, T. Y. Kuc, S. H. Ji, and Y. S. Moon. A search and coverage algorithm for mobile robot. In *8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 815 – 821, Incheon, Korea, 2011.

[102] P. Sturm and S. Maybank. On plane-based camera calibration: A general algorithm, singularities, applications. In *the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 432 – 437, Fort Collins, CO, USA, 1999.

[103] C. Trevai, J. Ota, and T. Arai. Multiple mobile robot surveillance in unknown environments. *Advanced Robotics*, 21(7):729–749, 2007.

[104] H. H. Viet, V. H. Dang, M. N. Uddin Laskar, and T. Chung. BA*: an online complete coverage algorithm for cleaning robots. *Applied Intelligence*, 39:217 – 235, 2013.

[105] A. Xu, C. Viriyasuthee, and I. Rekleitis. Optimal complete terrain coverage using an unmanned aerial vehicle. In *IEEE International Conference on Robotics and Automation, ICRA 2011*, pages 2513 – 2519, Shanghai, China, 2011.

[106] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330 – 1334, 2000.

[107] X. Zheng, S. Jain, S. Koenig, and D. Kempe. Multi-robot forest coverage. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2005*, page 38523857, 2005.

Appendix A

Eddy: a differential autonomous surface vessel

This appendix describes the design of a large autonomous surface vessel (see Figure 1.1(b)), known as Eddy, that is the main element of the fleet of vessels used in this thesis. The vehicle described here is a modified version of the Kingfisher M100 platform developed by Clearpath Robotics company [1]. As shipped this robot is equipped with a differential GPS as well as a long range radio communication channel. For the work described in this thesis the vehicle has been augmented with additional onboard computation and sensors.

A.1 Basic vessel design

The Kingfisher M100 platform (Eddy) is an autonomous surface vessel (ASV) that is designed for environmental engineers, consultants and researchers who perform lengthy hydrological studies in difficult to access bodies of water. The manufacturer describes it as a "portable, agile, and easy-to-use unmanned surface vehicle for rapid prototyping applications" [44]. As shipped this ASV is equipped with a differential GPS, a compass, and a PC that is configured to run Ubuntu Linux and the Robot Operating System (ROS).

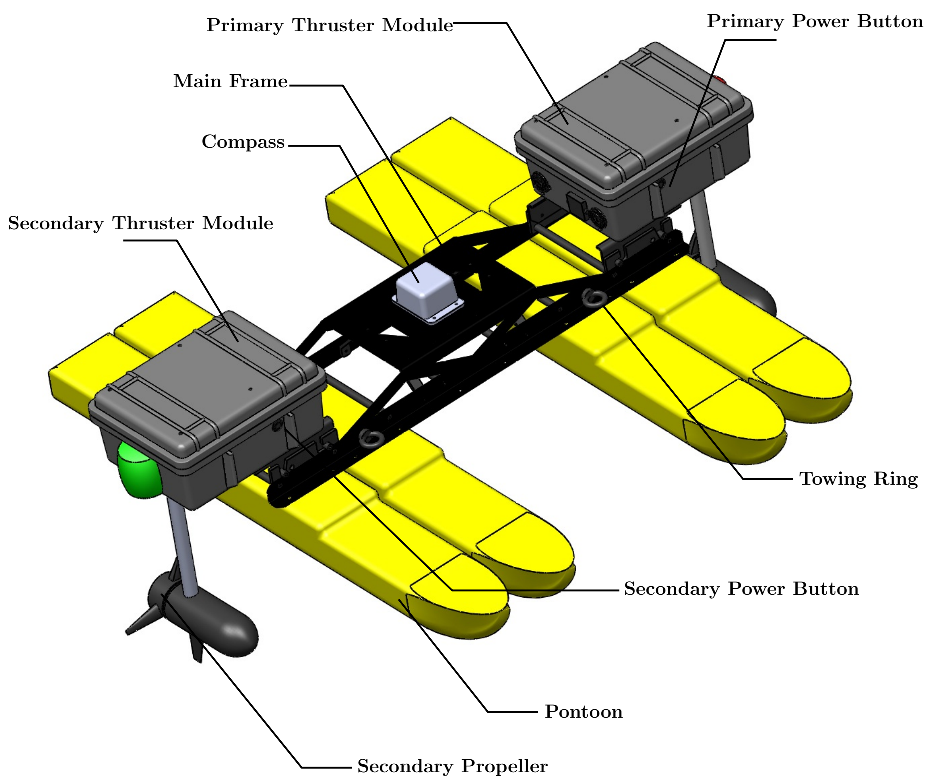
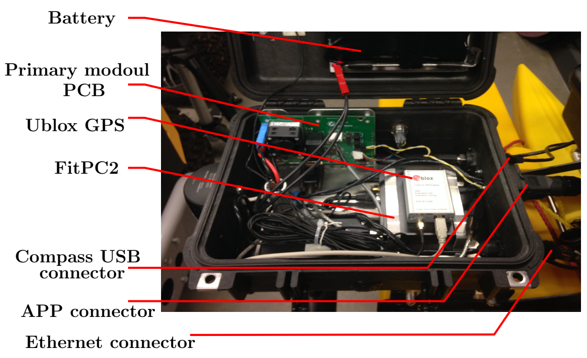


FIGURE A.1: Kingfisher M100 at a glance. Figure copyright Clearpath robotics and is used with permission.

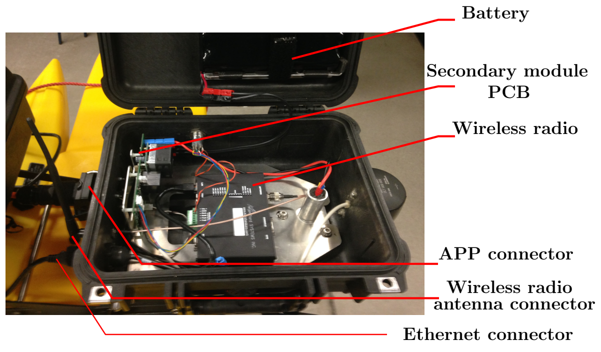
A.1.1 Hardware

The Kingfisher fits within a 1270 x 1270 x 520 mm envelope and weighs about 30 kg. The Kingfisher M100 platform is not the fastest vehicle, with a maximum speed of 1.3 meters per second. Physically it consists of four hard-shelled, plastic flotation devices (pontoons) held together by a metal main frame [45]. Its size allows for a 5.7 kg payload. Figure A.1 gives a tour of important Kingfisher M100 components. Two waterproof boxes are mounted on the starboard and port sides of the vehicle. The primary thruster module consists of a 12V battery, a control system, a FitPC2, a GPS module, and a motor mounted below. The secondary thruster module houses a 12 battery, a control system, a VIP2400 series wireless radio, and motor mounted below. These two boxes are connected with each other using waterproof Ethernet network cable and power cables. Figure A.2(a) and A.2(b) provides an overview of the insides of the primary and secondary thruster modules respectively.

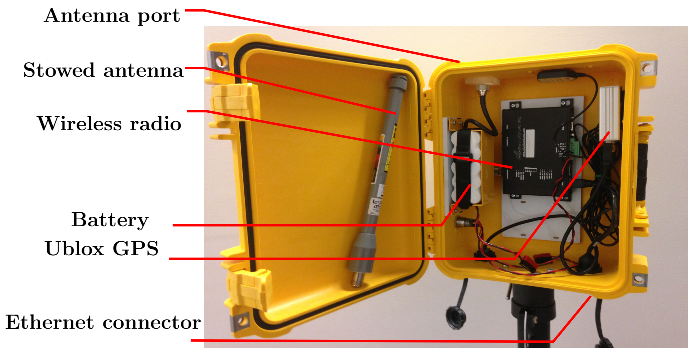
A mobile base station provides connectivity to off-board computation and storage. Figure A.2(c) gives an overview of the Clearpath Robotics mobile base station. This station consist of a battery, a GPS module, and a VIP2400 series wireless radio. Key specifications of the Kingfisher



(a)



(b)



(c)

FIGURE A.2: (a) Kingfisher M100 primary thruster module. (b) M100 Kingfisher secondary thruster module. (c) Base station.

Dimensions	1270x1270x520 mm (length/width/height)
Weight	30Kg
Payload	5.7 Kg
Speed(max)	1.3 m/s forward and 0.7 m/s reverse
Thrust	270 N
Operating time	4 hours typical
Battery	2x 12V 14 Ah NiMH
System interface	RS-232 Serial, 115200 baud
Radio interface	2.4 GHz, 54 mbps broadcast

TABLE A.1: Eddy system specifications. Data taken from [45].

M100 are shown in Table A.1.

