

3-29-2017

Techniques for Adaptive Navigation of Multi-Robot Clusters

Robert McDonald

Santa Clara University

Department of Mechanical Engineering

Date: March 29, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY

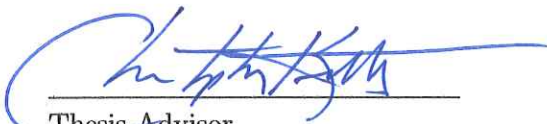
Robert McDonald

ENTITLED

Techniques for Adaptive Navigation of Multi-Robot Clusters

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF

MASTER OF SCIENCE IN MECHANICAL ENGINEERING



Thesis Advisor
Dr. Christopher Kitts



Thesis Reader
Dr. Timothy Hight



Chairman of Department
Dr. Drazen Fabris

Techniques for Adaptive Navigation of Multi-Robot Clusters

By

Robert McDonald

Graduate Thesis

Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Mechanical Engineering
in the School of Engineering at
Santa Clara University, 2017

Santa Clara, California

Techniques for Adaptive Navigation of Multi-Robot Clusters

Robert McDonald

Department of Mechanical Engineering
Santa Clara University
Santa Clara, California
2017

ABSTRACT

Sensor based navigation techniques are those that alter the path being followed based on realtime sensor readings. They are often used in scalar fields, which are regions with a scalar quantity of interest, such as temperature or concentration level, associated with every point in the field. An example of sensor based navigation in a scalar field is using a set of measurements from a distributed group of robots in order to estimate the gradient of a field; driving in the direction of the gradient leads the group to the maximum value in the field, a point that often has practical value. In previous work, researchers in the Santa Clara University Robotic Systems Laboratory have proposed multirobot sensor based controllers to move to maximum points, minimum points, and along field contour lines. This thesis work contributes to this work, presenting work performed with other researchers in the Lab to navigate ridges, trenches, and saddle points. Simulations have verified that these techniques can track field features with a minimal amount of steady state error. The complete set of primitive controllers was then used as a basis for an application controller that used a state machine architecture to switch between the primitive controllers in order to execute complex tasks. Four different applications were demonstrated in simulation using this method: ridge control with recovery, moving from one local maximum to another, mapping the contours around a maximum, and traveling to a designated location while staying above a particular scalar value. These controllers were successfully demonstrated in simulation under a number of ideal conditions. Future work is proposed in order to extend the capabilities of this technique and to improve robustness.

Acknowledgments

First and foremost, I would like to thank my advisor Dr. Christopher Kitts. He has provided endless guidance, support, patience, and inspiration. If not for him I would never have been motivated to pursue robotics research, and none of this would have been possible.

I would also like to thank all of my peers, coworkers, and predecessors in the lab. In particular, I want to acknowledge Thomas Adamek, who played a huge role in pioneering the Adaptive Navigation technique, and Michael Neumann, with his endless supply of both insight and criticism.

I also want to thank my friends and family. Specifically, my friend Dan Ward was always there to provide advice, edits, and to act as living proof that it is in fact possible to finish a Master's thesis. Last but certainly not least, I want to thank my wonderful wife Jill for her love, support, huge amounts of tolerance, and her uncanny ability to make me laugh at myself.

A portion of this work has been submitted for publication [24].

Table of Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Cluster Control | 2 |
| 1.2 | Adaptive Navigation | 3 |
| 1.2.1 | Previous Adaptive Navigation Work | 4 |
| 1.2.2 | Extending the Strategy | 8 |
| 1.3 | Project Statement | 10 |
| 2 | Ridge, Trench, and Saddle Navigation | 11 |
| 2.1 | Cluster Definition and Simulation | 11 |
| 2.2 | Ridge and Trench Navigation | 14 |
| 2.2.1 | Control Strategy | 15 |
| 2.3 | Saddle Point Navigation | 18 |
| 2.4 | Discussion | 21 |
| 3 | Adaptive Field Applications | 22 |
| 3.1 | Overview of Architecture | 22 |
| 3.2 | Ridge Controller with Recovery | 24 |
| 3.3 | From Local Maximum to Local Maximum | 28 |
| 3.4 | Contour Mapping | 34 |
| 3.5 | Maintaining Signal Strength | 38 |
| 3.6 | Discussion | 41 |
| 3.6.1 | Unknown Features | 42 |
| 3.6.2 | Unwanted Repetition | 42 |

| | |
|--|-----------|
| 4 Conclusion | 43 |
| 4.1 Summary | 43 |
| 4.2 Future Work | 44 |
| 4.2.1 Exploration with Memory | 44 |
| 4.2.2 Adaptive Cluster Sizing | 44 |
| 4.2.3 Additional Field Testing | 45 |
| Bibliography | 46 |
| Appendices | A1 |

List of Figures

| | | |
|-----|---|----|
| 1.1 | General cluster space architecture for n robots | 3 |
| 1.2 | A three robot cluster in an equilateral triangle formation | 5 |
| 1.3 | A robot cluster moving in two spatial dimensions, and navigating a scalar surface [11] | 6 |
| 1.4 | The vectors used to calculate the gradient using a three robot cluster [11] | 6 |
| 1.5 | Three simulated clusters of robots, each of which navigates to a different extremum | 7 |
| 1.6 | Three simulated clusters of robots, navigating three different contour lines | 8 |
| 1.7 | Three automated kayaks traversing a scalar field at Stevens Creek Reservoir in Cupertino, CA [11] | 9 |
| 1.8 | Layers of the Adaptive Navigation Strategy, where solid lines represent commands, and dashed lines represent environmental data | 10 |
| 2.1 | The general form of the 5-robot cluster definition | 13 |
| 2.2 | A five robot cluster for ridge and trench navigation | 13 |
| 2.3 | One cluster (in red) navigates down a curved ridge, and another (in blue) goes up a trench to a saddle point. | 14 |
| 2.4 | The same paths from Figure 2.3, but from an overhead contour perspective | 15 |
| 2.5 | A time history plot of the scalar differential between robots 2 and 3, as well as between robots 4 and 5, from the same simulation run as Figures 2.3 and 2.4. | 18 |
| 2.6 | The blue cluster is the same set of robots from Figures 2.3, and 2.4, traveling up a trench to the saddle point, and the green path is coming down a ridge into the same point. | 19 |
| 2.7 | The same paths from Figure 2.6 from a two dimensional overhead perspective | 20 |
| 2.8 | A time history plot of the scalar differentials between robots 2 and 3, 4 and 5, 2 and 4, and 3 and 5, as the cluster travels up a trench to a saddle point | 20 |

| | | |
|------|---|----|
| 3.1 | Layers of the Adaptive Navigation Strategy, where solid lines represent commands, and dashed lines represent environmental data | 23 |
| 3.2 | The nine-robot cluster used for the state level controllers | 24 |
| 3.3 | State diagram depicting the states required for the ridge following controller with a recovery mode | 24 |
| 3.4 | Two robot clusters navigate a scalar surface, one of which seeks and tracks a ridge, the other does the same with a trench | 26 |
| 3.5 | The same robot clusters from Figure 3.4 viewed from above | 27 |
| 3.6 | Time history plots of the states and transition criteria for the red path from Figures 3.4 and 3.5 | 27 |
| 3.7 | State diagram for going from local maximum to local maximum | 28 |
| 3.8 | A robot cluster relative to a contour line (displayed in red). | 30 |
| 3.9 | A robot cluster travels to one local max, then to another | 32 |
| 3.10 | The time response plots for the states and transition criteria throughout the path displayed in Figure 3.9 | 33 |
| 3.11 | State diagram for mapping the contour lines around a peak | 34 |
| 3.12 | A robot cluster mapping contours around a maximum in the field | 36 |
| 3.13 | Contour mapping time history containing state and transition information | 37 |
| 3.14 | A robot cluster gets stuck while mapping contours due to a secondary maximum | 38 |
| 3.15 | State diagram for navigating to a point while maintaining signal strength | 39 |
| 3.16 | A robot cluster navigating to $x = 75$, $y = -75$, using a state controller to remain above a scalar threshold | 40 |
| 3.17 | Time response containing state and transition information for maintaining scalar magnitude | 41 |
| 1 | Simulink block diagram used for all simulations. | A1 |
| 2 | This diagram is a subsection of Figure 4, and calls a '.m' file that changes based upon which controller is in use. | A1 |
| 3 | This Simulink diagram is inside the kinematics block in Figure 4. | A2 |
| 4 | This Simulink diagram is inside the transfer function block in Figure 3. | A2 |

List of Tables

| | | |
|-----|--|----|
| 2.1 | Cluster variable definitions | 12 |
| 3.1 | Summary of Transition Criteria for Ridge Recovery | 25 |
| 3.2 | Summary of Transition Criteria for Extrema Hopping | 28 |
| 3.3 | Summary of Transition Criteria for Contour Mapping | 34 |
| 3.4 | Summary of Transition Criteria for Extrema Hopping | 39 |

CHAPTER 1

Introduction

Robotic systems are an excellent tool for executing any number of tasks, particularly those that would be tedious, difficult, or dangerous for a human to accomplish. It is particularly effective to customize a robotic system for a particular purpose, increasing its speed, precision, and mobility to maximize performance. While single robots can solve a large variety of problems, multi-robot systems can provide improved performance and flexibility for many applications. Some of the advantages of using multiple mobile robots include distributed sensing and actuation, redundancy, and reconfigurability [1].

Multirobot systems are effective in a variety of applications. For example, a team of robots can work together to perform object transportation tasks. In [2] a multirobot system is used to manage the distribution of supplies in a hospital setting. There are also higher precision techniques being developed, like using two robots to move a single unwieldy object with the use of force control [3]. The use of multiple mobile robots is also well suited for any application that involves covering a large amount of area, like in [4], where a system of robots is used to deploy communications relays for network coverage.

Systems of multiple robots are becoming more common in mapping and exploration applications. These techniques use either vision or distributed sensing, have a framework for building maps as the robots progress, and use path planning to reach unexplored regions [5]. An example is mapping an environment via omnidirectional vision localization with 360 degree cameras mounted on multiple robots [6]. There have been variations on this theme, including using topographical mapping strategies instead of traditional grid maps [7]. Once obstacles are introduced, methods like the fuzzy logic approach presented in [8], or the tree-based approaches presented in [9] can be used to avoid them along the way.

In contrast to typical map-generation exploration techniques that try to document an entire region, an adaptive navigation strategy is one that varies the path traveled in

order to take a more direct route to points of interest. These techniques are based upon reactive controllers that are perpetually updating the desired travel bearing based on realtime measurements of the environment. This not only reduces the required mission time and resources, but also makes these techniques far more suitable for transient environments than purely “mow the lawn” mapping. Adaptive navigation decisions are based upon distributed sensor information provided by multiple robots traveling in formation. Because of this, either implicit or explicit formation control is typically required to get accurate information about the field.

This thesis review contributions to expanding the repertoire of adaptive navigation control primitives and explores the sequencing of specific adaptive navigation strategies in order to perform interesting environmentally-dependent tasks. This introductory chapter discusses the previous work in this area by the Robotic Systems Laboratory (RSL) at Santa Clara University. This includes a method for controlling a cluster of multiple robots (section 1.1), and methods for using these clusters to navigate to or along basic features of a scalar field (section 1.2). Both of these categories of work have been verified and implemented in simulation, and in some cases in field environments. It is this body of work that forms the foundation for the research presented in this thesis.

1.1 Cluster Control

The Robotic System Laboratory at Santa Clara University developed the cluster control method for controlling a number of mobile robots at once. This method has been verified in the field, on many different platforms, including rovers [10], marine platforms [11], and airborne robots [12]. Not only has the cluster control method been tested in a variety of environments, field testing has also been conducted for both two [10] and three [12] dimensional environments. This control technique is suited for a wide variety of applications, and has been field tested with a variety of objectives and configurations, including: a set of robots in a guarding configuration [13], a line of communication relay robots arranged to optimize signal strength[14], and the adaptive navigation of scalar fields [11]. Adaptive navigation is one of the cluster space control areas which has large potential for expansion.

Cluster space control treats n robots as part of a virtual, articulating mechanism, with full degree of freedom control [10]. As an operational space control approach, it offers the advantage of specifying the motion of the cluster rather than those of the individual

robots. The cluster space pose is defined by the position and orientation of the cluster, its geometry, and the relative rotation of individual robots. Formal kinematic transforms are used to relate the cluster variables to the individual robot pose variables, and to relate the sensed robot state data back to the cluster space pose data [10]. An inverse Jacobian transform can be used to convert the cluster level velocity commands to the appropriate robot level velocity commands, and a Jacobian transform can be used to transform robot level velocity data back to the cluster space.

It is important to note that as both Jacobian transforms are dependent on the exact cluster configuration, the matrices need to be updated over time to compensate for changes in formation [10]. Figure 1.1 is the general form of the cluster space architecture. Higher level controllers are typically built on top of this structure, including those used for adaptive navigation.

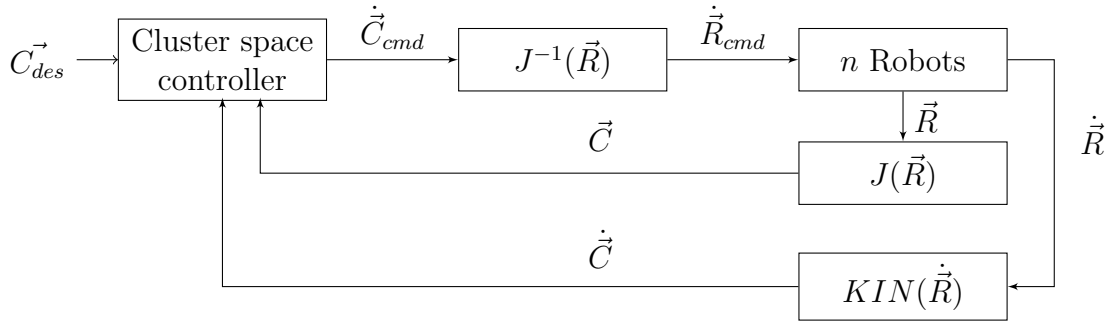


Fig. 1.1: General cluster space architecture for n robots

1.2 Adaptive Navigation

A scalar field is a physical area in which each spatial point has a corresponding scalar value. These scalar values can represent phenomena such as temperature, bathymetry, and many others. The distribution of the field can usually be represented visually by a heat map or virtual surface. Adaptive navigation techniques use scalar readings to generate spatial movement commands, which lead the robots to points of interest in the virtual surface that represents the scalar field.

Planar adaptive navigation of scalar fields has been conducted using a variety of controllers and platforms. For single-robot setups, dithering or circular movement is required to estimate the field gradient in order to navigate to the minimum or maximum

of a scalar field [15] [16]. These methods have the advantage of simplicity, but can be slow due to extraneous movement and perform poorly when tracking dynamic fields. An alternative single robot solution to this problem is a vehicle with a differential array of sensors on board. This method has been used to follow a gas odor in the form of a plume [17].

Using multiple robots to navigate scalar fields provides a number of advantages including the instantaneous computation of field characteristics critical to navigation decisions, reduced maneuver costs, robustness to vehicle failure, and the ability to adjust the formation size to accommodate the particular field of interest [18]. Multirobot methods have been implemented by a number of groups, some of which included lab testing, and others that were strictly in simulation. One of these groups used multiple robots in an experimental setting to navigate to a maxima in a field, however it still required one of the robots to dither spatially to acquire the necessary gradient data [19]. In [20] and [18] a multi-robot adaptive navigation technique was developed, stability was proven, and the gradient navigation was decoupled from the formation keeping function. This work was verified in [21] and [22], where it was simulated using temperature data collected from the Autonomous Ocean Sampling Network in Monterey Bay, CA. However, this work has never been demonstrated experimentally. Overall, nearly all previous work has focused on extrema-finding, little of the work has been experimentally verified, and none of it has been demonstrated in the field.

1.2.1 Previous Adaptive Navigation Work

The adaptive navigation control primitives developed previously by RSL researchers have used gradient based navigation techniques. To facilitate the calculation of the gradient, these control methods make use of three robots in a triangular formation as seen in Figure 1.2 and Figure 1.3. This allows for three scalar measurements, which is enough to compute an estimate of the local gradient. The first step is to use Equations 1.1, and 1.2 to construct two vectors, where x and y are the physical position of a particular bot, and z is the scalar value at that point. Equation 1.3 is then used to create a normal vector, followed by Equation 1.4 to project that vector onto a two dimensional plane, to provide the bearing of the gradient (b_{grad}) [11]. The relationships between these vectors is displayed in Figure 1.4.