Propellers. The Kingfisher uses two propellers for thrust. Each two-blade propeller has a diameter of 23 cm and is driven by a DC servo motor. Each blade is structured as an air foil and allows water to wrap around the leading edge in a uniform streamline.

Control mode. There are two micro-controllers units (MCU) onboard the Kingfisher to control the propellers (See Figure A.3). These control units are pre-programmed to move both of the

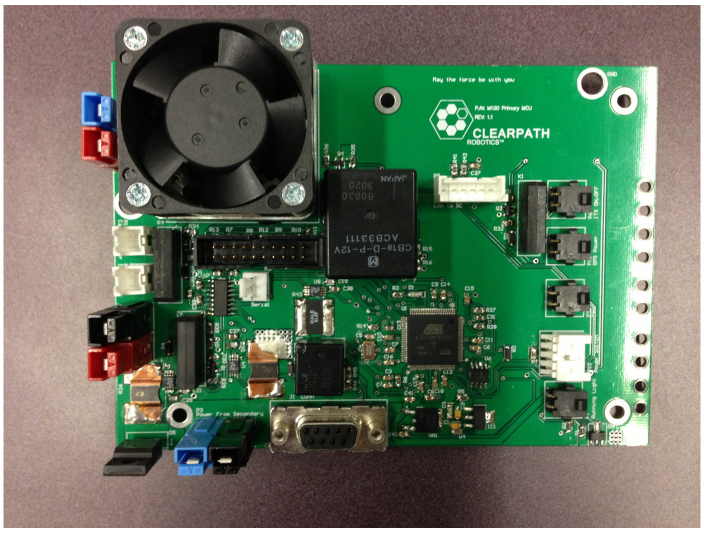


FIGURE A.3: Kingfisher primary Microcontroller.

motors in unison for any action. At the controller level, the Kingfisher provides a direct control over the PWM output to the motors that drive the two propellers. Voltage control allows the direct control of motor voltage, specified as a percentage of 12V.

Computation. The onboard FitPC2 [5] communicates with the micro-controller using the Clearpath Control Protocol (CCP) over RS-232 [45]. FitPC2 is a miniature fan-less PC based on an Atom CPU. It has a small form factor and is extremely energy efficient. The onboard computer runs Ubuntu Linux, and the Robot Operating System (ROS) nodes provide basic functional control of the robot. Speed or position control is implemented at the PC controller level, using GPS, compass, and other sensor data which are connected to the FitPC2 via USB.

Sensors. The Kingfisher platform as shipped is equipped with a differential GPS system and a tilt-compensated compass.

- **GPS.** Eddy is equipped with two GPS receivers. The local base station is a *u-Blox LEA-6T* receiver in an EVK-6T evaluation kit [10]. The Eddy receiver is also a *u-Blox LEA-6T* that is connected to FitPC2 onboard computer. Since the local base station position needs to be known before it can be used as a reference station, the base station needs to be set up on the shore about 30-40 minutes prior to operation to allow the static solution of the base station's position converge. Once the local base station position is determined, it can be used as the reference station for a DGPS solution for the rover receiver mounted on the Eddy.

- **Compass.** An *OceanServer OS5000 Tilt Compensated 3 Axis Digital Compass Kit* [9] is connected to the onboard computer via USB. The OS5000 Compass is a small form factor (1" x 1") three axis, tilt compensated digital compass. The compass is connected via USB and provides precise heading, roll and pitch data ideal for rapid attitude measurement.

A.2 Kingfisher customizations

In order for the Kingfisher to act as a master robot in a heterogeneous robot collective, it was desirable to increase the robot's onboard computational and sensing abilities. This involved embedding an additional PC laptop onboard the robot and providing the robot with a panoramic camera sensor.

A.2.1 Additional computation

A Lenovo laptop running Ubuntu Linux and ROS is mounted in a water-tight case in the middle of the Kingfisher's main frame to provide additional onboard computation. The onboard computer network was modified to accommodate this additional computer and to link this computer with the existing computers on the Kingfisher. As shipped, the Kingfisher is equipped with a 2.4 GHz wireless radio modem to communicate with its base station equipment. An additional wireless router is used to join the onboard FitPC2, the wireless radio modem, and the additional onboard PC laptop to the same network through waterproof Ethernet network cables. Figure A.4 gives an overview of the resulting network infrastructure.

A.2.2 Panoramic camera sensor

Since Eddy is a main element of the fleet of vessels used in this thesis and a visual localization method is employed to localize the other elements of the fleet (the minnows), a panoramic camera system is mounted on Eddy to monitor a 360 degree horizontal viewing angle and to track and localize the other robots around Eddy. Eight *DCS-930L-DLink* IP cameras [13] are integrated in a circle unit to provide a 360 degree horizontal angle of view. Each camera's 640x480 resolution provides 45.3 and 34.5 degree horizontal and vertical viewing angle respectively. Due to mounting difficulties, this ring of cameras may not provide a complete 360 degree view – there exist some regions without overlap, and no effort has been made here to correct the intensities

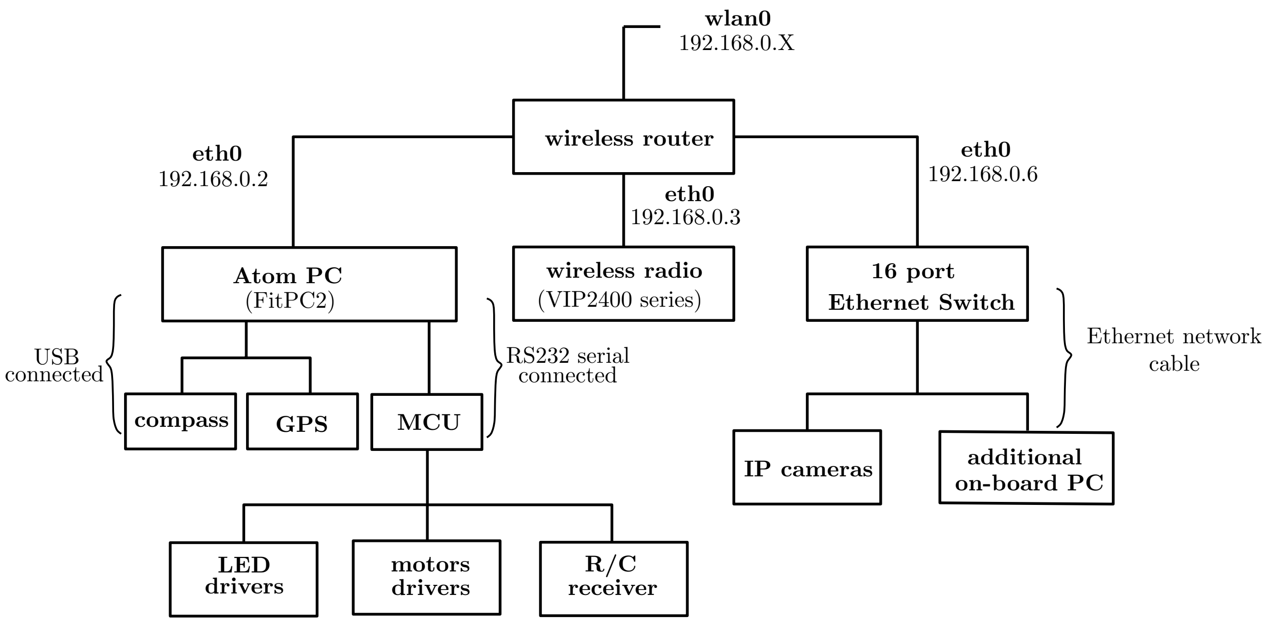


FIGURE A.4: Eddy architecture.

returned from the individual cameras. The images from the IP cameras are transmitted through the ethernet network cable, and processed on the onboard PC laptop.

Each of the eight image sensors has its own independent coordinate frame. Each of these coordinate frames follows standard camera coordinate conventions in that:

- The origin of the coordinate frame is the optical centre of the sensor
- The z-axis points out of the sensor towards the scene
- The x-axis points to the right
- The y-axis points down
- The x and y axes correspond to the image u and v axes where (0,0) is the upper-left corner of the image with the u axis pointing to the right and the v axis pointing down.

The Figure A.5 shows the sensor coordinates as they are oriented on a camera unit.

Figure A.6 shows an overview of the camera system. Figure 3.13 represents the eight frames from the image stream broadcast from the robot's cameras as it operates in the Stong Pond at York university, Canada.