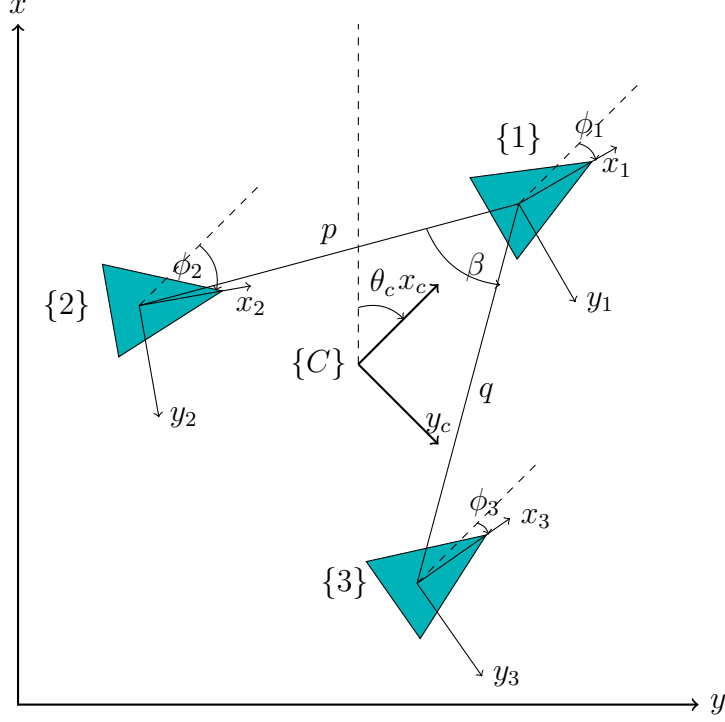


Fig. 1.2: A three robot cluster in an equilateral triangle formation

$$\vec{R}_{12} = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{bmatrix} \quad (1.1)$$

$$\vec{R}_{13} = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \end{bmatrix} \quad (1.2)$$

$$\vec{N} = -\vec{R}_{12} \times \vec{R}_{13} \quad (1.3)$$

$$b_{grad} = \pi/2 - \tan^{-1}(N_y/N_x) \quad (1.4)$$

The first control primitive, navigating to a minimum or maximum, is possibly the most obvious one. To get to the maximum, the robot cluster is simply commanded to travel in the direction of b_{grad} , so it just follows the gradient to a local maximum in the scalar field. Conversely, by going the other direction, the robot cluster will reach a local

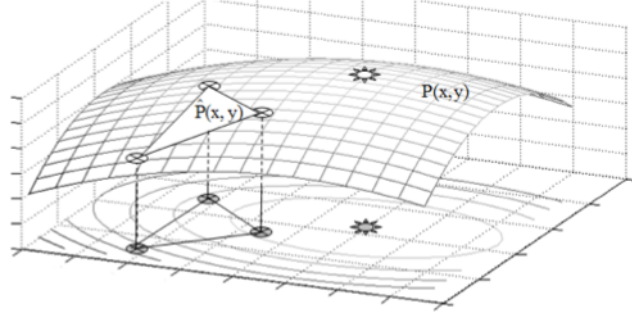


Fig. 1.3: A robot cluster moving in two spatial dimensions, and navigating a scalar surface [11]

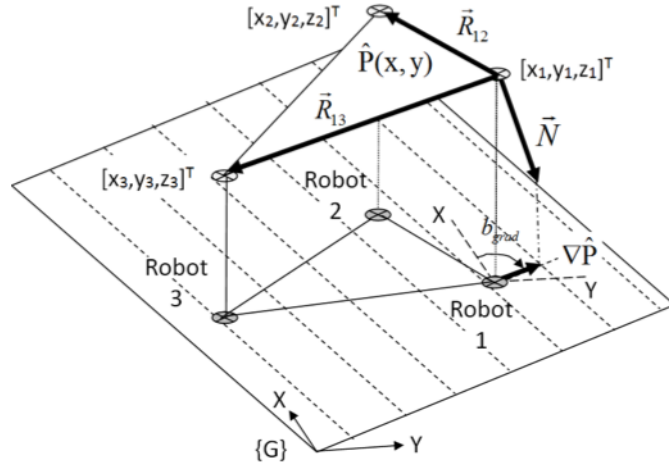


Fig. 1.4: The vectors used to calculate the gradient using a three robot cluster [11]

minimum of the field. With this in mind, the desired bearing (b_{des}) is calculated using Equation 1.5, where $d = 0$ to climb to a maximum, and $d = 1$ to descend to a minimum [11].

$$b_{des} = b_{grad} + (d * \pi) \quad (1.5)$$

This ability to find extrema in a scalar field is useful in a variety of applications, including seeking the source of a transmission, pollutant, et cetera. Figure 1.5 contains simulated results of this navigation method, with two paths going to maximums, and one to a minimum. One clear limitation of these methods is that they only find the local minimum/maximum and provide no indication of whether that local one is also the global value.

The second primitive available is designed to follow the contours of a scalar field. The

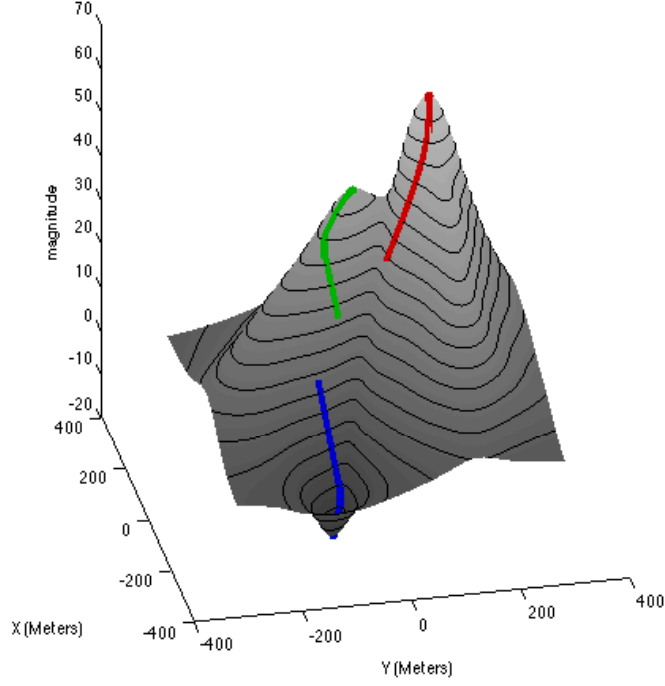


Fig. 1.5: Three simulated clusters of robots, each of which navigates to a different extremum

calculation of the desired bearing is nearly the same as before, however adding $\pm\pi/2$ to the bearing angle results in movement perpendicular to the gradient, which results in following a contour. In the holonomic implementation, additional terms are added to the desired bearing to keep the cluster on the desired contour level. The method for finding the desired angle of travel is shown in equation 1.6, where z_{des} is the desired contour level, z_c is the current cluster scalar level (averaged), and K_{ct} is the cross track gain. The desired \dot{x}_c and \dot{y}_c are calculated using equations 1.7 and 1.8, which simply break the desired bearing into component velocities and multiply by the gains K_y and K_x . Figure 1.6 demonstrates this contour following method for three different contour levels, including both clockwise and counter-clockwise travel.

$$\theta_{des} = b_{grad} + d \times (\text{sgn}(z_{des} - z_c) \times \min[K_{ct} \times |z_{des} - z_c|, \pi/2] - \pi/2) \quad (1.6)$$

$$\dot{y}_c = K_y \sin(\theta_{des}) \quad (1.7)$$

$$\dot{x}_c = K_x \cos(\theta_{des}) \quad (1.8)$$

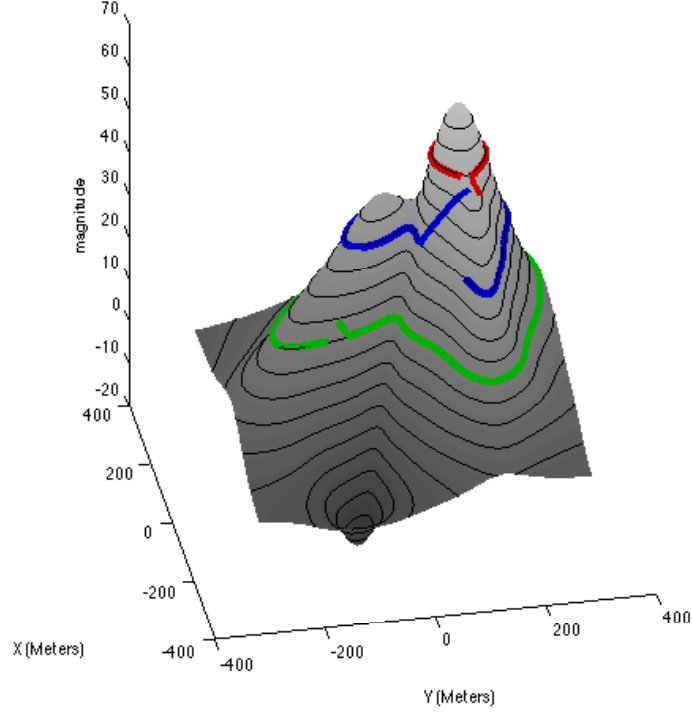


Fig. 1.6: Three simulated clusters of robots, navigating three different contour lines

These two primitives provide basic adaptive navigation capabilities, and were successfully tested in the field. The first set of tests were conducted using autonomous kayaks. The tests were conducted at Stevens Creek Reservoir Cupertino California [1.7](#), and near the Southwestern shore of Lake Tahoe [\[11\]](#). Tests were also conducted at Bellomy Field at Santa Clara University, this time with rovers tasked with following radio frequency fields generated by an antenna set up for this purpose[\[23\]](#). These field deployments served as successful verification of the the adaptive navigation techniques.

1.2.2 Extending the Strategy

Our goal is to explore a region based on scalar measurements taken by individual robots, which are moving in formation. This sensor information is used to navigate with re-



Fig. 1.7: Three automated kayaks traversing a scalar field at Stevens Creek Reservoir in Cupertino, CA [11]

spect to primitive features in the scalar field. These primitives include: minimums, maximums, contours of constant value, ridges, trenches, and saddle points. Each of these features could be important to a mission. For example, maximums and minimums could represent sources or dead zones in a field. By following a contour, one could establish a perimeter at a particular scalar level. Ridges and trenches divide or accumulate resources, and saddle points represent the intersection between a ridge and a trench, which makes them act as a crossroads. All of these primitive features are useful regardless of the type of scalar field the robots are navigating, whether it is temperature, altitude, bathymetry, chemical concentration, et cetera.

While navigating with respect to a single feature has operational value, by chaining these primitive controllers together, more complex tasks can be accomplished. For example, the controller may start finding the local maximum, then move down a ridge, through a saddle and up an adjoining maximum. It is possible to imagine a variety of application or mission specific sequencing strategies, all coordinated through a governing state machine.

Adding a state level controller to the adaptive navigation technique adds an additional layer to the control architecture. Given the mission profile and realtime scalar and robot data, the state machine selects and configures the appropriate primitive controller. The primitive controller, in turn, interacts with the cluster space controller to execute that specific adaptive navigation behavior.

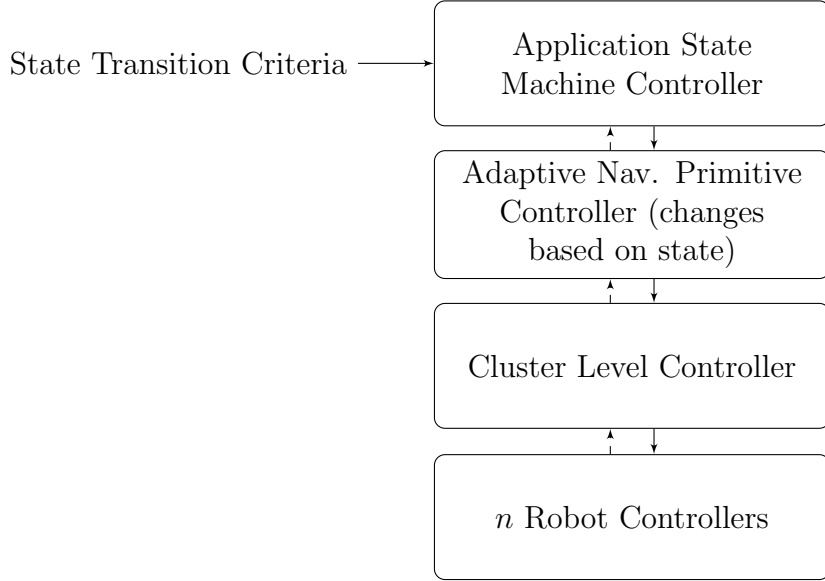


Fig. 1.8: Layers of the Adaptive Navigation Strategy, where solid lines represent commands, and dashed lines represent environmental data

1.3 Project Statement

The research covered in this thesis has contributed to the development of an extended set of adaptive navigation control primitives and has explored the use of a state machine to execute several specific applications.

The first task was performed collaboratively with other members of the multirobot control research group; this author’s role included helping to conceptualize and iterate appropriate control primitives, simulating behaviors, and developing visualizations of how the robot clusters moved through a scalar field. Chapter 2 presents this work.

The second major task was led by this author and consisted of implementing the state machine control approach, integrating it with existing adaptive navigation primitive controllers, developing several plausible application-specific primitive sequences, and simulating the results. Chapter 3 presents this work.

Overall the contribution of this work is a significant extension to the field of multirobot adaptive navigation. A first of its kind comprehensive suite of adaptive navigation primitives has been established; furthermore, it has been shown how these primitives can be sequenced to perform a variety of real world applications.

CHAPTER 2

Ridge, Trench, and Saddle Navigation

With extrema and contour navigation established, the next logical step was the development of a controller that could navigate ridges, trenches, and saddle points. As discussed in Chapter 1, each of these features play an important role in a scalar field. In nature, ridges tend to divide resources, trenches accumulate, and saddle points act as a gateway between local extrema. In addition to their individual value, the ability to navigate these features is also required for higher level tasks, as discussed in Chapter 3.

A number of different control strategies were simulated in an attempt to find a viable solution, with varying levels of success. While the first two control primitives developed used a gradient-based navigation technique, this controller doesn't require the computation of the gradient, and instead uses the scalar measurements as the basis for a differential control strategy. Another key difference in implementation is the need to add an extra robot to verify proper functionality for the ridge/trench following primitive.

The development of the control primitives present in this chapter was a collaborative effort involving multiple researchers within the Robotic Systems Laboratory. The author actively collaborated in the design process, and was responsible for creating simulations to test a variety of techniques.

2.1 Cluster Definition and Simulation

This implementation of the controller utilizes a five robot cluster, as opposed to the three robots used for extrema seeking and contour following. The cluster variables are listed in Table 2.1. These variables are required to describe and control the position and pose of the cluster, which is required for interpreting the sensor data.

The general form of the five robot cluster used in this chapter is displayed in Figure

Table 2.1: Cluster variable definitions

| Variable | Description |
|------------|---|
| x_c | global x position of cluster origin |
| y_c | global y position of cluster origin |
| θ_c | rotation of the cluster in global frame |
| ϕ_i | angle of an individual robot in cluster frame |
| d_2 | line between robots 1 and 2 |
| d_3 | line between robots 1 and 3 |
| d_4 | line between robots 2 and 4 |
| d_5 | line between robots 3 and 5 |
| β_4 | angle between $-\hat{y}_c$ and d_3 |
| β_4 | angle between \hat{x}_c and d_4 |
| β_5 | angle between \hat{x}_c and d_5 |

2.1. This setup mimics aerospace frame conventions, with the x vector for each robot pointing out the front, the y vector pointing out the right side, and the z vector pointing downward. The frame for the cluster of robots follows the same format, as does the global frame. The frames were kept consistent because it is convenient to have the angle between frames equal to zero when the axes are aligned.

As this work consisted of initial development and testing for these particular navigation algorithms, the full set of robot dynamics was not included. Instead of separate dynamics for each robot, the cluster was modeled as a first order system, with one second time constants for translation in the x direction, the y direction, and rotation about the cluster origin. Because the only dynamics were at the cluster level, the formation was assumed to be ideal throughout operation. For this application, the ideal formation is when $d_3 = d_2$, $d_5 = d_4$, and all the β angles are zero. This idealized configuration is depicted in Figure 2.2. The simulation was generated using Matlab and Simulink, and the implementation details can be found in the Appendix.

The scalar values generated via robots 2 through 5 are used to make control decisions, while robot 1 is positioned as a reference point for determining whether the cluster is on the feature of interest. All of these functions are formation dependent, which is why the cluster control method is vital to this application.

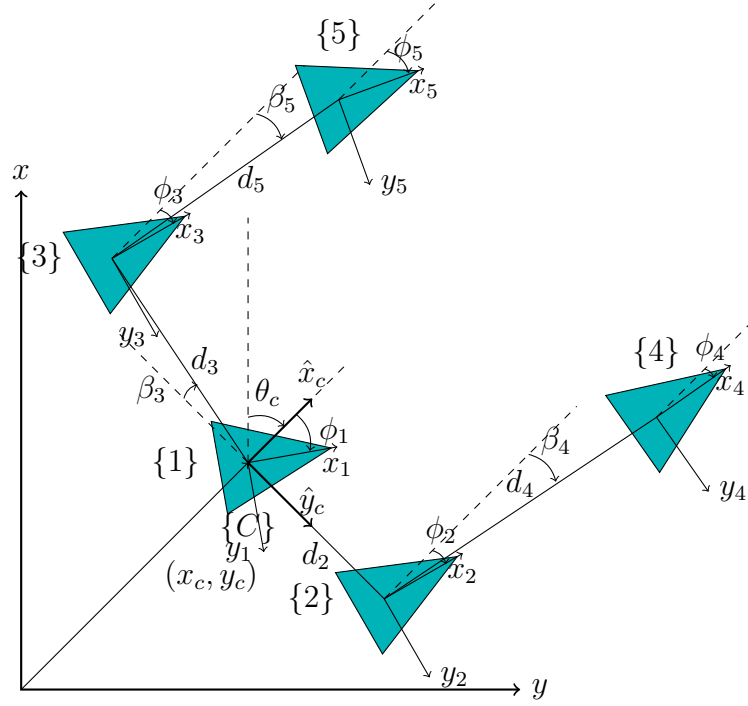


Fig. 2.1: The general form of the 5-robot cluster definition

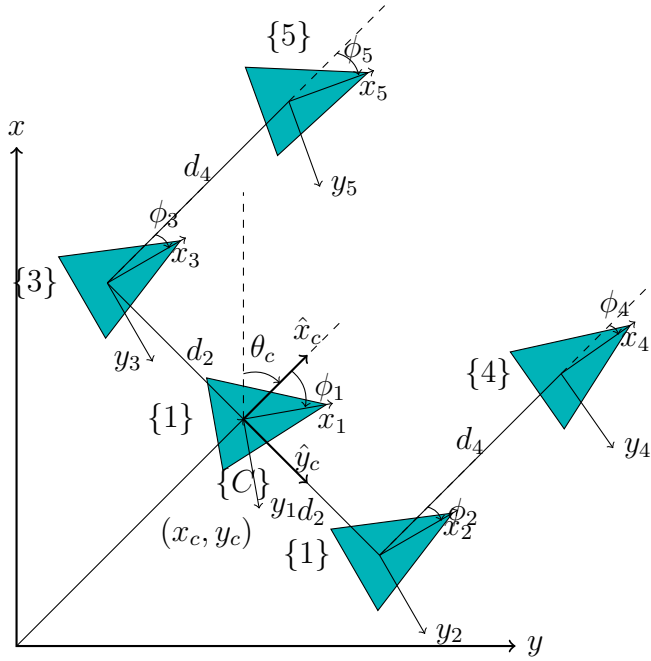


Fig. 2.2: A five robot cluster for ridge and trench navigation

2.2 Ridge and Trench Navigation

There are a number of ways to interpret what a “ridge” is in a scalar field. In this case it is a well defined path of shallowest descent. The red path in Figure 2.3 is following such a ridge. The path is still headed downward along the crest of the ridge, however it is staying on a maximum in the transverse direction. This required directionality was a large challenge during the design process. The blue path in Figure 2.3 is following a trench, which is the inverse of a ridge. To follow it, the robots stay close to a minimum in one direction, and travel upward in another.

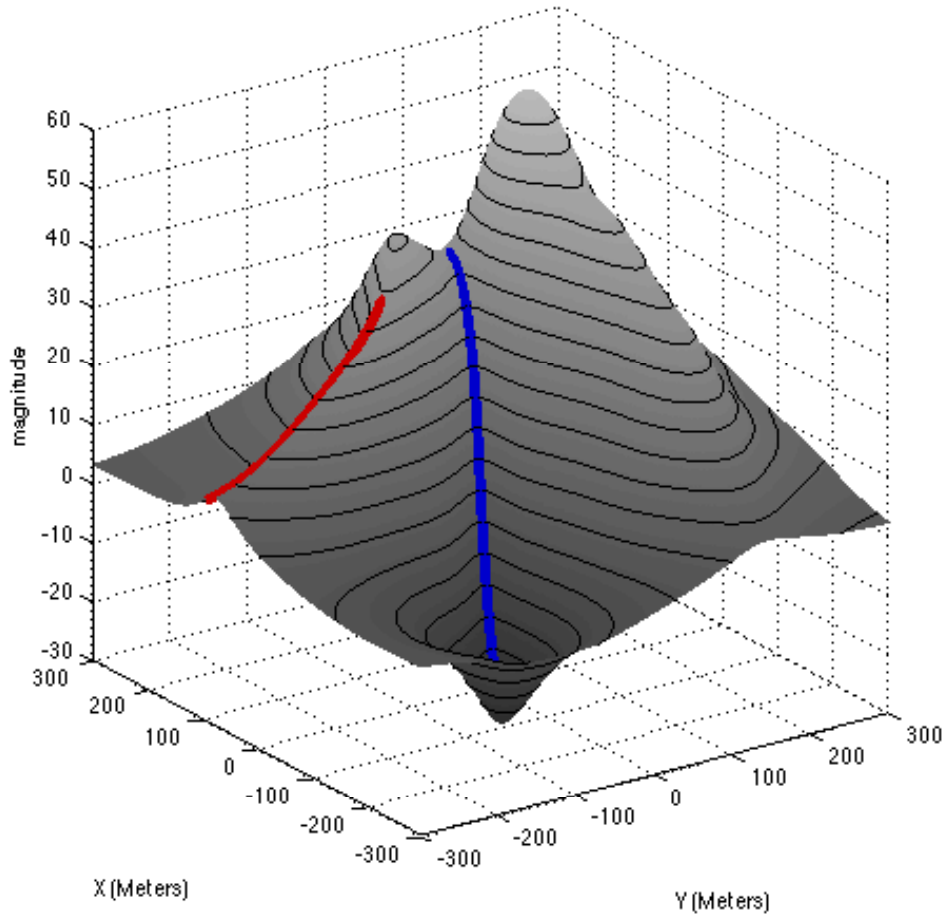


Fig. 2.3: One cluster (in red) navigates down a curved ridge, and another (in blue) goes up a trench to a saddle point.

Another key attribute of both ridges and trenches is a local increase in the curvature of the contour lines in the field, which is centered on the ridge/trench. This is more easily noted in Figure 2.4, as it is a contour map of the same scalar field. There is a clear,

sharp bend in the contour lines where the ridge or trench is located, with the vertex at the crest/minimum of the feature. This can be a useful tool for determining whether a ridge exists in a location, and how prominent it is.

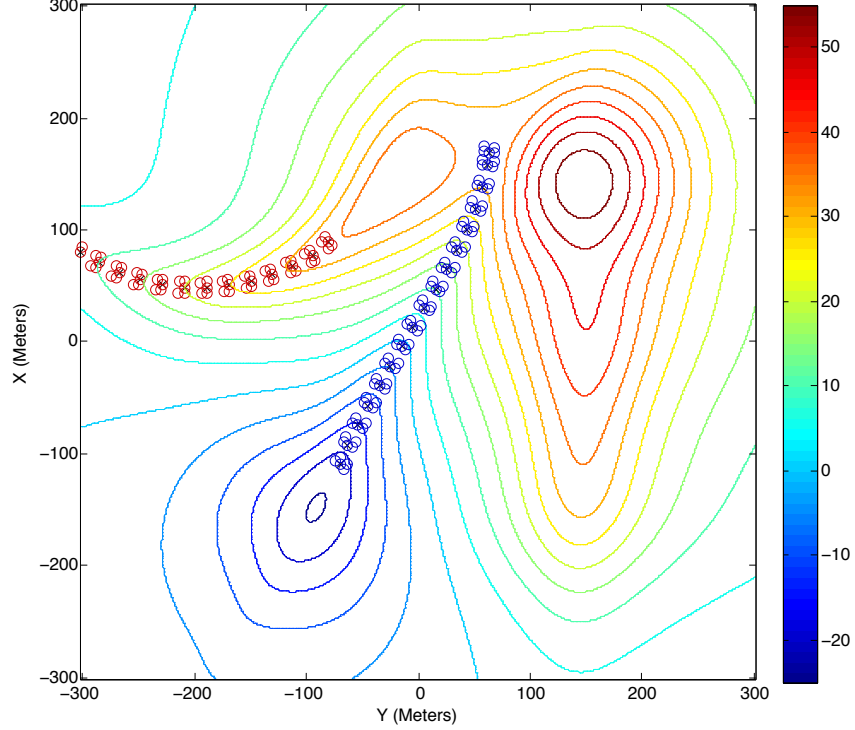


Fig. 2.4: The same paths from Figure 2.3, but from an overhead contour perspective

Knowledge of these attributes was used in the development of control methods for navigating ridges and trenches. Because the robot cluster needs to move up in one direction relative to the ridge/trench, and down in the other, it became evident that it would be difficult to attempt to use control approaches that were independent of the orientation of the cluster, as was accomplished with the extrema seeking and contour following primitives. This led to the development of a controller that is different than the previous gradient approaches.

2.2.1 Control Strategy

As stated previously, the navigation of the robot cluster is based on sensor readings from each robot. These readings are scalar values, which provide an indication of the magnitude of a scalar field at a given point. When following a ridge in the field, the

objective is to remain on the crest of the ridge, but to follow it downward. This means that the cluster needs to climb in one direction, and descend in another. This led to a differential control strategy, which attempts to balance the scalar values of different robots in the cluster.

The angle controller attempts to align the cluster properly with the ridge or trench. The optimal alignment for a ridge is with the cluster x direction pointed down the ridgeline. In this ideal configuration, robots 2 and 3 should have the same scalar value, and robots 4 and 5 should have the same value. In this state, if the ridge were perfectly symmetrical, the cluster would be perfectly aligned. Equation 2.1 generates the rotational velocity command for the cluster, where v_{turn} is the maximum velocity an individual robot within the cluster is permitted to contribute to turning the formation. This value is selected to prevent actuator saturation. This means $\frac{v_{turn}}{d_2}$ is the maximum turning rate of the cluster. The trench controller is the same, except the cluster is oriented up the trench, instead of down the ridge. For all equations in this section, $d = 1$ for ridge navigation, and $d = -1$ for navigating a trench.

$$\dot{\theta}_c = (-d) \frac{v_{turn}}{d_2} \times \text{sign}[(z_2 - z_3) - (z_4 - z_5)] \quad (2.1)$$

The controller for the cluster for the velocity in the x direction is fairly straightforward. It's sole goal is to continue downhill (or uphill in the case of a trench), and is represented by equation 2.2. It subtracts the sum of the scalar values of the front two robots from the sum of the values for the two robots in the rear. It gives a forward command if the result is positive, and a reverse command if the result is negative.

$$\dot{x}_c = d \times \text{sign}[(z_2 + z_3) - (z_4 + z_5)] \quad (2.2)$$

The controller for the y velocity is quite similar, except that it attempts to move up the surface if it is on a ridge, and down if it is on a trench. This upward tendency keeps the cluster centered on the ridge in the y direction. Equation 2.3 is used to compute the direction of the desired cluster velocity in the y direction. It compares the scalar readings of the robots on one side of the ridge, to those on the other side of the ridge, in an attempt to center itself.

$$\dot{y}_c = d \times \text{sign}[(z_2 + z_4) - (z_3 + z_5)] \quad (2.3)$$

This controller is effective, however it operates on the assumption that the cluster of robots is spanning a ridge in the scalar field, and in good alignment. When this assumption is incorrect, acceptable performance can not be guaranteed. To counter this issue, a fifth robot is used for the purpose of sensing whether or not the cluster is spanning the ridge. Making this check is as simple as confirming that robot 1 is reading a higher scalar value than robots 2 and 3, plus an adequate margin (equations 2.4, 2.5). In the case of a trench, it checks to be sure that z_1 is less than z_2 and z_3 , and subtracts the margin.

$$z_1 > z_2 + z_{mar} \quad (2.4)$$

$$z_1 > z_3 + z_{mar} \quad (2.5)$$

The z_{mar} would be selected based upon how defined the ridge or trench is required to be. It is also important to note that if the cluster fails the check, it only means that it isn't guaranteed to be on a ridge. The ridge could just be too flat, or the cluster may be in a poor orientation. If the cluster falls off the ridge, due to a disturbance or a complex ridge shape, an altered controller must be used to lock onto the ridge again.

All of the control equations produce discrete velocity values for the purpose of reducing dependency on the topology of the surface. This is extremely beneficial, as the environment robots traverse is often largely unknown, and being less dependent on the shape of the scalar field means less prior knowledge is required.

The tracking performance of the controller can be represented by scalar field differentials generated from measurements taken by the individual robots. Figure 2.5 is a time response plot for the front and rear scalar differentials of the robot cluster as it follows the red path in Figure 2.4. These two differentials represent both the lateral and angular alignments of the robot cluster, and are used in the corresponding control equations. As seen from the plot, there is a brief period where the cluster is aligning with the field, followed by small oscillations for the rest of the time history. These steady-state oscillations are caused in part by the discrete velocity commands generated by the control equations. Despite these small oscillations, the performance is excellent. The simulated cluster not only follows the ridge, but does so very closely, as the differentials settle to within ± 0.005 scalar units.

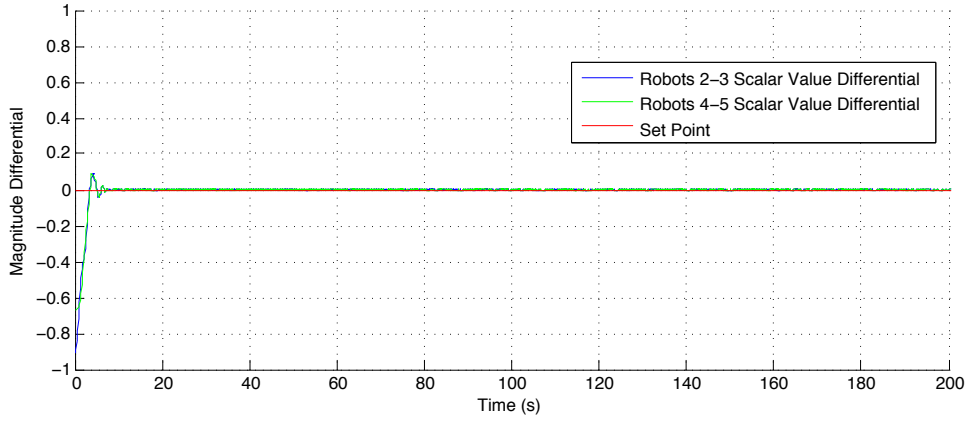


Fig. 2.5: A time history plot of the scalar differential between robots 2 and 3, as well as between robots 4 and 5, from the same simulation run as Figures 2.3 and 2.4.

2.3 Saddle Point Navigation

A saddle point is a point at which the derivative is zero in both the x and y directions, but is not a local maximum or minimum. They can occur in a variety of places, but are often found between two local maximums, and/or at the intersection of ridge and trench features. Figures 2.6 and 2.7 show two cluster paths, the green one using the ridge controller to descend down into the saddle point, and the blue path using the trench controller to go up into it.

Because of the differential control style used in both the ridge and trench controllers, the cluster automatically settles at the saddle point. This removes the requirement for a separate controller to hold the robot cluster on the point. As seen in Figure 2.7, the two clusters sit on the point, but are oriented ninety degrees from one another. This occurs because in one direction a saddle point has ridge-like features, and in the other they are similar to a trench.

The results of the saddle point scenario can be represented using differentials as well. Figure 2.8 shows how the differentials on all four sides change over time. In this case it is for the path the robot cluster takes up a trench to a saddle point as shown in Figure 2.6. As evidenced from the plot, the front and back differentials have only small transient behaviors correlating with the time the cluster takes to snap onto the ridge and align. This behavior is very similar to Figure 2.5. In contrast, the right and left differentials have significant magnitudes during much of the travel time. This is what results in the

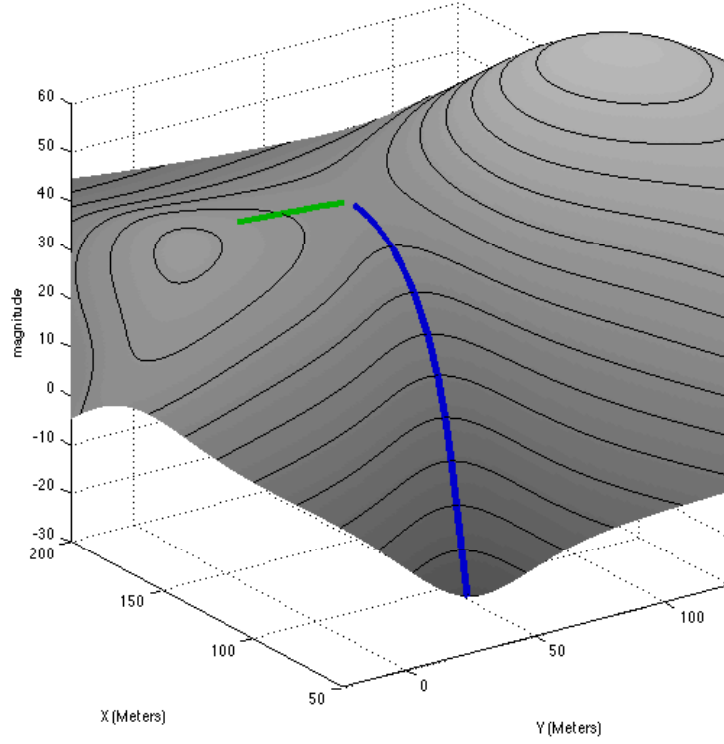


Fig. 2.6: The blue cluster is the same set of robots from Figures 2.3, and 2.4, traveling up a trench to the saddle point, and the green path is coming down a ridge into the same point.

\hat{x}_c control equation commanding movement up the trench. Upon reaching the saddle point these differentials become minimal, oscillating between -0.001 and $+0.001$ scalar units. This behavior indicates the cluster has reached steady state on the saddle point.