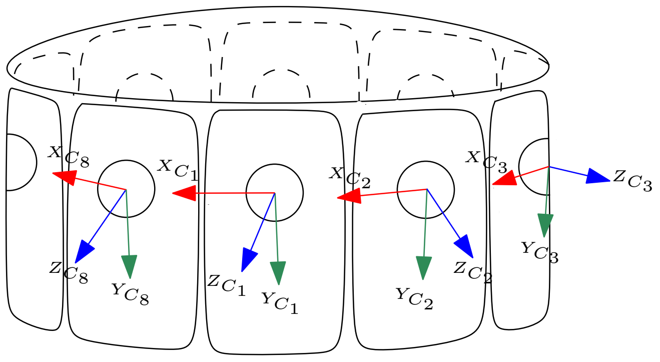
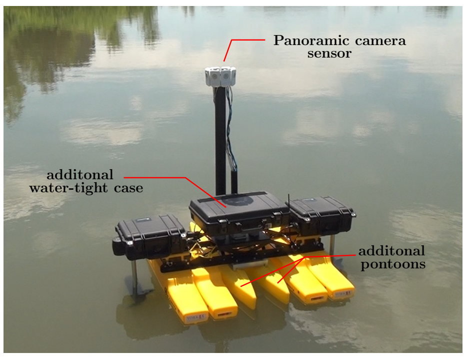
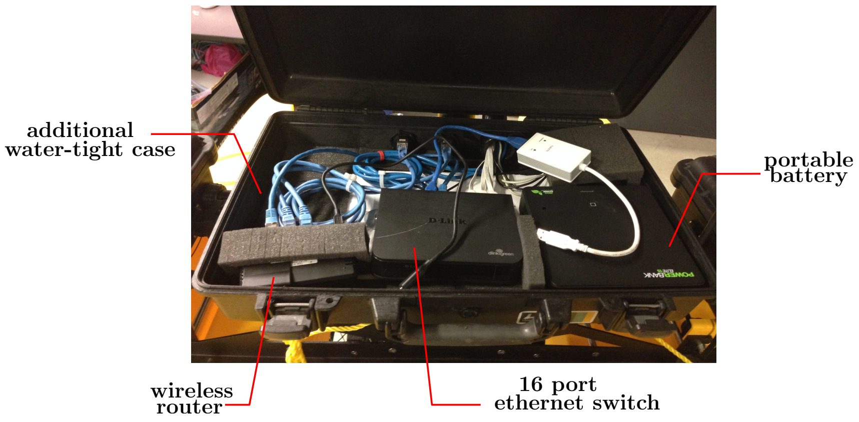


FIGURE A.5: Omnidirectional sensor coordinates.



(a)

(b)



(c)

FIGURE A.6: Omnidirectional camera system. (a) Mounted camera system. (b) Additional onboard PC running Ubuntu and ROS. (c) Camera system components.

A.2.3 Additional pontoons

Two additional pontoons were installed on the shipped Kingfisher to provide required buoyancy to carry the extra mass of a watertight case, additional computer, Ethernet switch, router, camera stand, and camera unit (see Figure A.6(a)).

ROS packages	Functionality
`clearpath_base`	sets the raw motor outputs on the platform subscribing to a *GeometryMsgs::Twist* topic
`clearpath_teleop`	consumes `Joy` topices, applies scale factors and produces the corresponding `Twist` topic
`joy`	interfaces a generic Linux joystick to ROS an publishes a `Joy` topic

Table A.2: Main ROS packages to control Eddy.

A.3 Software control

There are two onboard computers running on Eddy to control the robot: the FitPC2 running Ubuntu 11.10 and ROS Electric and the onboard PC laptop running Ubuntu 12.04.1 LTS (Precise Pangolin) and ROS groovy.

Main ROS packages on Eddy. Several ROS packages define the structure of the basic software control of Eddy. These are summarized in Table A.2.

Localizing Eddy using DGPS and a tilt-compensated compass. To control the movement of Eddy from one place to another, the position and direction of the vessel must be determined. In this thesis, a differential GPS and a digital compass are employed to provide latitude/ longitude and orientation respectively. DGPS is an enhancement to the Global Positioning System that provides improved location accuracy. The general idea behind DGPS is that for two receivers placed relatively close to each other, the signals from the satellites will pass through similar sections of the atmosphere. They will both be affected by the same variability in the satellite ephemeris and clock data. By recording data at a known fixed location known as a *base station* and comparing it with the data recorded at the unknown location known as the *rover*, the effect of these different sources of error can be mostly eliminated, greatly improving accuracy [8].

To implement a DGPS, `RTKLIB` [22] is used which is an open source software library developed by Tomoji Takasu and Akio Yasuda of the Tokyo University of Marine Science and Technology for GPS processing. The library comes with command line and GUI programs for a variety of real-time and post-processing GPS tasks. It provides the following solution types: Single, Fixed, DGPS, Kinematic, Static, and so forth. A ROS wrapper for this library (`RTKLIB ROS`) [23] is designed such that two different node types can be launched via ROS launch files. The first is a `measurement stream` node. This node will read data from an input stream, such as a serial

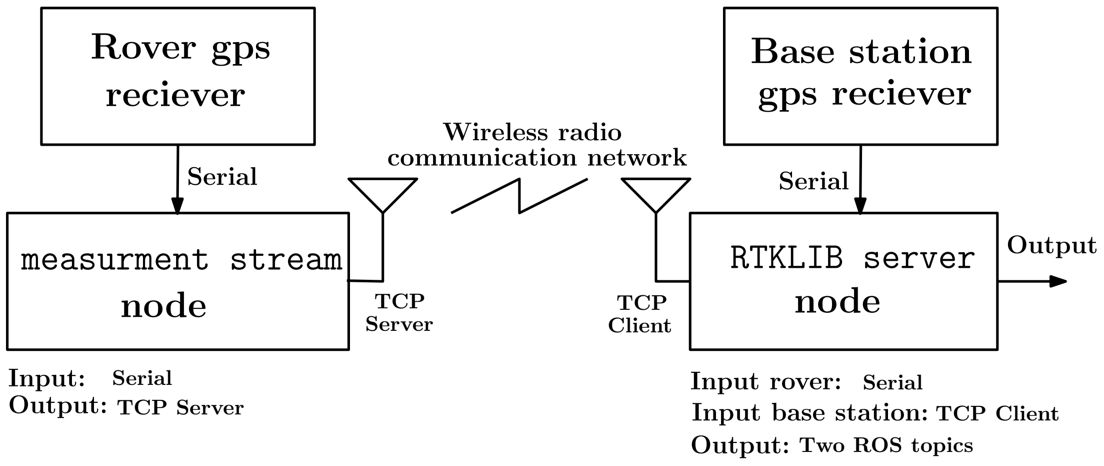


FIGURE A.7: Eddy DGPS network diagram. Inputs are both rover data from a remote receiver and base station data from s serial port. The output is two ROS topics that give the position of rover.

port, and send it to an output stream, such as a tcp socket. This is useful for streaming data from one processor connected to a receiver to another on which the `RTKLIB server` is running to compute the GPS solution. For example, this node can stream data from the receiver on board of Eddy to a base station for processing. The second node type is the `RTKLIB server`. The server receives data streams from both the *rover* and the *base station* and computes the DGPS solution. Solution parameters can be loaded on launch via a *.conf* file or a *.yaml* file. Finally, the `RTKLIB server` node publishes the solution via two ROS topics: `/baseline` (gives the position of the rover relative to the base station), and `/latlon` (gives the exact latitude, longitude, and height of the rover). Figure A.7 represents the DGPS system used for Eddy.

A tilt-compensated compass is used to determine the heading of Eddy (i.e. the direction of "true North"). However, the compass reading must be corrected for two effects. The first is magnetic declination, the angular difference between magnetic North (the local direction of the Earth's magnetic field) and true North. The amount to add to the magnetic compass reading (magnetic declination) in our testing location (North York, Canada) is approximately -10°. The second is magnetic deviation, the angular difference between magnetic North and the compass needle due to nearby sources of iron. Such errors are removed from the reading by calibrating the compass. The `os5000` node talks to the `digital Ocean Server 5000 compass` and gets the data from the compass, and then publishes on a *SensorMsgs::Imu* topic. The `Imu` topic is a standard ROS topic representing angles as quaternions. The magnetic declination is also considered in our testing positioning.

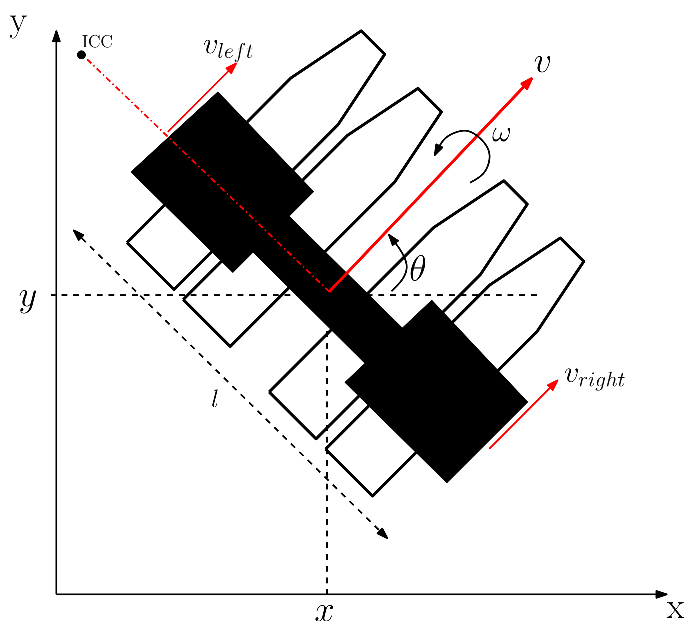


Figure A.8: A differential surface vessel. The boat's centre is located at (x,y) and the bow of the boat makes an angle θ with respect to the x axis. The linear speed of the right and the left motor are (v_{right},v_{left}). The boat has a linear velocity v and angular velocity ω. l is distance between two motors. The Instantaneous Centre of Curvature (ICC) is a point that the robot always rotate about it such that lies somewhere on the common axis of its two motors.

Point to point navigation. In 3-spaces a mobile robot, or vehicle, has 6 degrees of freedom (DOF) expressed by the pose $(x, y, z, Roll, Pitch, Yaw)$. For a robot on a two dimensional surface, the 2D pose (x, y, θ), where θ denotes the heading, is sufficient to describe its motion. The Kingfisher platform is an autonomous surface vessel with *differential drive* whose position is defined by three coordinates in a plane (x, y, θ). Figure A.8 shows Eddy, where (x, y) is the motion centre of the autonomous robot and the gravity centre of the platform. v and ω are the linear and angular velocity of the robot respectively. The linear speed of the right and the left motor are depicted by (v_{right},v_{left}).

The linear and angular speed of the robot can be calculated as Equations A.1 and A.2:

$$v = (v_{right} + v_{left})/2 \tag{A.1}$$

$$\omega = d\theta/dt = (v_{right} - v_{left})/l \tag{A.2}$$

The robot's coordinates (x, y) and its orientation θ change with respect to time and can be calculated using the following equation:

$$\begin{cases} dx/dt = \dot{x} = v\cos(\theta) \\ dy/dt = \dot{y} = v\sin(\theta) \\ \dot{\theta} = \omega \end{cases} \tag{A.3}$$

From the Equation A.3, the kinematic model of the mobile robot with two independently driving motors can be represented in cartesian model as Equation A.4.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{A.4}$$

where, (x, y) and θ describes the configuration (position and orientation) of the centre of the axis of the vehicle. v and ω are the linear and angular velocity of the robot respectively.

Equations A.1, A.2, and A.4 can be combined:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} (v_{right} + v_{left})/2 \\ (v_{right} - v_{left})/l \end{bmatrix} \tag{A.5}$$

A number of special cases are of interest. If $v_{right} = v_{left} = v$, Eddy's motion simplified to Equation A.6. Eddy moves in a straight line (either forward/reverse) and the twist linear component is (v,0,0) with a twist angular component of (0,0,0).

$$\begin{bmatrix} \acute{x} \\ \acute{y} \\ \acute{\theta} \end{bmatrix} = \begin{bmatrix} x + v\cos(\theta)\delta(t) \\ y + v\sin(\theta)\delta(t) \\ \theta \end{bmatrix} \tag{A.6}$$

where $(x, y, \theta)^T$ is the Eddy's pose at time t and $(\acute{x}, \acute{y}, \acute{\theta})^T$ is the Eddy's pose at time $t + \delta(t)$.

If $v_{right} = -v_{left} = v$, Eddy's motion is simplified to Equation A.7. Eddy rotates in place and the linear component is (0,0,0) while the angular component is (0,0,ω) [49].

$$\begin{bmatrix} \acute{x} \\ \acute{y} \\ \acute{\theta} \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + \omega\delta(t) \end{bmatrix} \tag{A.7}$$

Control of the Eddy robot is accomplished by providing the robot a sequence of waypoints $wp = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$ given a starting state (x, y, θ) for the robot. Driving the robot through these waypoints involves choosing $v(t)$ and $\omega(t)$ in order to make this happen. Here we assume that the robot has access to $(x, y, \theta)(t)$ of the vehicle through DGPS and compass data. Calculating the distance between two coordinates and the angle from one point to another based on a fixed coordinate system is a standard geometry problem. To drive Eddy to some waypoint, the following algorithm can be used to reach any target waypoint pose from any starting pose:

1 Spin in place until the robot's orientation coincides with the line from the starting position to the target position: $v_{right} = -v_{left}$.

2 Drive the robot forward until the position coincides with the target waypoint position: $v_{right} = v_{left}$. In this step, the robot frequently stops to correct its heading.

Localizing the minnow's from eddy. The system is explained in Chapter 3.3 completely. See Figures 3.13 and 3.21. You can find all Eddy's localization ros nodes in Table A.3.

Full functionality on Eddy is provided through a set of ROS nodes. These nodes are summarized in Table A.3.

A.4 Summary

This appendix describes the basic Kingfisher design and the modifications to the basic Kingfisher in order to act as a mother robot in a heterogeneous robot collective. As shipped, the Kingfisher is equipped with a DGPS and a compass as well as a long range radio transmitter channel. In order to increase the robot's onboard computational, networking, and sensing abilities, the Kingfisher was modified by mounting an additional PC laptop onboard the robot, an extra wireless router, and providing the robot with 360 degree video sensors. All the sensors provide

Category	**ROS node**	**Location**	**Descriptions**
Low level interface	`clearpath_base`	onboard (FitPC2)	serial interface for Clearpath Robotics
Manual control (joystick)	`clearpath_teleop`	onboard (FitPC2)	contains basic nodes used to quickly launch Clearpath Robotics platform with a joystick control
	`joy_node`	offboard	interfaces a generic Linux joystick to ROS
Manual control	`eddyctl`	offboard	basic data logging and user interface
Localization	`str2str (measurement stream)`	onboard (FitPC2)	read data from onboard gps as a serial port and send it to an output stream as a tcp socket
	`rtkrcv (RTKLIB server)`	offboard	receives data streams from both *rover* and *base station* and computes the DGPS solution
	`os5000`	onboard (FitPC2)	talks to the `digital Ocean Server 5000 compass` and gets the data from the compass, and then publishes on a `Imu` topic
	`converter`	onboard (FitPC2)	converts Quaternion orientation to Eulerian orientation
	`eddy_pose_publisher`	onboard (FitPC2)	provide pose of Eddy using DGPS and compass data
Point-to-point navigation	`eddy_waypoint_following`	onboard (Lenovo)	controlling $v(t)$ and $\omega(t)$ in order to drive Eddy through a sequence of waypoints, spun in place to head to the target, then driven forward to reach target waypoint
	`eddy_odom_publisher`	onboard (Lenovo)	provide pose of Eddy based on odometry data
	`eddy_waypoints`	offboard	provides a sequence of waypoints
	`eddygpsctl`	offboard	controlling $v(t)$ and $\omega(t)$ in order to drive Eddy through saved gps points
Camera system	`axis_camera`	onboard (Lenovo)	accessing an IP camera's MJPG stream and capture them
	`Calibration_homography`	onboard (Lenovo)	calibrate each camera and save homography matrix between camera and world planes
	`homography_publisher`	onboard (Lenovo)	publish all saved homography matrixes
Minnow localization	`child_localizing`	onboard (Lenovo)	minnow localizing using colour detection and calibration data

Table A.3: Eddy ROS nodes functionality.

the necessary onboard sensing capabilities to enable point-to-point control of the vehicle. A ROS-based control and sensing infrastructure is used to control the Kingfisher.

Appendix B

The Minnow: a small-scale autonomous surface vessel

This appendix describes the design and construction of a small-scale autonomous surface vessel, known as a minnow, that makes up the elements of the fleet of vessels used in this thesis. These devices are designed to be small and self-contained autonomous surface vessels (ASV) that operate as part of a larger fleet. Each robot is self-contained for power and computation and communicates with other robots via a standard wifi network.