These results confirm the behavior evidenced in Figures 2.6 and 2.7, which indicate that the controllers go up a trench and reliably settle at a saddle point.

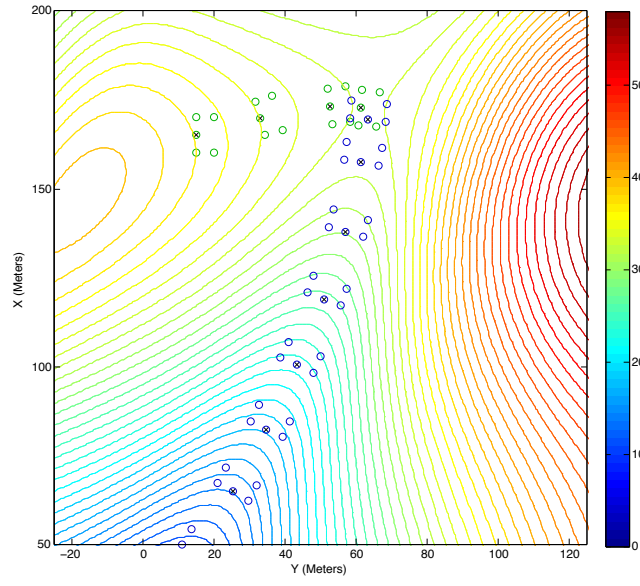


Fig. 2.7: The same paths from Figure 2.6 from a two dimensional overhead perspective

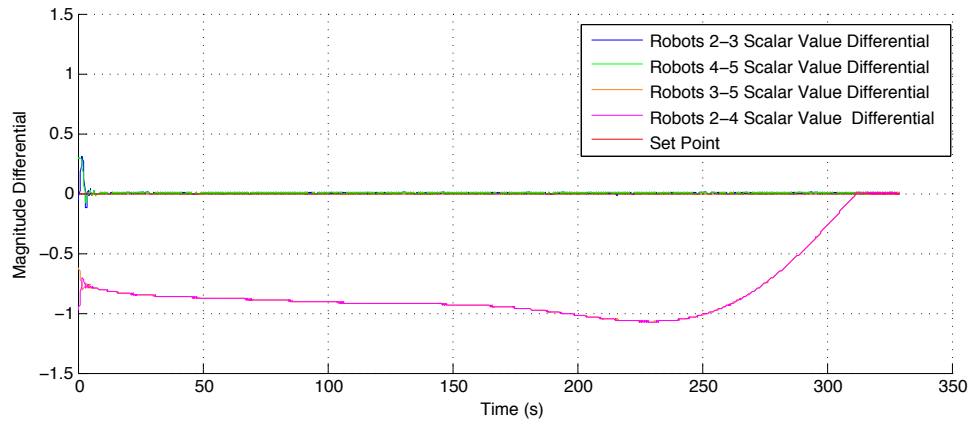


Fig. 2.8: A time history plot of the scalar differentials between robots 2 and 3, 4 and 5, 2 and 4, and 3 and 5, as the cluster travels up a trench to a saddle point

2.4 Discussion

While the performance of these controllers is excellent, as seen from the time responses, there are some additional considerations to be taken into account. For one, the simulation does not model everything. The dynamics of the individual robots are not included, and the scalar field is assumed to be continuous and without noise. Both of these factors will have an impact on the performance of the navigation strategy. While required for the final implementation, the inclusion of robot dynamics is unlikely to have too much of a negative impact, as other formations have been successfully implemented in previous work. That said, an adjustment of the speed of the robot cluster may be required to account for maneuvering time. The impact of the scalar field on the navigation is a more complex issue. This brings us to a second major point, the sizing of the cluster relative to the field.

The sizing of the robot cluster effectively has upper and lower bounds, which are dependent on mission requirements. The upper bound is designated by the features of interest within the scalar field. The cluster needs to be sufficiently small relative to the features in order to successfully navigate them. If it is too large, the navigation algorithm simply won't react to certain field features. This misrepresentation of the field is essentially signal aliasing based upon spatial parameters, which implies that the maximum size of the cluster should be half the size of the scalar feature, to comply with the Nyquist criteria.

The minimum size is based upon sensor and spatial noise. We are familiar with sensor noise, and in that scenario it clearly makes sense to keep the cluster large enough to get distinct readings from the sensors, which allows the controller to establish the required differentials. We consider spatial noise to be anything naturally occurring in the scalar field that is not relevant to the current mission. For example, if we are trying to determine the overall trend of a system, we want to see large scale features, not local changes. While spatial noise is from a different source, it is dealt with the same way. Increasing the cluster size will effectively damp out this noise, the same way you change sampling frequencies to attenuate signals that are not of interest.

CHAPTER 3

Adaptive Field Applications

While there are valid field applications for the individual control primitives, there are many more that can be generated by using various combinations of these simpler controllers. This chapter presents and discusses four such applications and their implementation details. Each application was developed and demonstrated in a simulation environment to explore the technique. As an initial study of the capability, many idealizations were assumed. Future work is required to more rigorously evaluate these strategies given real world effects such as noise, vehicle dynamics, complex fields, et cetera.

3.1 Overview of Architecture

The application controllers are constructed with a standard state machine configuration. There are a number of available states, which are each an implementation of an adaptive navigation control primitive. For each application there is a set of transition criteria, which is based upon the attributes of the scalar field and the position of the robot cluster. The criteria for each state vary from application to application as required. Figure 3.1 displays the layered structure of the application control architecture. State transitions within the state machine application controller are defined based upon mission requirements, which makes use of the full set of adaptive navigation primitives. The primitive used changes with the state of the system. The selected primitive generates cluster commands, which the cluster controller uses to generate commands for each individual robot controller.

Although the control primitives can be executed with cluster of three or five robots, it was unclear at the start of the project as to whether additional field samples would be necessary to support more robust performance and/or switching between states. For this reason, a nine-robot cluster was adopted, with the idea that a wider set of data points

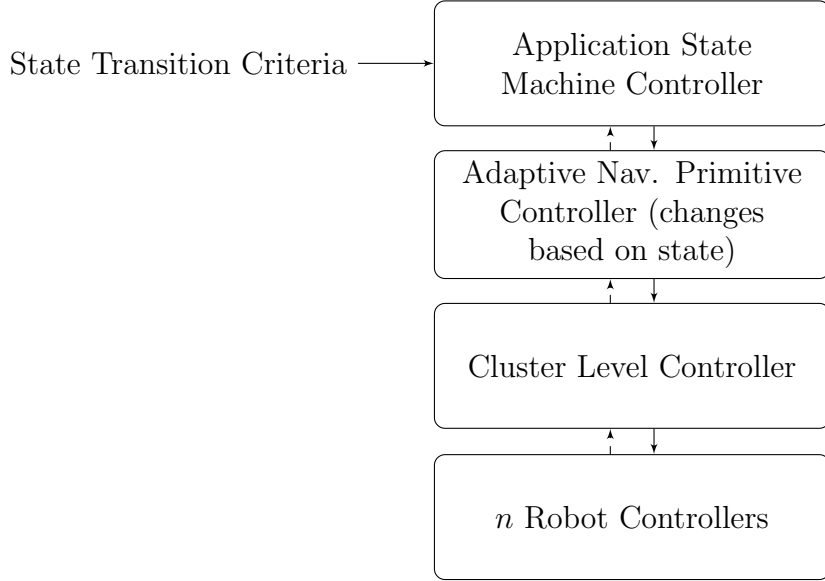


Fig. 3.1: Layers of the Adaptive Navigation Strategy, where solid lines represent commands, and dashed lines represent environmental data

would be available for evaluation and use. This formation allows an instantaneous three dimensional curvature estimate. The cluster was arranged in a three by three, equally spaced grid, with the ability to hold formation perfectly, as in Figure 3.2.

Figure 3.2 shows how the nine robot cluster is defined. The cluster frame is positioned on the center robot with the z vector pointing downward. d is the distance between one robot and the next. The different applications use a different number of robots, and each state within those controllers also uses a particular subset of the nine. That said, some of these methods do use all nine to execute a particular task. Future versions of these strategies may use other formations or techniques to acquire the necessary field information with fewer robots.

There are a few idealizations and assumptions associated with this early version of the technique. For example, the threshold values used for state transitions work with the given scalar field but are not particularly robust. There are also particular features required for the controllers to function, or that could result in the cluster getting trapped on a contour. One can compensate for issues such as these using the current architecture by adding additional transition conditions to avoid hazards. Finally, the simulations did not take environmental or sensor noise into account, so a field version would have to be adapted for the noise profile associated with the application.

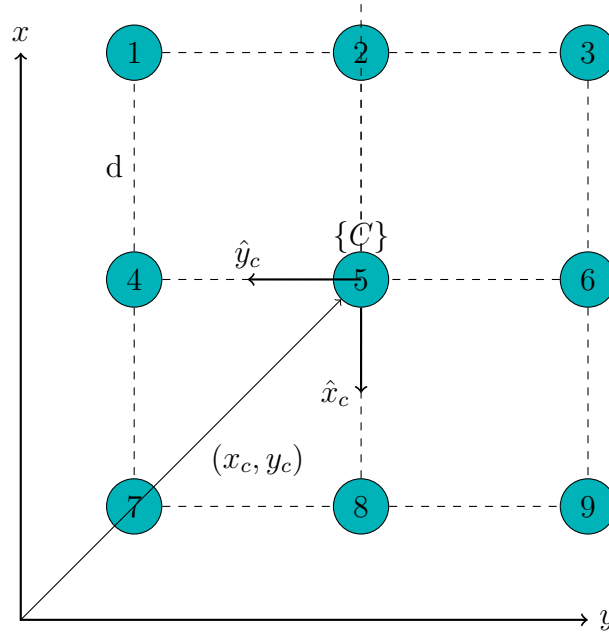


Fig. 3.2: The nine-robot cluster used for the state level controllers

3.2 Ridge Controller with Recovery

A simple example of a state machine implementation is a variation of the ridge and trench controller presented in Chapter 2 which is capable of recovering after it loses the ridge, as well as finding a ridge from an unknown location. An overview of the state controller is depicted in Figure 3.3.

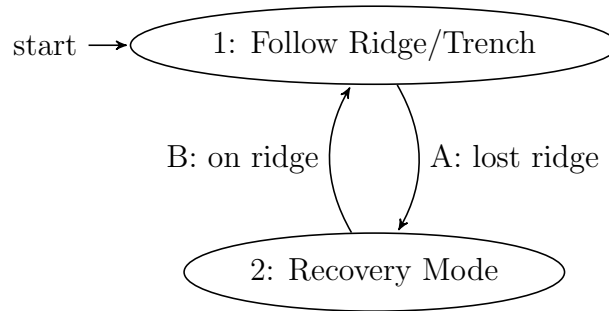


Fig. 3.3: State diagram depicting the states required for the ridge following controller with a recovery mode

There are two states required for this strategy. The first state, Follow Ridge/Trench, is the standard controller presented in Chapter 2, where $d = 1$ for a ridge, and $d = -1$ for a trench.

Table 3.1: Summary of Transition Criteria for Ridge Recovery

| Transition | Criteria |
|----------------------|---|
| A: $1 \rightarrow 2$ | $z_2 < z_1 + z_{mar}$ or $z_2 + z_{mar} < z_3$ |
| B: $2 \rightarrow 1$ | $z_2 > z_1 + z_{mar}$ and $z_2 > z_3 + z_{mar}$ |

The recovery controller is very similar to the standard ridge controller. The only difference is that the controller for \dot{x}_c has the opposite sign, as in equation 3.1.

$$\dot{x}_c = -d \times \text{sign}[(z_2 + z_3) - (z_4 + z_5)] \quad (3.1)$$

As before, $d = 1$ for a ridge, and $d = -1$ for a trench. Changing the sign of \dot{x}_c effectively makes the cluster go uphill in both the cluster x and cluster y directions, therefore it will be heading toward a maximum. Following an upward trajectory will take it back to the top of the ridge, and the unchanged $\dot{\theta}_c$ controller will align it when it is near the ridgeline. At this point the controller will recognize that the cluster is straddling the ridge, and the controller will return to the standard navigation state, to continue down the ridge. The opposite is true for the equivalent trench controller. It will head downward in the recovery state, until it again straddles the trenchline. This variation of the controller is used in all of the state level controllers presented in this chapter.

Figures 3.4 and 3.5 display two paths, one of which seeks a ridge then follows it, and the other does the same with a trench. In this case, the cluster begins far from the ridge, but the controller is equally effective when faced with smaller deviations.

Figure 3.6 contains time history information for this controller. The first subplot indicates the state of the system as it changes based upon the transition criteria from Table 3.1. For this particular controller, both transitions A and B are determined by whether the scalar value of robot 2 exceeds the other two by a threshold value. The second subplot displays the transition criteria by plotting the scalar differential between robots 2 and 1, and robots 2 and 3. If either of these lines is below the scalar threshold, z_{mar} , then the cluster is effectively off of the ridge, and the recovery controller activates.

From the response information displayed in Figure 3.6, we can confirm that the robot cluster began off a ridge, and climbed until it reached one. The state transitioned just as the ridge was attained, and remained constant as it traveled down. There is one point in the response where state one is entered briefly before the ridge is reached due

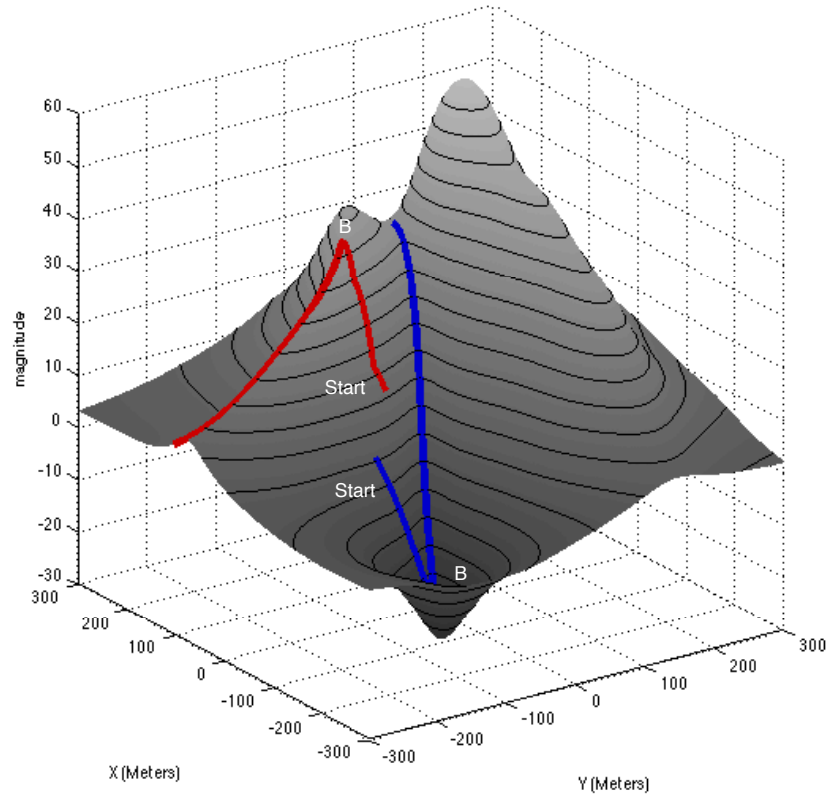


Fig. 3.4: Two robot clusters navigate a scalar surface, one of which seeks and tracks a ridge, the other does the same with a trench

to the rotation of the cluster, however this lasts only a moment as the cluster properly aligns itself. Only five robots are required for this controller. In the nine robot cluster, we use robots 1, 2, 3, 7, and 9.

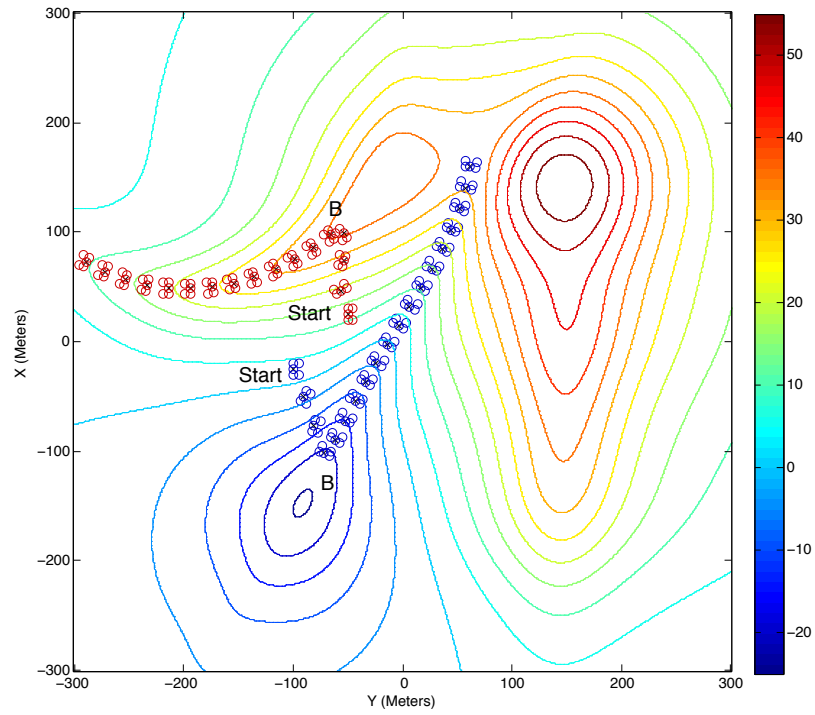


Fig. 3.5: The same robot clusters from Figure 3.4 viewed from above

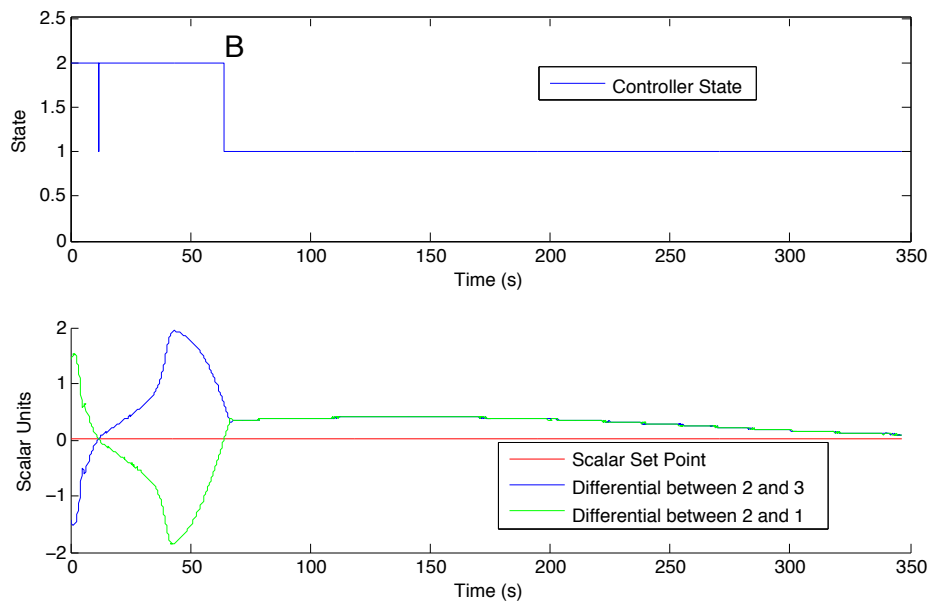


Fig. 3.6: Time history plots of the states and transition criteria for the red path from Figures 3.4 and 3.5

3.3 From Local Maximum to Local Maximum

The control primitive that finds a maximum in a scalar field is limited because it will find a local maximum, and remain there. This application controller attempts to mitigate this limitation, moving from one local maximum to another, while taking an efficient path. The cluster first moves to a local maximum, then circles a contour until it finds a ridge, follows that ridge to a saddle point, then moves up to a neighboring maximum. These same methods can be used for moving from minimum to minimum, following trenches instead of ridges. This particular controller uses three different control primitives to accomplish its objective. Figure 3.7 is a state diagram presenting a high level view of the required states and transitions, and Table 3.2 provides the criteria for these transitions.

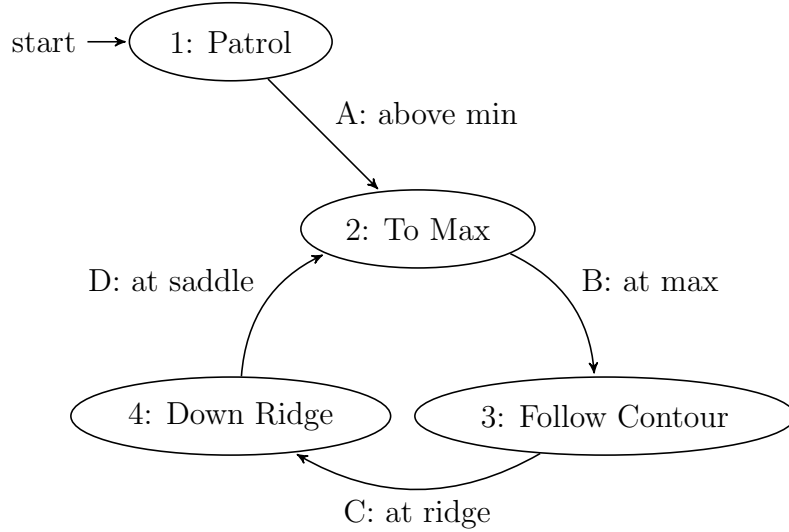


Fig. 3.7: State diagram for going from local maximum to local maximum

Table 3.2: Summary of Transition Criteria for Extrema Hopping

| Transition | Criteria |
|----------------------|--|
| A: $1 \rightarrow 2$ | $z_c > z_{cutoff}$ |
| B: $2 \rightarrow 3$ | $z_5 - z_{mar} \geq \{z_1, z_2, z_3, z_4, z_6, z_7, z_8, z_9\}$ |
| C: $3 \rightarrow 4$ | $c_{curv} = (c_1 + c_2)/d > c_{min}$ and $ z_5 - z_{contour} < e_{max}$ |
| D: $4 \rightarrow 2$ | $z_5 - z_{mar} > z_{4,6}$ and $z_5 + z_{mar} < z_{2,8}$ |

The first state of the controller is a Patrol mode. It could be a search pattern appropriate

for the current application, a straight line, a lawn cut, et cetera. In the case of this simulation, the cluster was traveling in a straight line, in the x direction of the cluster frame. There are many viable options for the transition criteria; however, for simulation purposes, it was set to transition when the scalar value at the center of the cluster passed a threshold. This criteria is represented by Equation 3.2, where z_c is the scalar value at the center of the cluster, and z_{cutoff} is the cut-off value for the transition. When this inequality is true, the controller switches to the next state.

$$z_c > z_{cutoff} \quad (3.2)$$

Future iterations of this patrol mode would likely have more strict criteria for transition, incorporating gradient magnitude, spatial regions, et cetera.

To get to the first local maximum, the robot cluster enters the To Max state and utilizes the technique described in Section 1.2.1 in order to travel up the gradient of the scalar field. Robots 1, 3, and 8 are used to estimate the gradient. The robot cluster simply follows the local gradient upward until it reaches a local maximum. It transitions to the next state when it determines that it has arrived at a maximum when Equation 3.3 is true, where the z values are the scalar values sensed by each robot. This condition indicates that the center robot is at a maximum. The margin z_{mar} is an operator selected value that should be significant enough to avoid a false positive due to sensor noise and other environmental disturbances. Similar margins are used in for many transition criteria to increase robustness.

$$z_5 - z_{mar} \geq \{z_1, z_2, z_3, z_4, z_6, z_7, z_8, z_9\} \quad (3.3)$$

In the third state, Follow Contour state, the contour controller described in Section 1.2.1, is used to find a ridge leading from a peak. Once again, robots 1, 3, and 8 are used to estimate the local gradient. The controller follows a contour a preset value below a peak. This value can be changed based on the characteristics of the field. The transition to the next state occurs when a ridge in the field is located. The ridge is identified based on the local curvature of the contour lines. Figure 3.8 is an example of a cluster of robots navigating a contour line, and demonstrates how the contour controller orients the robots with respect to the contour line.

The curvature of the contour line can be estimated using this information. The contour

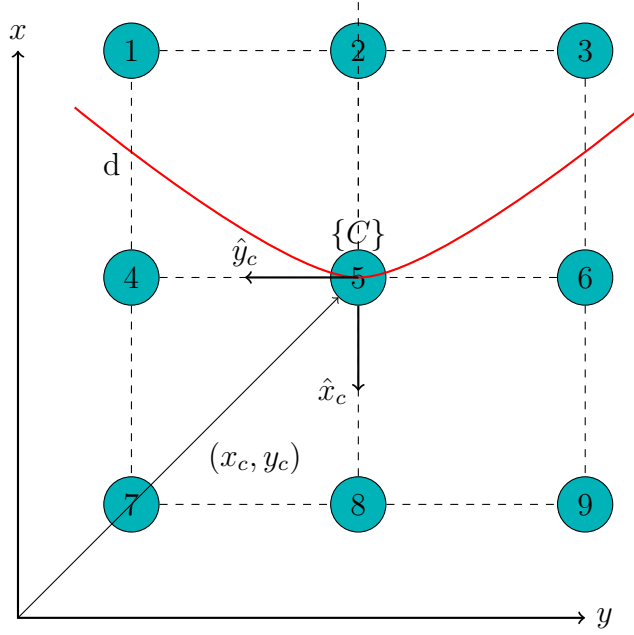


Fig. 3.8: A robot cluster relative to a contour line (displayed in red).

line of interest is the one corresponding to the scalar value of the center robot, z_5 . We know, by definition, that it passes through robot 5. To calculate the curvature, two more points on the contour are required. To find one of these points, equation 3.4 is used to interpolate the location of the point between robots 1 and 7. Equation 3.5 is used to do the same between robots 3 and 9.

$$c_1 = 2d \frac{(z_5 - z_1)}{(z_7 - z_1)} - d \quad (3.4)$$

$$c_2 = 2d \frac{(z_5 - z_3)}{(z_9 - z_3)} - d \quad (3.5)$$

With these two interpolated points and the center of the cluster, there is enough information for a curvature estimate using equation 3.6.

$$c_{curv} = \frac{(c_2 - 0)}{d} - \frac{(0 - c_1)}{d} = \frac{c_1 + c_2}{d} \quad (3.6)$$

This calculated curvature value is used to determine whether the location meets the criteria to be considered a viable ridge. A cut off value is assigned to determine whether the requirement is met. This value is assigned based on the field in question. In addition

to the curvature requirement, the controller requires that the cluster has already reached the desired contour level, by checking that $|z_5 - z_{contour}| < e_{max}$ where e_{max} is the maximum allowable contour tracking error. If a suitable ridge is located while at the correct contour level, the transition to the next state occurs. Because the contour was being followed with the \hat{x}_c vector pointing downhill, the cluster will already be facing the correct way to execute the ridge following algorithm.

The final state of this application controller is the navigation of down a ridge to a saddle point. This state uses the method described in Chapter 2, using scalar values for robots 1, 2, 3, 7, and 9. For the purposes of this navigation strategy, the ridge navigation strategy from section 3.2 was implemented to provide an extra level of robustness to unknown conditions. As before, upon reaching a saddle point, the cluster holds position. The cluster determines it has reached the saddle point by checking the relationship between the scalar values of the robots in the cluster. Two checks are required to confirm the cluster location. The first is that $z_5 - z_{mar}$ is greater than z_4 and z_6 . Similarly, $z_5 + z_{mar}$ must be less than z_2 and z_8 . Once the cluster is confirmed to be on the saddle, the controller returns to the ‘Go to Max’ state, to proceed up to the next maximum.

Simulation results of this controller have been successful. Figure 3.9 is a demonstration of the controller. The plot shows the cluster moving up to a maximum, down into a saddle, and up to the next maximum as expected. There is some noticeable deviation from the optimal path as the transition from the ridge controller to the extrema seeking controller takes place due to a turning maneuver, but it still reaches the desired destination.

Figure 3.10 contains the time history information for the cluster as it switches from state to state to complete its objective. As expected, as soon as it reaches the scalar set point it exits the patrol state. Each line on the third subplot is the difference between the scalar value of each robot, and the center one. To meet the transition criteria, all of these differentials need to be above the threshold point, which indicates that the cluster is sitting on the maximum. The criteria for the next transition is twofold; the curvature needs to reach the minimum value to identify a ridge, and the scalar value of the cluster and the desired contour need to be the same within a certain margin. Both of these criteria are represented on the fourth subplot. The final subplot represents the criteria for identifying a saddle point. Once again scalar differentials between robots are used to determine the shape of the local area.

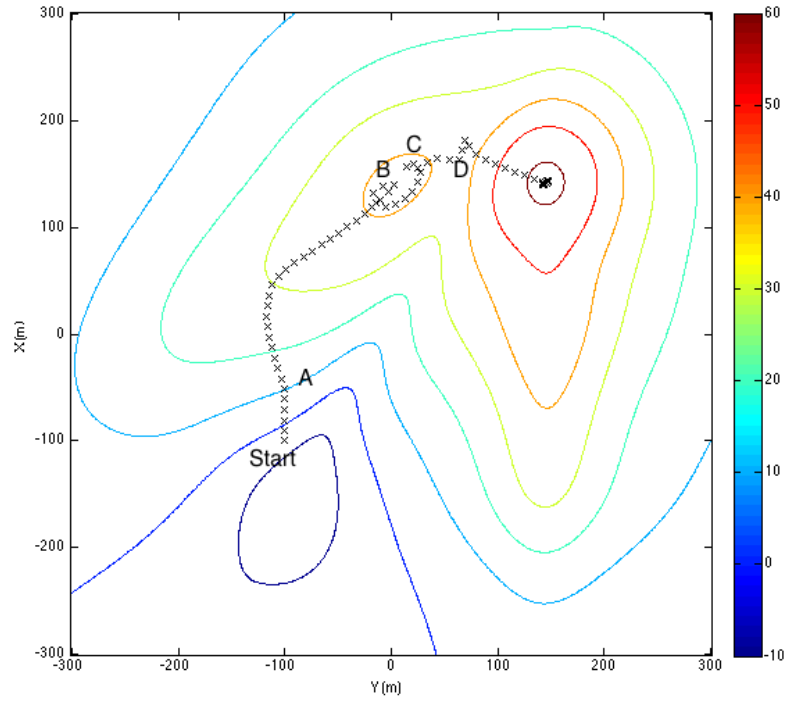


Fig. 3.9: A robot cluster travels to one local max, then to another

As seen from the figures, each transition occurs at the intended point, and the cluster makes it from one peak to the other, despite transient behaviors due to turning mechanics. All nine robots are used in this method. Nine are used to determine whether an extrema is present, three for extrema finding and contour following, and five for the ridge and saddle controller.

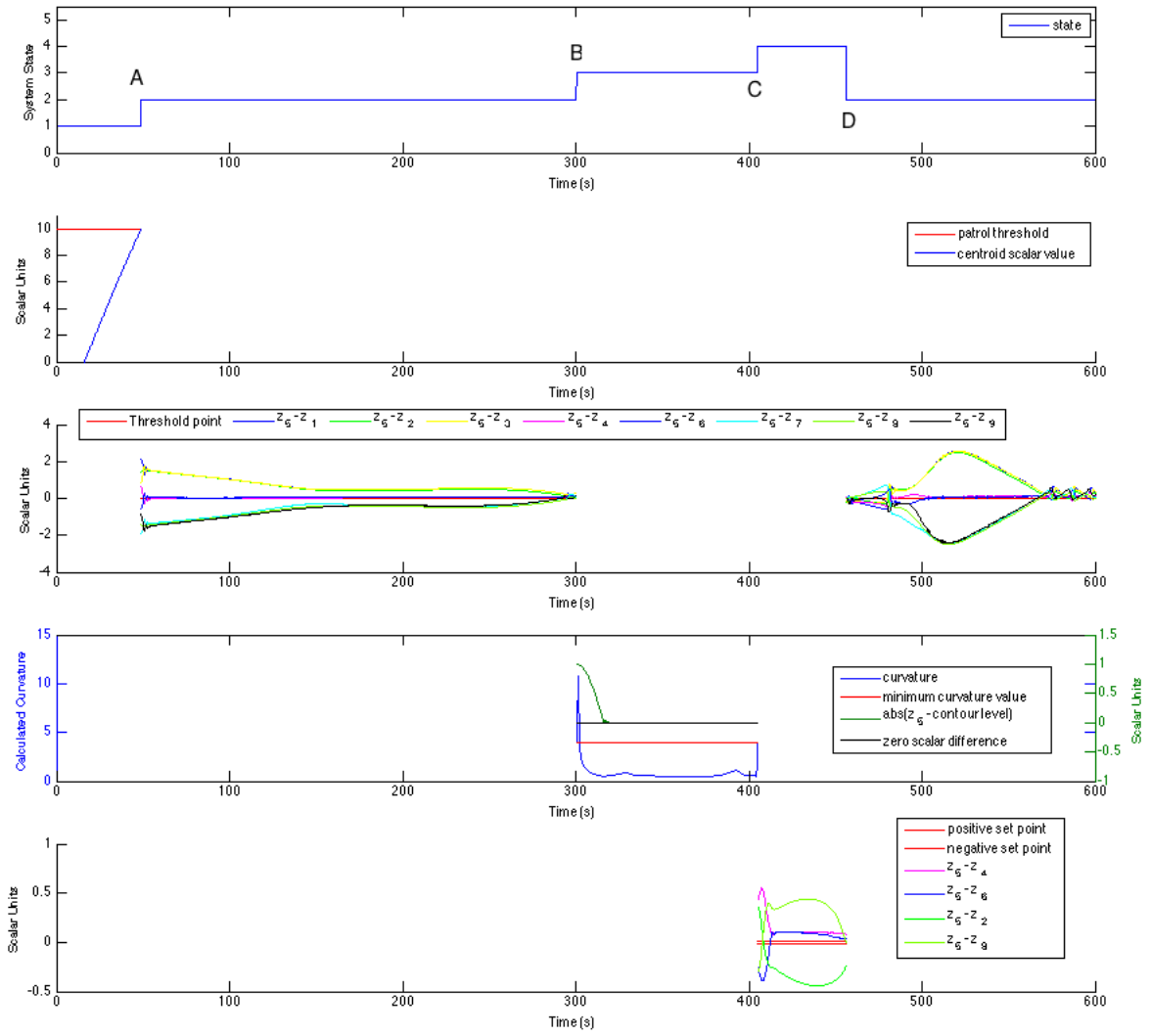


Fig. 3.10: The time response plots for the states and transition criteria throughout the path displayed in Figure 3.9

3.4 Contour Mapping

The contour mapping strategy is an alternative to traditional mapping strategies. Rather than exploring the area based on spatial criteria, this method explores the area based on the scalar levels and contour lines. In this fashion, a topographic map can be generated directly, as opposed to conducting a comprehensive sweep of the area. The controller presented here centers the contour map around a local maximum; however, the same methods could be used to map an area around a minimum. The controller begins by ascending to a local maximum, dropping a fixed amount while aligning with a contour, following that contour around past the starting point, and repeating the process until the area has been sufficiently mapped. Figure 3.11 shows the state transitions required to execute contour mapping, and Table 3.3 contains the required conditions for each of these transitions to take place.