B.1 Vessel design

A basic design question in the development of any autonomous system is the size of the device. Size, measured in volume or mass, provides a range of constraints related to power, locomotive strategies, onboard sensor capabilities, onboard computation, communication opportunities and the like. This constraint of size is especially true in the development of a small-scale autonomous surface vessel. In order to meet cost and design timeline constraints, the minnow robot platform repurposes a standard radio controlled (RC) hobby boat design. RC vessels are typically small, designed to be easily portable and with relatively short operational periods (typically under an hour, and often significantly less). RC vessels typically avoid regulatory issues associated with operating a “human sized” vessel autonomously. One critical question in terms of the repurposing of RC vessels is the nature of the vessel power plant. RC watercraft are typically developed with electrical systems for onboard control and communication, and are available

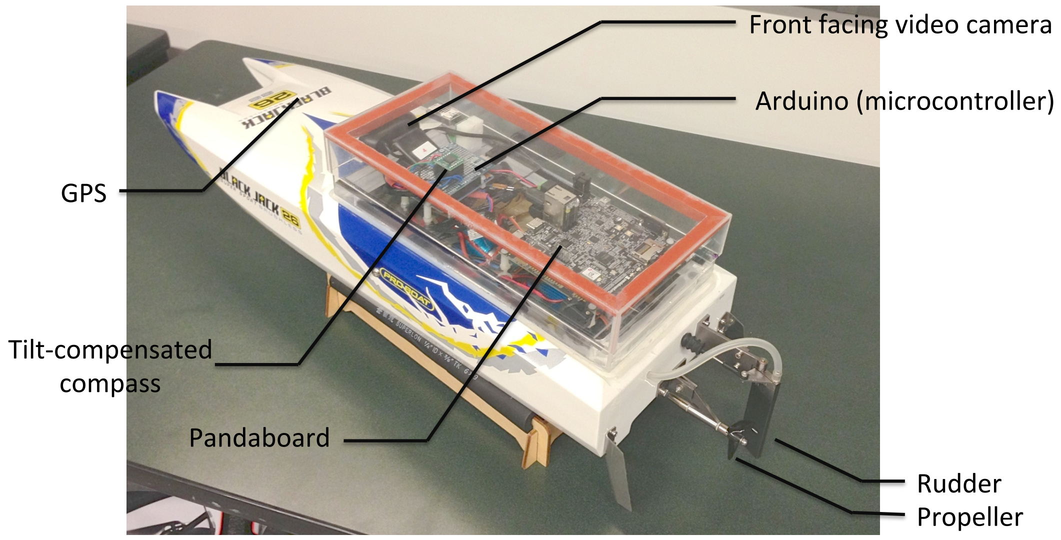


Figure B.1: The minnow at a glance.

with electric, gasoline and nitro drive systems. Each of these drive alternatives come with their own capabilities and constraints. Electrical powered systems are emission free and thus are easily deployed in indoor swimming pools, but they lack the extended range and maximum power output associated with gasoline and nitro power plants. (A properly tuned nitro boat can exceed 80 kilometres an hour in terms of speed.) Fortunately the basic control mechanisms remain unchanged over the various power plants, but the choice of plant type has a significant impact on maximum speed, available test environments and operational period.

B.1.1 Hardware design

For this project the “Blackjack 26” RC boat was chosen as the underling hardware platform. The “Blackjack 26” is a “ready to run” 26” fibreglass catamaran RC vessel powered by a single propeller and equipped with a single rudder (see Figure B.1). The vessel is powered by a 1500 RPM brushless DC motor and utilizes a standard RC servo motor to provide positional control over the rudder. The boat comes equipped with a water-cooled 45A programmable Electronic Speed Controller (ESC) that provides control over the drive motor, power to the servomotor, and exposes itself to control circuitry as a standard servomotor. As shipped, the vessel contains a pair of 7.2V sub-C battery packs that provide power for both the motors and the associated electronics. Normally the vessel is controlled by a remote radio controller, but this device was removed prior to re-purposing the vehicle.

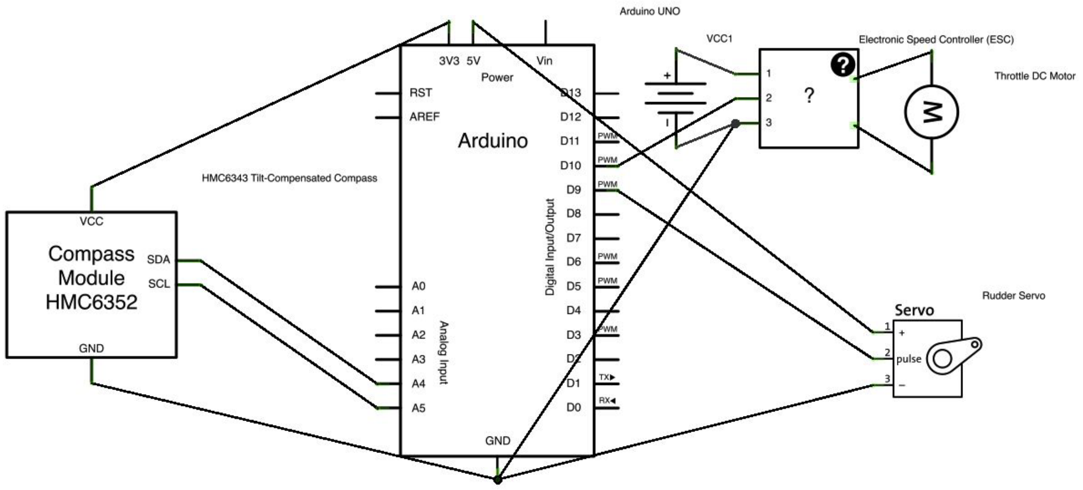


FIGURE B.2: Wiring diagram.

The physical structure of the vessels have been modified only in a very minor way through the addition of a water-tight housing for the added electronics and sensors. Figure B.1 gives a tour of important components of the device.

Electromechanical issues. A surface vessel like the Blackjack 26 is essentially a two degree-of-freedom (DOF) device. There is a rudder and throttle, both of which are controlled by 5V signal lines. The necessary control signal to these two inputs is provided by a radio receiver in the stock RC boat. A key step in re-purposing the vessel as an ASV is providing computer control of these two degrees of freedom. An Arduino microcontroller [3] was chosen to provide this interface. The Arduino (μC) was chosen for a number of reasons related to ease of use, ability to provide the necessary control interface to the underlying hardware, and also the existence of an Arduino ROS package [4] that allows ROS nodes to be executed on the Arduino directly and to be made visible to standard ROS nodes operating elsewhere.

The electrical control subsystem is sketched in Figure B.2. The Arduino micro-controller provides an interface to a tilt-compensated compass and provides reference voltages and PWM control signals to the rudder servo and the electronic speed control (ESC) controlling the throttle. The Arduino also provides power to the rudder servo. Normal power from the drive system provides power to the ESC. The Arduino is connected via USB to an onboard PandaBoard which provides higher-level control as well as providing power to the Arduino. Each of the motor controllers on-board the robot requires a reference ground and a 5V control line. These are easily mapped to the Arduino's digital out and reference lines as shown in Figure B.2.

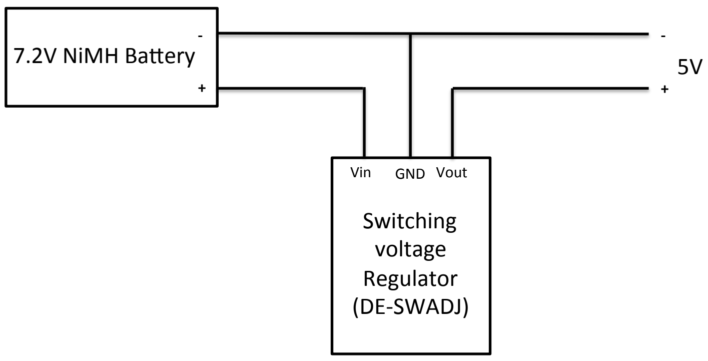


FIGURE B.3: PandaBoard power.

Computation. The limited volume and mass characteristics of the "Blackjack 26" places extreme constraints in terms of potential onboard computation. Furthermore, the decision to build a ROS-compatible device requires that the onboard computer run some version of Linux. Although a number of small form factor devices meet all of these requirements, the decision was made to proceed with a PandaBoard platform [6]. The benefits of using a PandaBoard have been demonstrated against several other small form factor boards. The PandaBoard runs a quasi-standard version of Linux, provides a reasonable number of input and output ports, including on-board video and wifi, and is reasonably inexpensive. Furthermore, it supports a number of different power inputs, including being able to be powered by a USB power source. To power the PandaBoard, a 7.2V rechargeable battery pack and a switching voltage regulator (DE SWADJ) are used to provide 5V power (see Figure B.3).

The PanadaBoard was configured to run Ubuntu Linux and the Robot Operating System (ROS)[94]. When on, the PandaBoard serves as a link between the base station computer (when instructions are sent to, or information is requested from the robot) and the low level systems of the robot through an 802.11g connection. The control of the robots single servo, electronic speed controller, and compass [2] is delegated to the Arduino Uno [3] which is connected to the PandaBoard via USB. Connected directly to the PandaBoard are an on-board camera and a GPS module commands to alter the motion of the robot are either generated on the Pandaboard or received by the PandaBoard and directed to the Arduino. Figure B.4 gives an overview of this structure.

Sensors. Given the experimental nature of the vessel being constructed a wide range of different sensors have been built into the vehicle including GPS, video and a tilt-compensated compass.

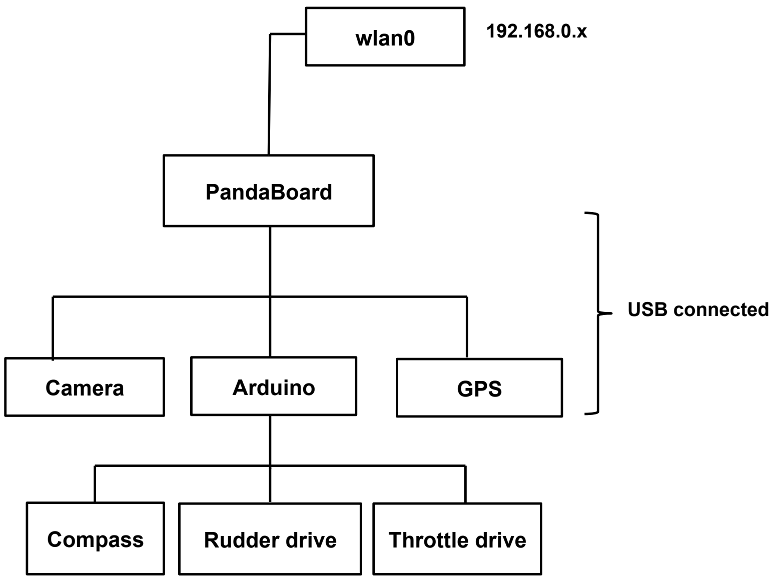


FIGURE B.4: Minnow architecture.

- **GPS.** *BU-353 USB GPS Navigation Receiver* connects to the PandaBoard through a usb connection which propagates GPS data in NMEA format to the PandaBoard.
- **Front facing video camera.** A USB *Logitech HD webcam C270* with 1280x720 resolution camera provides images up to 3.0Mpixel at 5fps and is UVC compliant (allowing for simple interfacing within the Ubuntu environment).
- **Tilt-compensated compass.** A tilt-compensated *3D HMC6343 compass* was used on the minnow. Although a tilt-compensated compass is more expensive than the non-tilt-compensated counterpart, a tilt-compensated compass is essential on a surface vessel where wave action will ensure that the vehicle is rarely horizontal. Few tilt-compensated compasses exist with a USB interface, and thus it was necessary to decode the signal from the device in order to capture the vessel pitch, roll and yaw. Given the existence of the Arduino onboard the vessel to control the actuators, the Arduino was purposed to monitor the compass as well.

B.1.2 Software design

As much as possible, the software developed for the vehicle leverages existing ROS infrastructure rather than attempting to develop a custom software environment for the vehicle. This choice enabled the project to leverage a significant open source code base and lead to a number of

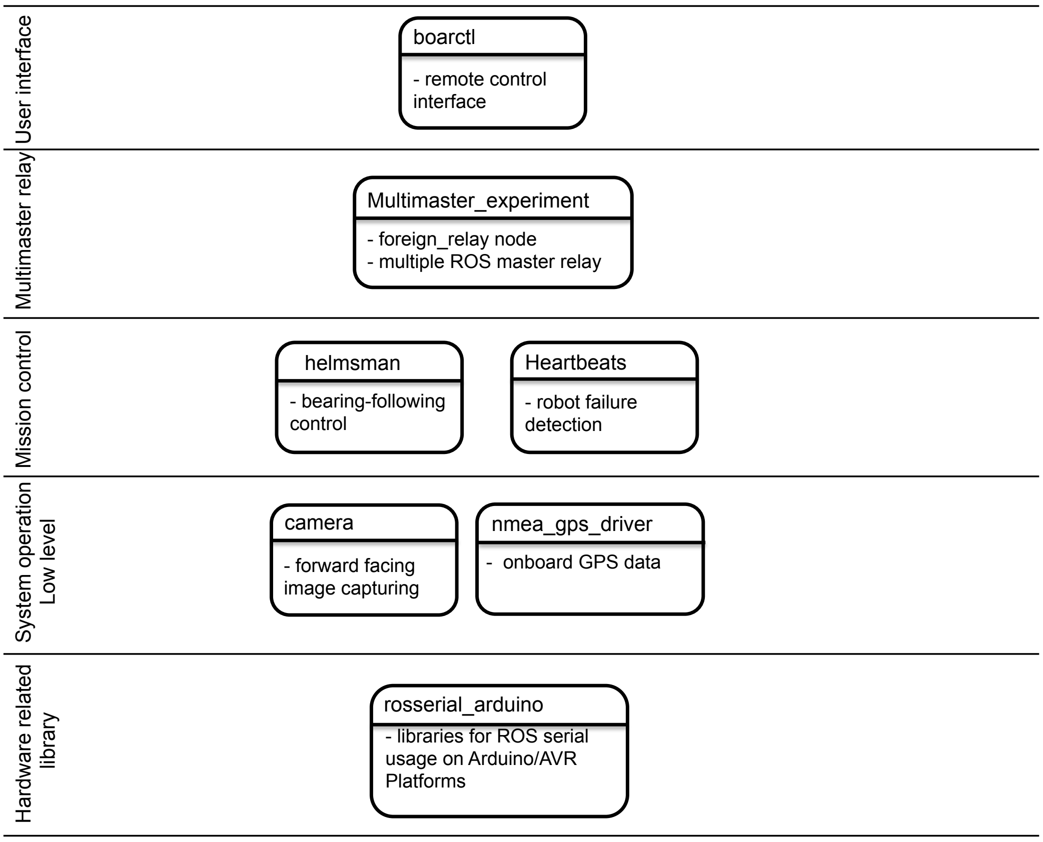

User interface
boarctl
- remote control interface
Multimaster relay
Multimaster_experiment
- foreign_relay node
- multiple ROS master relay
Mission control
helmsman
- bearing-following control
Heartbeats
- robot failure detection
System operation Low level
camera
- forward facing image capturing
nmea_gps_driver
- onboard GPS data
Hardware related library
rosserial_arduino
- libraries for ROS serial usage on Arduino/AVR Platforms

Figure B.5: ROS packages for operating the minnow.

design decisions in terms of hardware and software infrastructure. The process of configuring a PandaBoard to run ROS was not without its difficulties. ROS Fuerte base and Ubuntu 12.04.1 LTS (Precise Pangolin) are installed on the PandaBoard.

ROS packages on the minnow. Several packages are defined to structure the software for the ASV. These are summarized in Figure B.5 along with an overview of the package and Figure B.6 shows their communication network for simple vessel control. At its lowest level the software infrastructure provides a ROS message that sets the rudder orientation and throttle. The `rosserial_arduino` package [7] allows ROS nodes to operate on the Arduino with direct access to the Arduino control and data lines. This provides a very clean interface between low level vehicle control and the ROS environment. In addition to subscribing to a `setRudder` and `setThrottle` message, the minnow also publishes a `vesselStatus` flag via a custom ROS `Boat.msgs` message which provides low-level information from the Arduino controller. This message contains the current commanded position of the rudder and throttle along with vessel pitch, roll and yaw.

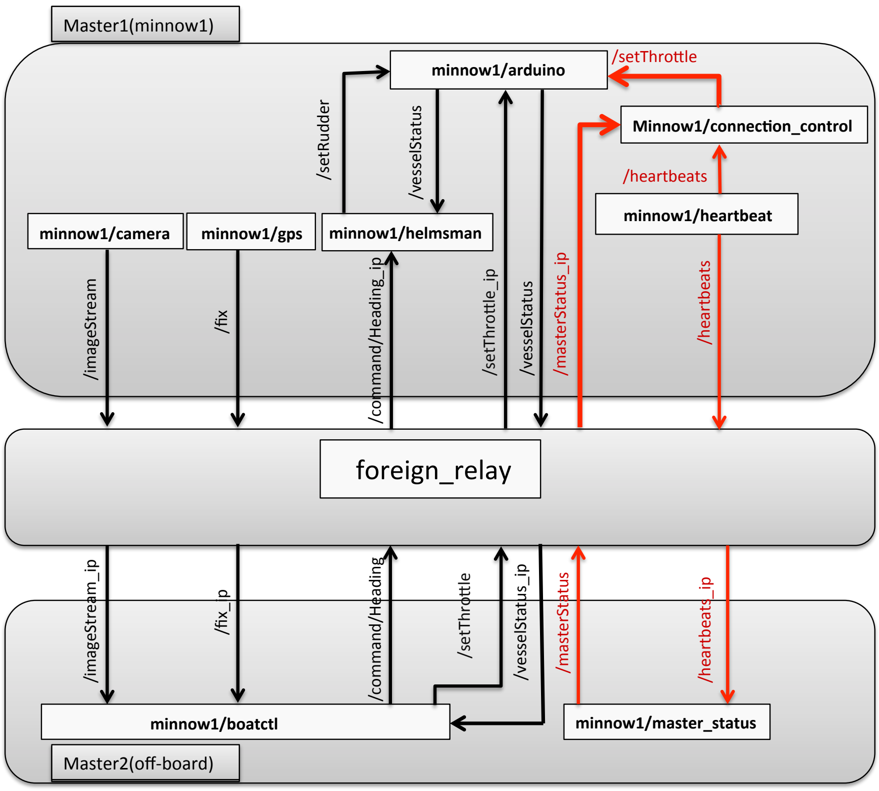


FIGURE B.6: ROS control of the minnow. Here */minnow1* is driven by */boatctl*.