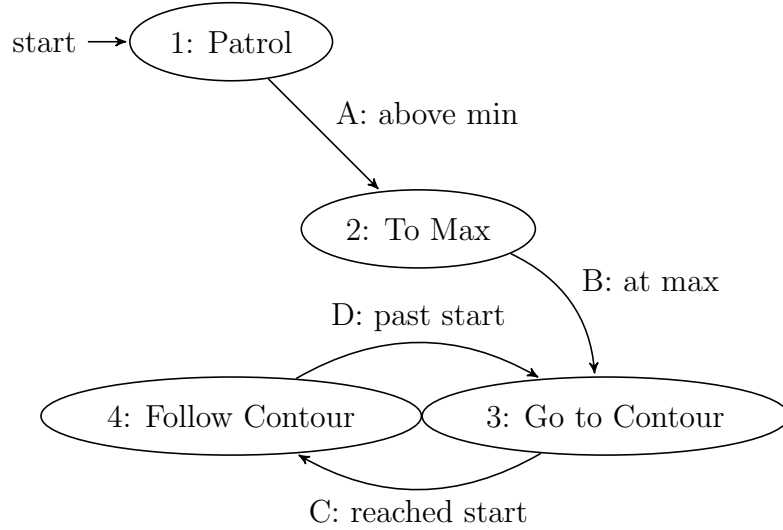


Fig. 3.11: State diagram for mapping the contour lines around a peak

Table 3.3: Summary of Transition Criteria for Contour Mapping

| Transition | Criteria |
|------------|---|
| A: 1 → 2 | $z_c > z_{cutoff}$ |
| B: 2 → 3 | $z_5 - z_{mar} \geq \{z_1, z_2, z_3, z_4, z_6, z_7, z_8, z_9\}$ |
| C: 3 → 4 | $ z_5 - z_{set} < z_{mar}$ and $ \dot{\theta}_c < \dot{\theta}_{mar}$ |
| D: 4 → 2 | $\phi > \phi_o + \phi_{mar}$ and $r > r_o \pm r_{mar}$ |

Once again, the first state is a Patrol mode with an arbitrary termination condition. In the simulation, the controller was set up such that the state transition occurred once the cluster value exceeded a defined scalar value, as with the previous navigation strategy.

The cluster then enters the To Max state, using the extrema controller to climb to the local maximum. When the maximum is reached as defined by transition criteria B in Table 3.3, the location of the point in the global frame is recorded for later reference, then the controller transitions into the next state.

The purpose of the third state, Go To Contour, is to travel to the next contour line from the previous location, and to align properly. To achieve this, the contour following control primitive is used, and the desired contour level is set to a preset interval below the previous location. As three robots are required to compute the gradient, the scalar values from robots 1, 3, and 8 are used to make the calculation. The desired contour level is compared to the scalar value of robot 5. The size of this interval is set prior to the mission, and is selected to match the desired resolution of the contour map. The controller transitions to the next state when it determines that the cluster is on the appropriate contour, and aligned properly. There are multiple checks to determine whether these conditions are met. The requirements are that the scalar level z_5 is within a preset margin of the desired level, and that the cluster is aligned with the contour within a specific margin of error. Once the cluster is in the proper position relative to the contour line, the controller transitions to the next state.

The fourth state, Follow Contour, is designed to follow the contour line around its entire perimeter, and then go a little further to ensure complete coverage. The contour following control primitive is used, once again using robots 1, 3, 8, and 5. When the transition into this state occurs, a polar position is recorded with the maximum as the origin. This means there is a known starting radius and angle. These values are used as a reference when determining whether the robot cluster has passed through the start point again. It is important to use both of these values, as depending on the shape of the contour, the angle from the maximum (and certainly the radius) will repeat itself; by combining the two, we ensure a unique position.

The transition back to the Go To Contour state occurs when the radius from the maximum to the cluster is within a preset margin of the starting radius, and the angle is past the starting point by a preset value. Hysteresis is added to ensure that the transition does not occur the first time around the contour.

The results of the contour mapping controller are positive, as seen in Figure 3.12, which is an example of a successful mapping. As seen in the figure, the contours of interest are an even distance apart and completely encircled, as desired. Figure 3.13 contains time response plots for the state of the system and the transition criteria as the robots map the contours of the field. From the response plot we can see that each transition is successfully executed when the relevant conditions are met.

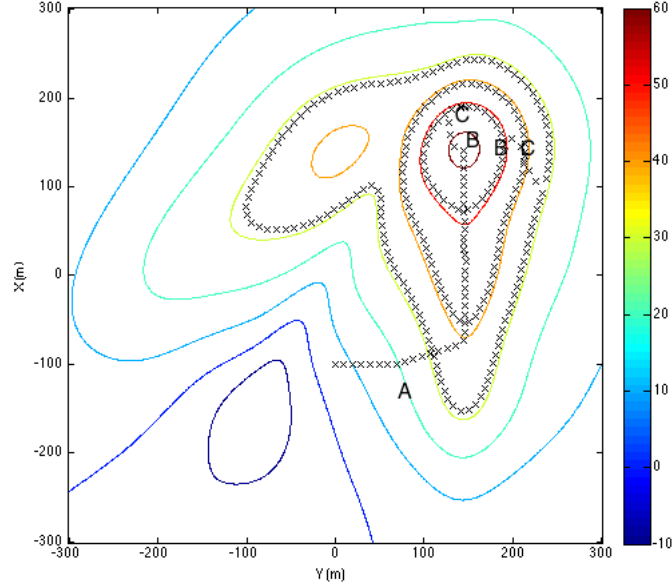


Fig. 3.12: A robot cluster mapping contours around a maximum in the field

In this example, the first transition criteria is met when the scalar level of the center robot passes 15 scalar units. As the cluster reaches the maximum all the scalar differentials on the third subplot of Figure 3.13 are above the margin value, initiating the next state. The fourth subplot demonstrates that the next transition occurs when the scalar level of the center robot is close enough to the desired contour level, and the rotational command is minimal. This is represented by plotting the rotational command, and the difference between the scalar value of the cluster and that of the contour line. Finally, the last plot compares the starting angle to the current angle, and the radius differential, indicating that the final transition occurs when the polar position is just past the starting value.

The controller is most susceptible to error at the point at which it determines whether it should switch to the next contour level, as the characteristics of the field can have

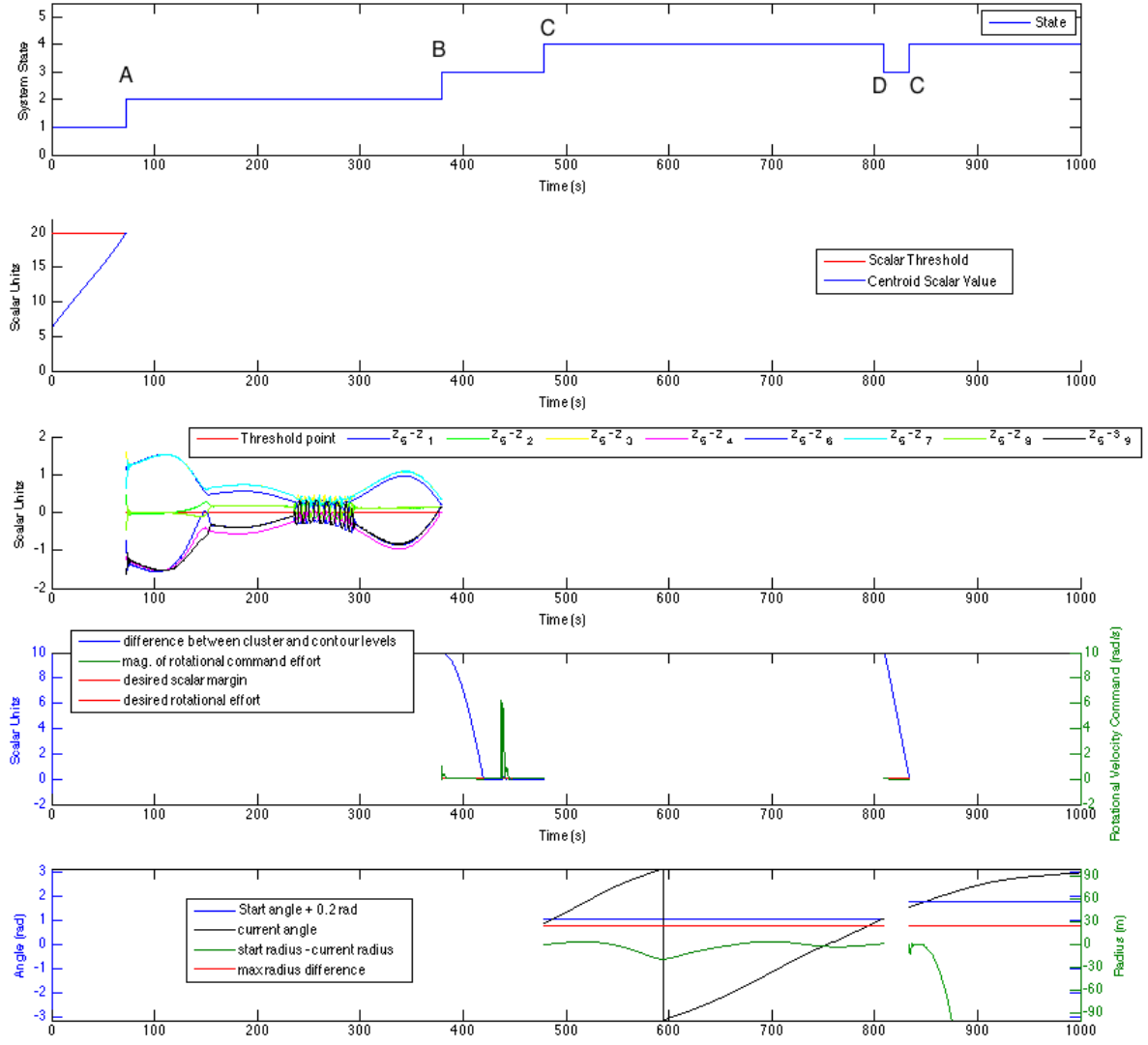


Fig. 3.13: Contour mapping time history containing state and transition information

a significant impact. Figure 3.14 is an example of a potential failure scenario. The spacing between contour lines is such that the robot cluster gets stuck on a nearby maximum instead of continuing downward. While only a small adjustment of the spacing is required to avoid this scenario, it could still be difficult to avoid in the field, due to unknown conditions. A final version of the controller may include additional measures to make it more resistant to this failure mode. All nine robots were required to confirm the presence of an extrema, however the rest of the application only requires three.

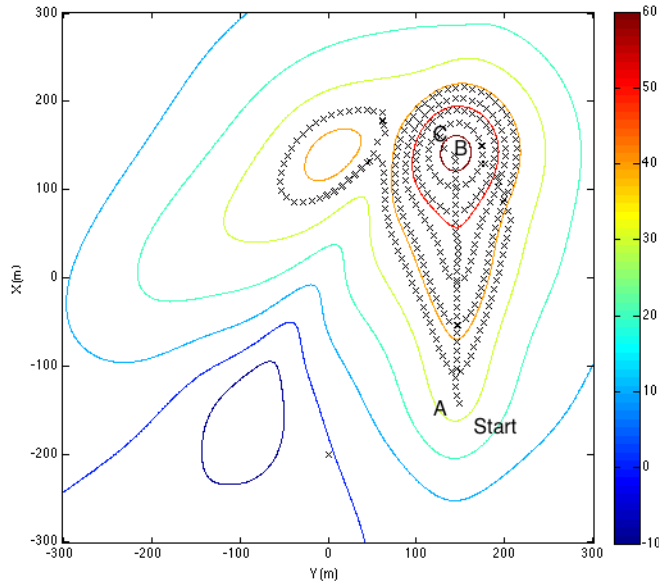


Fig. 3.14: A robot cluster gets stuck while mapping contours due to a secondary maximum

3.5 Maintaining Signal Strength

The objective of this controller is to keep the robot cluster above a particular scalar value as it navigates to a predetermined destination. This makes it a little different from previous techniques, as instead of navigating to an unknown destination, the end point is known, and it is the path that is adaptive. Remaining above a set sensor value has a variety of applications, including staying above safe thresholds. One example of great interest is maintaining a particular radio control signal strength, which would allow robots to complete their mission without fear of losing signal from the base of operations. The controller attempts to take a straight path to the destination, and when the cluster drops below a set scalar value, it switches to following the contour until the straight path is close enough to the direction of the gradient. Figure 3.5 displays the states and transitions required for this controller, and Table 3.4 presents the associated transition criteria.

The Toward Destination state simply takes a direct path to a predetermined destination. The associated cluster velocities are calculated using Equation 3.7, where x_{tar} and y_{tar} are the coordinates of the destination.

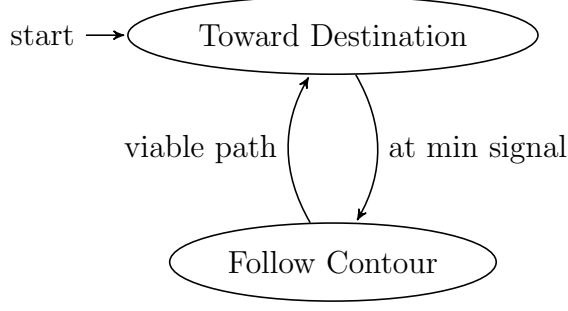


Fig. 3.15: State diagram for navigating to a point while maintaining signal strength

Table 3.4: Summary of Transition Criteria for Extrema Hopping

| Transition | Criteria |
|----------------------|---|
| A: 1 \rightarrow 2 | $z_c > z_{min}$ |
| B: 2 \rightarrow 1 | $\theta_{align} = \cos^{-1}(\hat{v}_{tar} \cdot \hat{g})$ |

$$v_{tar} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \cos \theta_c & \sin(\theta_c) \\ -\sin(\theta_c) & \cos(\theta_c) \end{bmatrix} \begin{bmatrix} \text{sign}(x_{tar} - x_c) \\ \text{sign}(y_{tar} - y_c) \end{bmatrix} \quad (3.7)$$

If the whole path were above the threshold, the robot cluster would simply follow a straight line all the way there. That said, if the scalar readings drop too low, the controller will change to the next state to avoid falling below the desired threshold.

The second state uses the contour following control primitive, with scalar values from robots 1, 3, 8, and 5. Upon entering this state, it determines whether to follow it clockwise or counter-clockwise by selecting the direction that is best aligned with a direct path to the target location. This is implemented by setting \dot{y}_c to the same direction as it was in the previous state.

This contour following controller continues until the direct path to the objective and the gradient of the scalar field are somewhat aligned. This alignment is calculated using Equation 3.8, which simply finds the angle between the gradient and the heading using the unit vecotors indicating the directions of the gradient and the target point.

$$\theta_{align} = \cos^{-1}(\hat{v}_{tar} \cdot \hat{g}) \quad (3.8)$$

When the desired direction of travel and the gradient are aligned, this indicates that

there is a viable path to the desired location that travels up the scalar surface, and will therefore provide a path that will take the robots above the desired scalar value. Once the criteria is met, the controller switches back into the first state.

The simulation demonstrated that the controller successfully kept the cluster above the threshold level, as seen in Figure 3.16. This path is a clear example of the cluster taking a straight path, then following the contour around until it finds a clear route to the destination, just as we would expect. Figure 3.17 contains time history plots with state and transition criteria information. From this data we can see that transitions are occurring as we would expect them to based upon the relevant scalar field information.

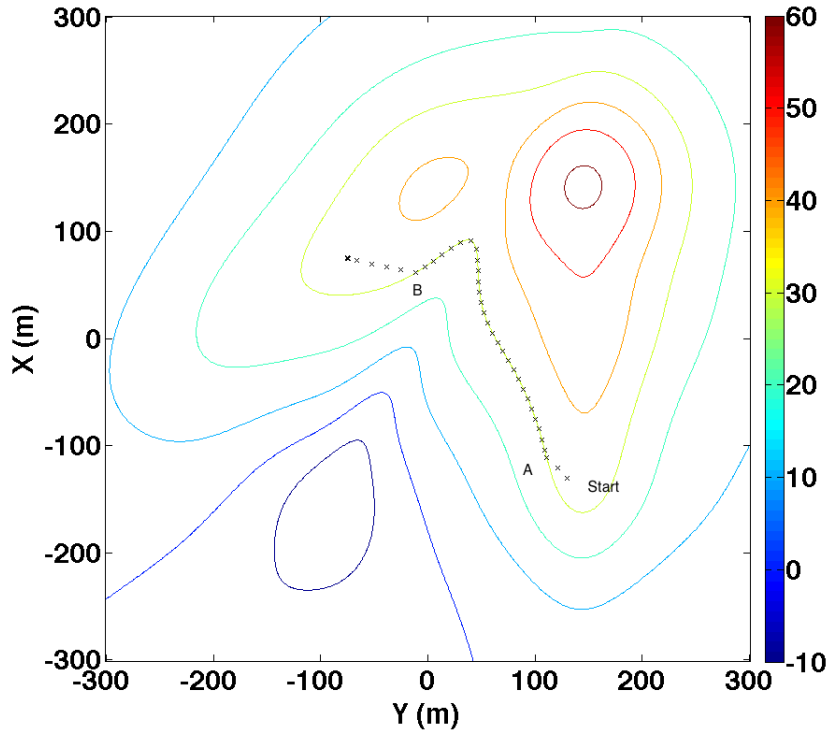


Fig. 3.16: A robot cluster navigating to $x = 75$, $y = -75$, using a state controller to remain above a scalar threshold

The first transition takes place when the scalar value for the cluster drops below the minimum allowed value, which is confirmed by the second subplot of Figure 3.17. The third plot displays the angle between the desired bearing and the gradient. As expected, the transition occurs when it reaches the set point.

While this is an example of success, there are certainly situations in which the controller would fail. A simple but significant example would be a scenario in which there isn't

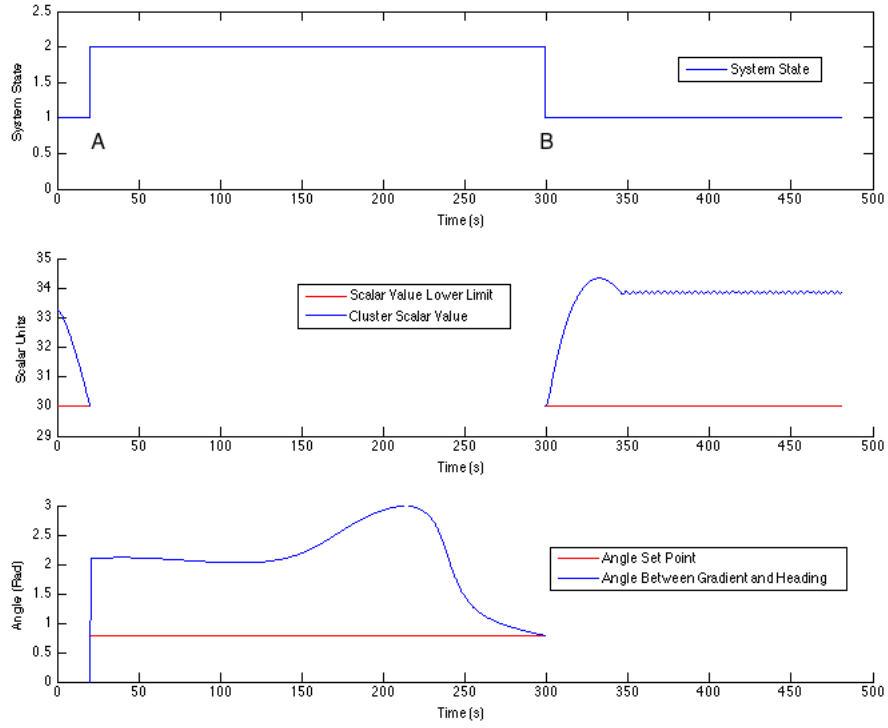


Fig. 3.17: Time response containing state and transition information for maintaing scalar magnitude

a viable path to the desired destination. Some of these scenarios will be discussed in section 3.6. While the simulation environment contained nine robots, this controller only made use of three of them.

3.6 Discussion

While these state-level controllers have been demonstrated to be functional in a simulated environment, there are several potential pitfalls associated with these methods. The largest causes for concern are the unknown nature of the field, and the potential for repetition.

3.6.1 Unknown Features

All of the controllers in this chapter make some assumptions of the field which is being navigated. First, there is the same assumption that is made for the individual feature controllers, which is that the field is continuous, and the feature size is navigable by the robots available. As before, this issue is largely a matter of sizing the cluster and tuning the controller to compensate for any gains introduced by the scalar field. The second assumption is that the feature set of interest is actually present. This second assumption poses a particular challenge for controlling robot clusters at the state-level, as each controller assumes a different set of field features is present, and that they meet a particular criteria.

3.6.2 Unwanted Repetition

As stated previously, the controllers do not store any data about the field. This means that the controller does not take into account where the cluster has been, and therefore would not know if it had been there before. This could lead to the cluster just traveling back and forth between two points, instead of continuing to explore the field. Similarly, as in Figure 3.14, there are situations where the cluster could be continuously following the same contour in a circle, and the controller would have no way of knowing. There are also a number of transitions between states that could be improved if there was more thorough knowledge of the areas that had already been traveled.

CHAPTER 4

Conclusion

4.1 Summary

The work presented in this thesis accomplished two primary objectives. First, the base set of primitive scalar field navigation controllers were completed, and second, they were combined to perform complex tasks.

The addition of the ridge, trench, and saddle navigation capabilities was an essential step toward expanding the adaptive navigation technique. Unlike the previous controllers, these primitives use differential sensing to generate the necessary information about the scalar field. This gives the cluster the ability to navigate with respect to the particular feature, rather than with respect to the field gradient. The performance of the controllers was successfully verified using a Matlab simulation. This additional information came at the cost of additional robots, as it requires five instead of three within a cluster. When added to extrema seeking and contour following, all the key features of interest within a scalar field can be navigated. To our knowledge, we are the first to propose such a comprehensive suite of primitive scalar field controllers [24].

While the individual primitive controllers perform valuable functions, it is possible to sequence them in order to create an enhanced set of capabilities. The first step in creating these enhanced capabilities was developing a state machine architecture to switch between the primitive controllers. Based on the application, this state machine switches between the individual control primitives using a defined set of criteria. Methods for moving from one extrema to another, mapping contours, and maintaining a magnitude were all implemented successfully in simulation. Primitive sequences and transition criteria were determined based upon the objectives of each application.

4.2 Future Work

While the work covered in this thesis represents a significant contribution to the capabilities of the adaptive navigation technique, there are many opportunities for establishing improved performance, enabling enhanced capabilities, verifying performance, et cetera. Three broad categories of work are adding memory to the state level exploration strategies, improving the effectiveness of the primitive controllers, and performing a series of field tests to verify the techniques in a uncontrolled environment.

4.2.1 Exploration with Memory

Adding memory to the adaptive navigation process can improve both the performance and the capabilities of the technique given that controllers could exploit knowledge about portions of the field through which the cluster has already navigated. The addition of memory to the controller can be used to prevent repetition/backtracking and to characterize the spatial characteristics of the field. The research and implementation of such techniques will require methods for efficient storage of field data and filtering algorithms appropriate to how previously measured field data can be exploited.

4.2.2 Adaptive Cluster Sizing

The size of the robot cluster relative to spatial characteristics of the field has a significant impact on the performance of the adaptive navigation controllers, both for estimating gradients and taking differential measurements. The aperture of the sensor distribution determines which spatial frequencies are attenuated, allowing execution of the primitive controllers to be tuned to spatial frequencies of interest. Up until now, we have assumed that the operators deploying the robot cluster have some idea of the frequencies of interest, and that they specify the cluster's size accordingly. This strategy is effective when there is some prior knowledge of the environment of interest; however, if an adaptive sizing policy is adopted, the cluster can actively optimize itself for the region it is currently traversing.

The ability to adaptively size the cluster would be beneficial for many applications. For instance, we could actively tune out spatial frequencies that are not of interest within current mission parameters, reducing the effective noise in the system. Varying the size

can also reveal previously unknown features. For example, the cluster may register as being on a flat surface, but as the size increases, it may become apparent that the flatter region was a part of a feature that was too large to detect with the original aperture size. For yet another example, if the cluster is seeking an extrema in a region, it may be beneficial to start with a low resolution configuration, and progressively increase resolution as the search progresses.

4.2.3 Additional Field Testing

Last but not least, all of these methods need to be fully verified in a field setting. The Robotic Systems Laboratory has multiple test platforms for multi-robot applications, all of which would be suitable for testing the results of both the research presented in this thesis and future adaptive navigation work. The first step will be to verify the ridge, trench and saddle controller, as it is required for the more complex exploration strategies. Based on the equipment available, it would make sense for the scalar field of interest to be bathymetric, a temperature distribution, or a radio frequency field generated by a designated antenna. The radio frequency field would be the simplest test environment for the ridge controller, as it is fairly simple to generate a ridge shape with a patch antenna [25].

Testing the exploration strategies would be more logistically difficult, as more complex fields would be required, with several key features. This would require the operators to locate a suitable naturally occurring field, or develop a method or medium for customizing an artificial scalar field. While conducting these tests will be more challenging, they will be an excellent verification of the techniques employed. This additional testing will aid in the refinement of the controllers, providing better performance in the future.

Bibliography

- [1] C. Kitts and M. Egerstedt, “Design, control, and applications of real-world multi-robot systems [from the guest editors],” *Robotics & Automation Magazine, IEEE*, vol. 15, pp. 8–8, March 2008. [1](#)
- [2] S. Jeon and J. Lee, “Multi-robot multi-task allocation for hospital logistics.,” *2016 18th International Conference on Advanced Communication Technology (ICACT)*, p. 339, 2016. [1](#)
- [3] M. A. Neumann and C. A. Kitts, “A hybrid multirobot control architecture for object transport,” *IEEE/ASME Transactions on Mechatronics*, vol. 21, pp. 2983–2988, Dec 2016. [1](#)
- [4] Y. Pei and M. W. Mutka, “Stars: Static relays for remote sensing in multirobot real-time search and monitoring,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 2079–2089, Oct 2013. [1](#)
- [5] X. Dai, H. Zhang, and Y. Shi, “Autonomous navigation for wheeled mobile robots-a survey,” in *Second International Conference on Innovative Computing, Information and Control (ICICIC 2007)*, pp. 551–551, Sept 2007. [1](#)
- [6] y. Yun-Won Choi¹, k. Kee-Koo Kwon¹, s. Soo-In Lee¹, c. Jeong-Won Choi², and s. Suk-Gyu Lee³, “Multi-robot mapping using omnidirectional-vision slam based on fisheye images.,” *ETRI Journal*, vol. 36, no. 6, pp. 913 – 923, 2014. [1](#)
- [7] B. C. Akdeniz and H. I. Bozma, “Exploration and topological map building in unknown environments,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1079–1084, May 2015. [1](#)
- [8] A. Melingui, T. Chettibi, R. Merzouki, and J. B. Mbede, “Adaptive navigation of an omni-drive autonomous mobile robot in unstructured dynamic environments.,” *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, p. 1924, 2013. [1](#)

- [9] S. Sharma and R. Tiwari, “A survey on multi robots area exploration techniques and algorithms,” in *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, pp. 151–158, March 2016. [1](#)
- [10] C. A. Kitts and I. Mas, “Cluster space specification and control of mobile multirobot systems,” *Mechatronics, IEEE/ASME Transactions on*, vol. 14, pp. 207–218, April 2009. [2](#), [3](#)
- [11] T. Adamek, C. Kitts, and I. Mas, “Gradient-based cluster space navigation for autonomous surface vessels,” *Mechatronics, IEEE/ASME Transactions on*, vol. 20, no. 1, pp. 506–518, 2015. [vii](#), [2](#), [4](#), [6](#), [8](#), [9](#)
- [12] A. Mahacek, “Autonomous cluster control of two robots in three dimensional space,” Master’s thesis, Santa Clara University, June 2014. [2](#)
- [13] I. Mas, S. Li, J. Acain, and C. Kitts, “Entrapment/escorting and patrolling missions in multi-robot cluster space control,” *Intelligent Robots and Systems. IEEE/RSJ International Conference on*, pp. 5855–5861, Oct 2009. [2](#)
- [14] J. Shepherd, *A Framework for Collaborative Multi-Task, Multi-Robot Missions*. Doctoral Thesis, Santa Clara University, Santa Clara, California, 2016. [2](#)
- [15] E. Burian, D. Yoerger, A. Bradley, and H. Singh, “Gradient search with autonomous underwater vehicle using scalar measurements,” in *Proceedings of the IEEE OES AUV conference*, 1996. [4](#)
- [16] C. Zhang, A. Siranosian, and M. Krstic, “Extremum seeking for moderately unstable systems and for autonomous vehicle target tracking without position measurements,” *AUTOMATICA*, vol. 43, no. 10, pp. 1832 – 1839, n.d. [4](#)
- [17] H. Ishida, T. Nakamoto, T. Moriizumi, T. Kikas, and J. Janata, “Plume-tracking robots: A new application of chemical sensors.,” *Biological Bulletin*, no. 2, p. 222, 2001. [4](#)
- [18] P. Ogren, E. Fiorelli, and N. Leonard, “Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment.,” *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, vol. 49, no. 8, pp. 1292 – 1302, n.d. [4](#)

- [19] E. Biyik and M. Arcak, “Gradient climbing in formation via extremum seeking and passivity-based coordination rules,” *ASIAN JOURNAL OF CONTROL*, vol. 10, no. 2, pp. 201 – 211, n.d. 4
- [20] R. Bachmayer and N. E. Leonard, “Vehicle networks for gradient descent in a sampled environment,” in *Proceedings of the 41st IEEE Conference on Decision and Control*, vol. 1, pp. 112–117, Dec 2002. 4
- [21] E. Fiorelli, N. Leonard, P. Bhatta, D. Paley, R. Bachmayer, and D. Fratantoni, “Multi-auv control and adaptive sampling in monterey bay,” *Oceanic Engineering, IEEE Journal of*, vol. 31, no. 4, pp. 935–948, 2006. 4
- [22] E. Fiorelli, P. Bhatta, N. Leonard, and I. Shulman, “Adaptive sampling using feedback control of an autonomous underwater gliderfleet,” in *Proc. Symp. Unmanned Untethered Submersible Technology*, pp. 1–16, 2003. 4
- [23] J. Acain, C. Kitts, T. Adamek, K. Ebadi, and M. Rasay, “A multi-robot testbed for adaptive sampling experimentation via radio frequency fields,” *ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications*, vol. 9, no. 1, 2015. 8
- [24] C. Kitts, R. McDonald, and M. Neumann, “Adaptive navigation primitives for multirobot clusters: Extrema finding, contour following, ridge/trench following, and saddle point station keeping,” *Submitted to IEEE Access*, 2017. 43
- [25] A. Mulcahy, “Radio frequency testbed for adaptive navigation,” *MS Capstone Report*, 2016. 45

Appendix A: Simulink Diagrams

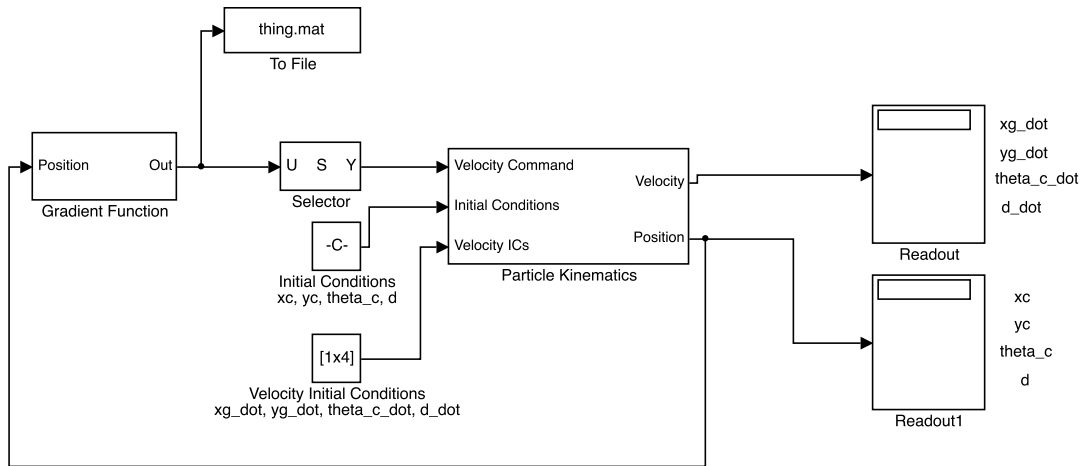


Fig. 1: Simulink block diagram used for all simulations.

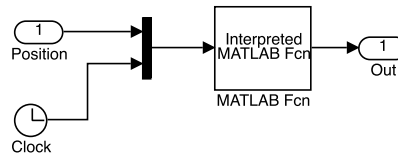


Fig. 2: This diagram is a subsection of Figure 4, and calls a '.m' file that changes based upon which controller is in use.