Although the use of the Rosserial package greatly simplifies integrating low-level control of an actuator into a collection of ROS nodes, it is not without its limitations. The existing version of Rosserial_arduino in the library limits the number of publishers and subscribers on the Arduino. Nor are all primitive types supported in messages that can be passed. Furthermore, the limited bandwidth between the Arduino and its host computer places limits on how much, and how quickly, data can be passed from code running on the Arduino and that running on the host device. (For example, sending long string messages is not to be encouraged given the limited 280 byte buffer size available).

A ROS `nmea_gps_driver` package interfaces with the *BU-353 USB GPS Navigation Receiver*, parsing the data stream (in NMEA 0813 V2.2 format). The node parses only the GPS system fix data (*GPGGA*). These strings contain information on the latitude, longitude and altitude as well as associated data on fix quality and time. The parsed latitude / longitude data is converted into a $+/-$ degree with positive values going North (Latitude) and East (Longitude). The latitude and longitude are encapsulated in a custom gps message within the ROS framework. If the GPS fix quality is zero an error latitude / longitude value is returned.

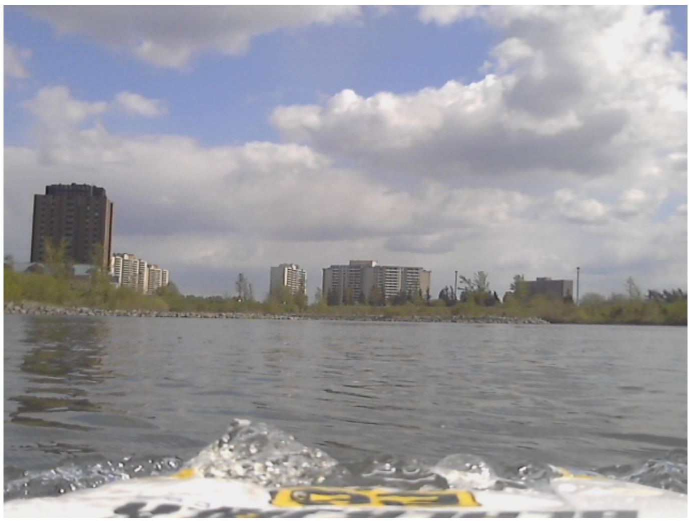


Figure B.7: Streamed digital camera footage from the minnow.

The process of capturing video onboard the robot is accomplished using the standard Video4Linux library [11] and images are transferred as a single ROS sensor message using the `camera` package. Other nodes, including nodes off-board the vehicle can subscribe to this published image stream. Figure B.7 shows a frame from the image stream broadcast from the robot as it operates in the Stong Pond near York university, Canada.

The `helmsman` package implements a standard PID controller to maintain a compass bearing given a commanded heading (specified in a `commandHeading` message) and a given speed. The vessel can turn more sharply at slower speeds and thus the bearing following controller (see Figure B.9) was tuned differentially for different commanded velocities. This allows for a more effective turning angle to be chosen for slower speeds. Turning was accomplished by clamping the output of the PID to a speed-dependent maximum rudder deflection. Empirically the maximum safe turning angle was determined by tele-operating the vessel at various throttle values and determining the maximum safe turning angle by inspection. Using linear interpolation, the maximum safe turning angle for different throttle values was obtained (see Figure B.8).

Figure B.9 illustrates the general control process, where a command heading is compared with the current heading, the result is multiplied by a constant gain (k_p), and the result is the input to a hard limit function that is established for the safe rudder command output which is depends on vessel speed. The result is an output to the actuators (rudder).

The `boatctl` package provides a python-based graphical user interface to control the vehicle (see Figure B.10). This interface provides the on-board camera view, GPS coordinates, and the

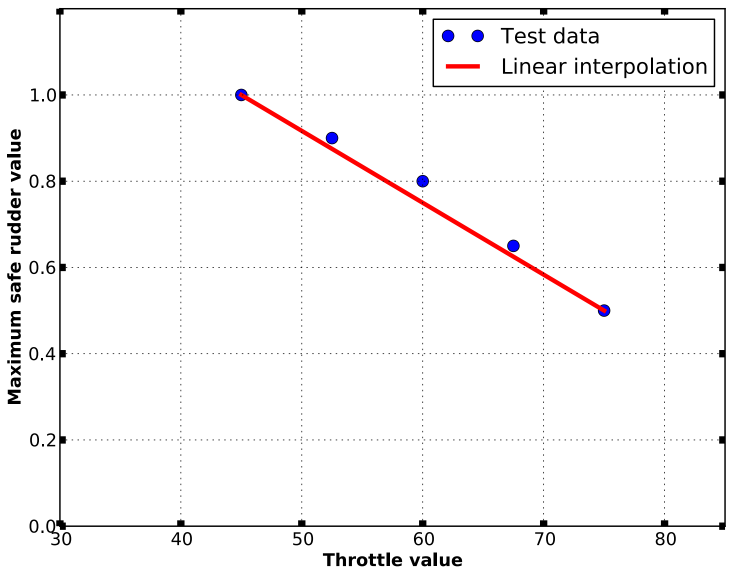


FIGURE B.8: The maximum safe turning value for different throttle values. The maximum safe rudder value is a value between 0 and 1, 0 corresponds no rudder change and 1 is corresponding to the maximum admissible turning rudder angle to one side.

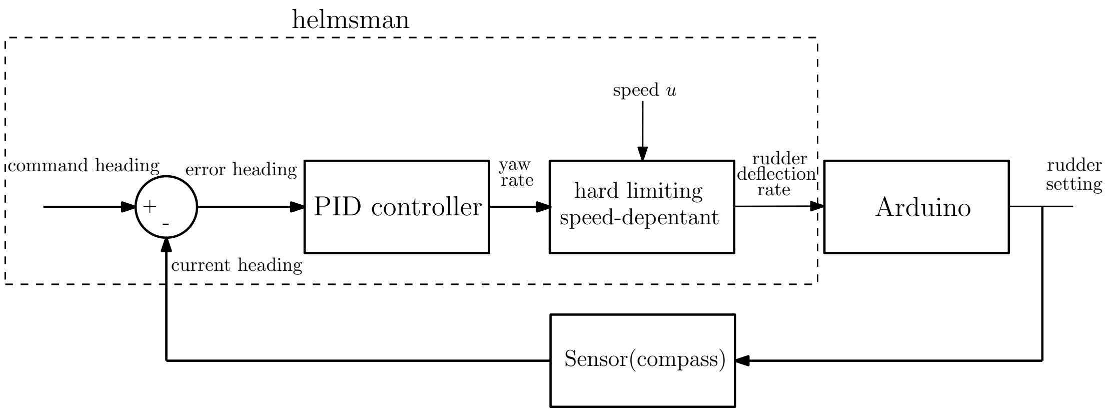
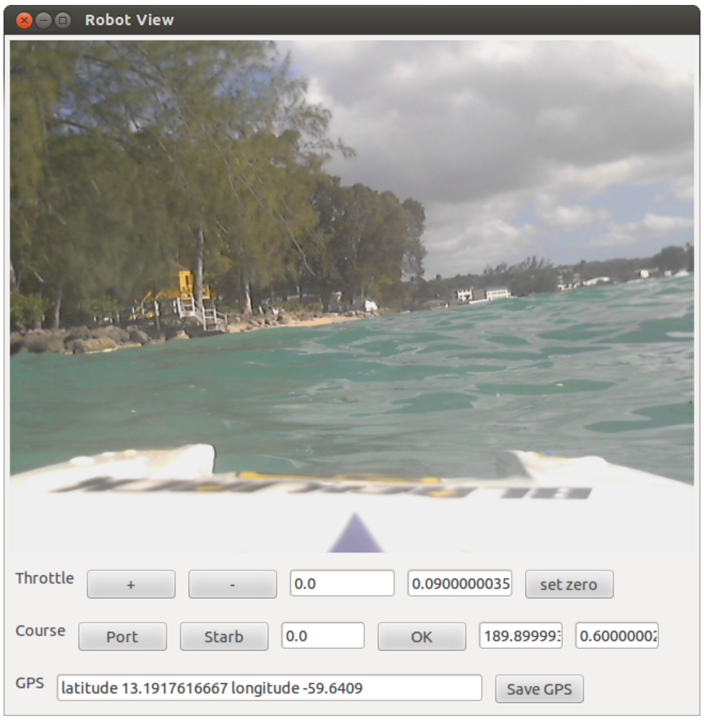


FIGURE B.9: Block diagram of minnow bearing following controller.

speed and heading of the boat. It also provides control of the throttle and heading of the vessel via button and text widgets.

Minnow waypoint following controller. Designing controllers for surface vessels subject to non-holonomic kinematic constraints is challenging. The full model of a surface vessel has 6-DOFs. For a surface robot this model can be simplified to a 3-DOFs model that only reflects surge, sway, and yaw (where surge is the linear longitudinal (front/back) motion and sway is the linear lateral (side-to-side) motion). These can be controlled by a waypoint following controller.

The kinematic model of a surface vessel that has a propeller and a single control surface (rudder)



Throttle 0.0 0.0900000035

Course 0.0 189.89999 0.6000000

GPS latitude 13.1917616667 longitude -59.6409

FIGURE B.10: Minnow interface (controlling the vehicle off of Holetown, Barbados)

is strongly similar to the kinematics model of a tricycle-drive robot. The model is given in Equation B.1 and depicted in Figure B.11 [36].

$$\begin{cases} \dot{x} = u\cos(\theta) \\ \dot{y} = u\sin(\theta) \\ \dot{\theta} = \frac{u}{L/2}\sin(\delta) \end{cases} \tag{B.1}$$

where (x, y) and θ describes the configuration (position and orientation) of the centre of the axis of the vehicle. Vehicle speed is u and the steering direction (rudder angle) is δ.

Control of the minnow robot is accomplished by providing the robot a sequence of waypoints $wp = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$ given a starting state (x, y, θ) for the robot. Driving the robot through these waypoints involves choosing $u(t)$ and $\delta(t)$ in order to make this happen. Here we assume that the robot has access to $(x, y, \theta)(t)$ of the vehicle through some combination of internal (e.g., onboard compass) and external (e.g., state estimation from the mother robot) data. First, the distance and the course that is necessary for the robot to reach the next waypoint are computed. Calculating the distance between two coordinates and the angle from one point to another based on a fixed coordinate system are a standard geometry problem. If the minnow is not heading towards the goal point, drive the robot at some velocity u and

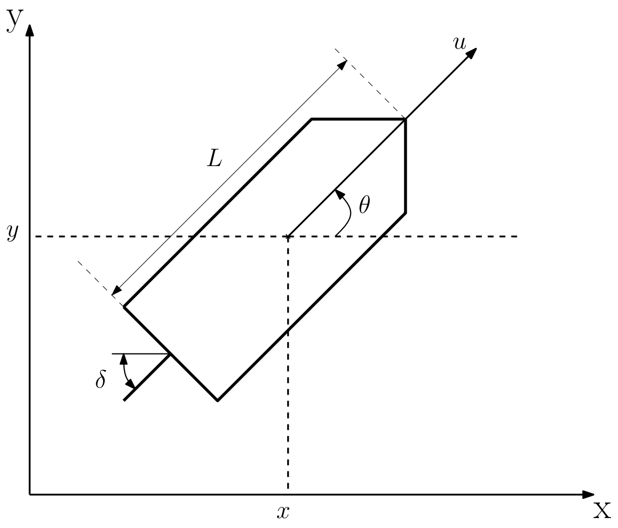


Figure B.11: Non-holonomic surface vessel kinematic model. The boat's centre is located at (x,y) and the bow of the boat makes an angle θ with respect to the x axis. The boat has a forward velocity u. The rudder makes an angle δ with respect to the midline of the boat.

frequently command the desired heading (passing command heading to the bearing controller which is depicted in Figure B.9) until the waypoint is reached. Otherwise, set the robot rudder angle to zero and drive towards the waypoint at some velocity u. When the waypoint is reached, the controller seeks the next waypoint.

Full functionality on the vehicle is provided through a set of ROS nodes. These nodes are summarized in Table B.1.

B.2 Vessel testing

The minnow USVs performed well in multiple pool and pond trials, as well as open water trials off the coast of Holetown, Barbados in January 2012 and January 2013. It has also undergone extensive testing in Stong pond at York university. During these trials several things became apparent. First, the hull was able to take on the extra payload and run quite efficiently. At 20% throttle the boat was able to navigate fairly choppy waters and outpace human swimmers. One issue that we expected to take considerable time during the field trials was tuning this PID controller for the open water conditions. Surprisingly, the tuning of the PID controller was less critical in open water as opposed to a pool or pond setting as the rough surface acts to damp the system making changes in rudder position take time to accumulate into the overall vehicle trajectory. In all, the vehicle was successfully tele-operated visually and commanded to follow

Node	Location	Description
`Arduino`	onboard	encapsulates the rudder and throttle control as well as monitoring the compass
`helmsman`	onboard	implements a standard PID controller to maintain a compass heading given a commanded heading in different speed
`gps`	onboard	encapsulates onboard GPS sensor
`camera`	onboard	encapsulates forward facing camera
`boatctl`	offboard	basic data logging and user interface
`foreign_relay`	onboard	designed for use with multiple robots running separate Masters
`heartbeat`	onboard	periodically send heartbeat messages in order to inform the other agents and server about its aliveness
`connection_control`	onboard	checks its connection to the server, robot goes to recovery mode it it is failed
`master_status`	offboard	publishing the time at which the last heartbeat message from agent is received
`minnow_waypoint_following`	onboard	choosing $u(t)$ and $\delta(t)$ in order to drive the minnow through a sequence of waypoints
`minnow_odometry_publisher`	onboard	provide pose of minnow based on odometry data
`minnow_waypoints`	offboard	provides a sequence of waypoints

Table B.1: Minnow ROS nodes full functionality.

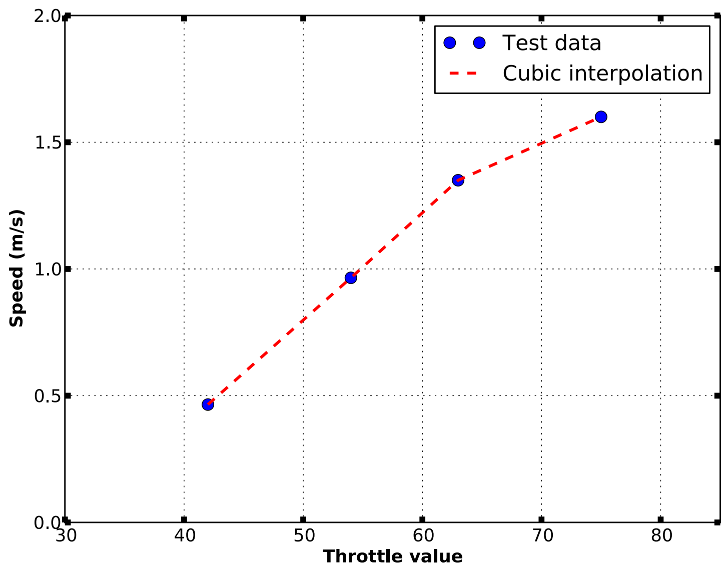


Figure B.12: Minnow speed in different throttle value.

a compass heading during several trials both controlling the rudder position directly as well as controlling the compass heading in different speeds through the PID "Helmsman" controller.

To model the minnow behaviour in simulation, an experiment was run in order to determine the relationship between the different throttle value settings and the resulting speed of the vessel. A set distance was marked and the time it took the vessel to traverse that distance was recorded. From this information, the velocity over that throttle value was calculated. Using spline interpolation, the vessel speed for different throttle values was obtained (see Figure B.12).

B.3 Summary

This appendix described the process of repurposing an inexpensive radio-controlled (RC) electric motorboat as a small scale autonomous surface vessel. Standard electronics components were used to interface with the RC boat electronics, and the vessel was augmented with GPS, vision, and a tilt-compensated compass to provide the necessary onboard sensing capabilities to enable point-to-point and target-based control of the vehicle. A ROS-based control and sensing infrastructure is used to operate the vehicle on-board.