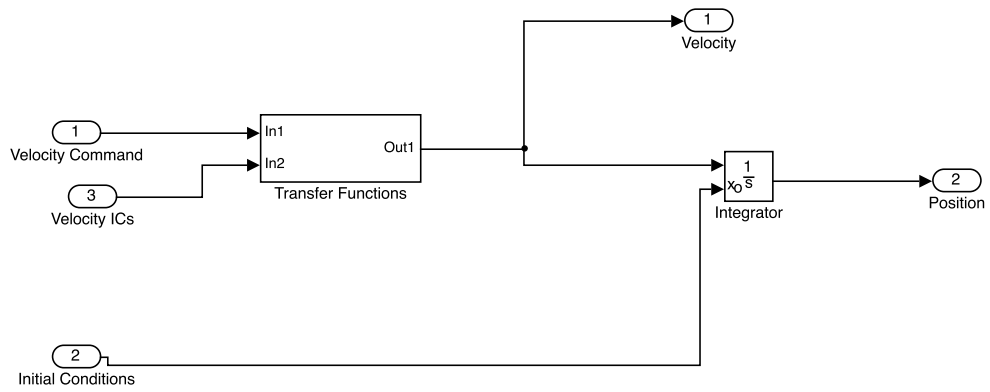


Fig. 3: This Simulink diagram is inside the kinematics block in Figure 4.

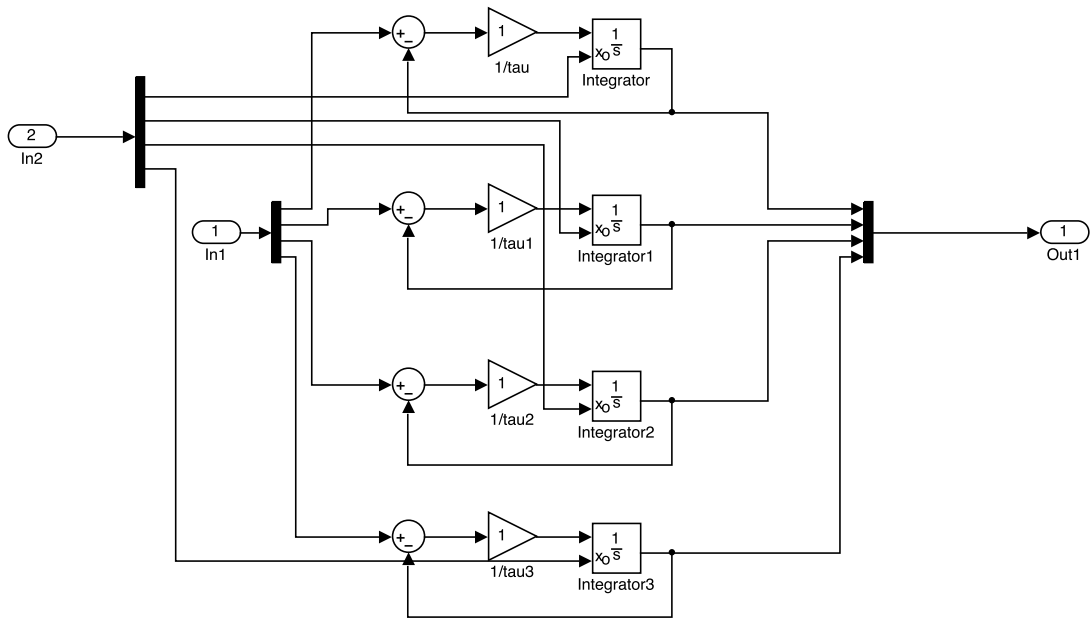


Fig. 4: This Simulink diagram is inside the transfer function block in Figure 3.

Appendix B: Matlab Functions

Ridge/Trench Matlab Function

```
function [Vcmd] = five_bot( position )
%FIVE_BOT
%Takes in the position and orientation of the cluster center and uses the
%information to determine where individual robots would be, find scalar
%values, and generate the desired cluster velocities. These cluster values
%are then transformed into the global velocities, which are fed out to the
%simulink model.
%
%State mode: This control method uses a differential drive
%strategy to stay on the ridge while moving down it.
%If the cluster detects that it has left the ridge, it switches into a
%different mode and climbs back up until it is on the ridge again.

xc = position(1);
yc = position(2);
theta_c = position(3);
d = position(4);
time = position(5);

field_shift_rate = [0 0 0 0];
field_shift = field_shift_rate*time;

%From cluster to global.
Rgc = [cos(theta_c), -sin(theta_c);
       sin(theta_c), cos(theta_c)];

%Cluster formation
%      1  2  3
%
```

```

%           4           5

%%Cluster 1 Variables (downhill cluster)%%
d1 = 5; %Distance from cluster center to bot

d2 = 5;
vmax = 2;

%Robot positions in global frame
r1_pos = Rgc*[0; d1] + [xc; yc];
r2_pos = Rgc*[0; 0] + [xc; yc];
r3_pos = Rgc*[0; -d1] + [xc; yc];
r4_pos = Rgc*[d2; d1] + [xc; yc];
r5_pos = Rgc*[d2; -d1] + [xc; yc];

%Scalar value
scalar1 = scalar_surface(r1_pos(2), r1_pos(1), field_shift);
scalar2 = scalar_surface(r2_pos(2), r2_pos(1), field_shift);
scalar3 = scalar_surface(r3_pos(2), r3_pos(1), field_shift);
scalar4 = scalar_surface(r4_pos(2), r4_pos(1), field_shift);
scalar5 = scalar_surface(r5_pos(2), r5_pos(1), field_shift);

%Robot vectors including scalar values
R1 = [r1_pos; scalar1];
R2 = [r2_pos; scalar2];
R3 = [r3_pos; scalar3];
R4 = [r4_pos; scalar4];
R5 = [r5_pos; scalar5];

%Gradient calcs (NOT USED IN THIS VERSION)
g1 = grad_calc(R1, R3, R4);
g4 = grad_calc(R4, R1, R2);
g2 = grad_calc(R2, R4, R3);
g3 = grad_calc(R3, R2, R1);

%Angles between gradients (NOT USED FOR THIS VERSION)
crab = 0;

```

```

d_dot = 0; %Can be adjusted for adaptive sizing (other changes required ...
    as well

%%CONTROL EQUATIONS%%
v2 = scalar1 - scalar3;
v1 = scalar4 - scalar5;

%%for ridge%%
%thetac_dot = -1*vmax/d2*sign(v2 - v1);

% %%for trench%%
    thetac_dot = 1*vmax/d2*sign(v2 - v1);

% %%ON RIDGE%%
% if (scalar2 > scalar3 + .01) && (scalar2 > scalar1 + .01)
%     xc_dot = sign(scalar1 + scalar3 - scalar4 - scalar5);
%     yc_dot = sign(scalar1 - scalar3 + scalar4 - scalar5);
%
% %%OFF RIDGE%%
% else
%     xc_dot = -1*sign(scalar1 + scalar3 - scalar4 - scalar5);
%     yc_dot = 1*sign(scalar1 - scalar3 + scalar4 - scalar5);
%
% end

%%ON Trench%%
if (scalar2 < scalar3 - .01) && (scalar2 < scalar1 - .01)
    xc_dot = -sign(scalar1 + scalar3 - scalar4 - scalar5);
    yc_dot = -sign(scalar1 - scalar3 + scalar4 - scalar5);

%%OFF Trench%%
else
    xc_dot = 1*sign(scalar1 + scalar3 - scalar4 - scalar5);
    yc_dot = -1*sign(scalar1 - scalar3 + scalar4 - scalar5);
end

%Transformation into global frame
vg_dot = Rgc*([xc_dot yc_dot] ');
xg_dot = vg_dot(1);
yg_dot = vg_dot(2);

```

```

radius = sqrt(xc^2 + yc^2); %was for performance testing

%Variable numbers can be used to plot the appropriate values
Vcmd = [xg_dot; %1
yg_dot; %2
thetac_dot; %3
d_dot; %4
field_shift_rate(1); %5
field_shift_rate(2); %6
field_shift_rate(3); %7
field_shift_rate(4); %8
xc; %9
yc; %10
theta_c; %11
d; %12
r1_pos(1); %13
r1_pos(2); %14
r2_pos(1); %15
r2_pos(2); %16
r3_pos(1); %17
r3_pos(2); %18
r4_pos(1); %19
r4_pos(2); %20
r5_pos(1); %21
r5_pos(2); %22
radius; %23 %Messed up the first one to make room
g2(2); %24
g3(1); %25
g3(2); %26
g4(1); %27
g4(2); %28
crab; %29
field_shift(1); %30
field_shift(2); %31
field_shift(3); %32
field_shift(4); %33
];
Vcmd = Vcmd';
end

```


Function for Moving from Extrema to Extrema

```
function [Vcmd] = nine_bot( position )
%NINE_BOT generates cluster commands that allow a virtual cluster of
%nine robots to climb up to a maximum, travel down a ridge into a saddle,
%then climb to reach another max

xc = position(1);
yc = position(2);
theta_c = position(3);
d = position(4);
time = position(5);
des_level = position(6);
state = position(7);
prev_state = position(8);

sens = .01;
start_time = 0;
field_shift_rate = [0 0 0 0];
field_shift = field_shift_rate*time;

%From cluster to global.
Rgc = [cos(theta_c), -sin(theta_c);
       sin(theta_c), cos(theta_c)];

%Cluster formation
%      1  2  3
%      4  5  6
%      7  8  9

%%Cluster 1 Variables (downhill cluster)%%
d1 = 5; %Distance from cluster center to bot

vmax = 2;

%Robot positions in global frame
r1_pos = Rgc*[-d1; d1] + [xc; yc];
r2_pos = Rgc*[-d1; 0] + [xc; yc];
r3_pos = Rgc*[-d1; -d1] + [xc; yc];
```

```

r4_pos = Rgc*[0; d1] + [xc; yc];
r5_pos = Rgc*[0; 0] + [xc; yc];
r6_pos = Rgc*[0; -d1] + [xc; yc];
r7_pos = Rgc*[d1; -d1] + [xc; yc];
r8_pos = Rgc*[d1; 0] + [xc; yc];
r9_pos = Rgc*[d1; d1] + [xc; yc];

%Scalar value
scalar1 = scalar_surface_old(r1_pos(2), r1_pos(1), field_shift);
scalar2 = scalar_surface_old(r2_pos(2), r2_pos(1), field_shift);
scalar3 = scalar_surface_old(r3_pos(2), r3_pos(1), field_shift);
scalar4 = scalar_surface_old(r4_pos(2), r4_pos(1), field_shift);
scalar5 = scalar_surface_old(r5_pos(2), r5_pos(1), field_shift);
scalar6 = scalar_surface_old(r6_pos(2), r6_pos(1), field_shift);
scalar7 = scalar_surface_old(r7_pos(2), r7_pos(1), field_shift);
scalar8 = scalar_surface_old(r8_pos(2), r8_pos(1), field_shift);
scalar9 = scalar_surface_old(r9_pos(2), r9_pos(1), field_shift);

R1 = [r1_pos; scalar1];
R2 = [r2_pos; scalar2];
R3 = [r3_pos; scalar3];
R4 = [r4_pos; scalar4];
R5 = [r5_pos; scalar5];
R6 = [r6_pos; scalar6];
R7 = [r7_pos; scalar7];
R8 = [r8_pos; scalar8];
R9 = [r9_pos; scalar9];

g1 = -grad_calc(R8, R3, R1);
g1_unit = g1/norm(g1);

d_dot = 0; %Can be adjusted for adaptive sizing

c1 = -d1 + 2*d1*(R5(3) - R1(3))/(R7(3) - R1(3));
c2 = -d1 + 2*d1*(R5(3) - R3(3))/(R9(3) - R3(3));
cc = (c2 - 0)/d1 - (0 - c1)/d1;

if time < 1
    state = 0;
    prev_state = 0;
end

```

```

if scalar5 > scalar1 + sens && scalar5 > scalar2 + sens && scalar5 > ...
    scalar3 + sens && scalar5 > scalar4 + sens && scalar5 > scalar6 + ...
    sens && scalar5 > scalar7 + sens && scalar5 > scalar8 + sens && ...
    scalar5 > scalar9 + sens
    on_max = true;
else
    on_max = false;
end

if state == 0
    %%'Patrolling' state. Straight line for now.%%
    xc_dot = 1;
    yc_dot = 0;
    thetac_dot = 0;
    if scalar5 > 10
        prev_state = state;
        state = 1;
    end

elseif state == 1
    %%Climbing state%%
    des_theta = atan2(g1_unit(2), g1_unit(1));
    thetac_dot = sign(des_theta - theta_c);
    d_dot = 0; %%Can be used to change the size of the cluster
    yc_dot = 0;
    xc_dot = 1;
    if on_max == true && prev_state == 0
        des_level = scalar5 - 1;
        prev_state = state;
        state = 2;

        start_time = time;
    end

elseif state == 2
    %%Following contour to find ridge%%
    des_theta = atan2(g1_unit(2), g1_unit(1));
    thetac_dot = (des_theta - theta_c);
    d_dot = 0; %%Used to change cluster size over time
    %%xc_dot = -sign(des_level - (R2(3) + R7(3) + R9(3))/3);
    xc_dot = sign(des_level - R5(3));
    yc_dot = 1;

```

```

    %if (abs(cc) > 4 && abs(cc) < 100) && xc_dot < .01 && time - ...
        start_time > 200
    if (abs(cc) > 4 && abs(cc) < 100) && abs(des_level - R5(3)) < .01 ...
        %&& time - start_time > 200
        prev_state = state;
        state = 3;
    end

elseif state == 3
    %%Following ridge until saddle is reached%%
    v2 = scalar1 - scalar3;
    v1 = scalar7 - scalar9;
    thetac_dot = -1*vmax/dl*sign(v2 - v1);

    if (scalar2 > scalar3 + .01) && (scalar2 > scalar1 + .01)
        xc_dot = sign(scalar1 + scalar3 - scalar7 - scalar9);
        yc_dot = -sign(scalar1 - scalar3 + scalar7 - scalar9);

    else
        xc_dot = -1*sign(scalar1 + scalar3 - scalar4 - scalar5);
        yc_dot = -1*sign(scalar1 - scalar3 + scalar7 - scalar9);
    end

    if scalar5 > scalar4 + .01 && scalar5 > scalar6 + .01 && scalar5 + ...
        .01 < scalar2 && scalar5 + .01 < scalar8
        prev_state = state;
        state = 1;
    end

else
    %%Just tells everything to stop if we are in an unknown state.%%
    xc_dot = 0;
    yc_dot = 0;
    thetac_dot = 0;
end

%Transformation into global frame
vg_dot = Rgc*([xc_dot yc_dot] ');
xg_dot = vg_dot(1);
yg_dot = vg_dot(2);

%Variable numbers can be used to plot the appropriate values
Vcmd = [xg_dot; %1

```

```

yg_dot; %2
thetac_dot; %3
d_dot; %4
field_shift_rate(1); %5
field_shift_rate(2); %6
field_shift_rate(3); %7
field_shift_rate(4); %8
xc; %9
yc; %10
theta_c; %11
r1_pos(1); %12
r1_pos(2); %13
r2_pos(1); %14
r2_pos(2); %15
r3_pos(1); %16
r3_pos(2); %17
r4_pos(1); %18
r4_pos(2); %19
r5_pos(1); %20
r5_pos(2); %21
r6_pos(1); %22
r6_pos(2); %23
r7_pos(1); %24
r7_pos(2); %25
r8_pos(1); %26
r8_pos(2); %27
r9_pos(1); %28
r9_pos(2); %29
cc; %30
des_level; %31
state; %32
prev_state; %33
];
Vcmd = Vcmd';
end

```

Function for Staying Above a Given Magnitude

```
function [Vcmd] = signal_strength( position )
```

```
xc = position(1);  
yc = position(2);  
theta_c = position(3);  
d = position(4);  
time = position(5);  
des_level = position(6);  
state = position(7);  
prev_state = position(8);  
prev_heading_x = position(9);  
prev_heading_y = position(10);  
start_time = position(11);  
angle = position(12);  
passed_once = position(13);  
x_peak = position(14);  
y_peak = position(15);
```

```
prev_heading = [prev_heading_x; prev_heading_y];  
sens = .01;  
level_drop = 5;
```

```
destination_x = 75;  
destination_y = -75;  
signal_limit = 30;
```

```
field_shift_rate = [0 0 0 0];  
field_shift = field_shift_rate*time;
```

```
%From cluster to global.
```

```
Rgc = [cos(theta_c), -sin(theta_c);  
       sin(theta_c), cos(theta_c)];
```

```
%Cluster formation
```

```

%           1   2   3
%           4   5   6
%           7   8   9

%%Supercluster Variables%%
dc = 5; %Distance from cluster center to supercluster center

%%Cluster 1 Variables (downhill cluster)%%
d1 = 5; %Distance from cluster center to bot

vmax = 2;

%%Cluster 2 Variables (uphill cluster)%%
%d2 = 5; %Distance from cluster center to bot

%Robot positions in global frame
r1_pos = Rgc*[-d1; d1] + [xc; yc];
r2_pos = Rgc*[-d1; 0] + [xc; yc];
r3_pos = Rgc*[-d1; -d1] + [xc; yc];
r4_pos = Rgc*[0; d1] + [xc; yc];
r5_pos = Rgc*[0; 0] + [xc; yc];
r6_pos = Rgc*[0; -d1] + [xc; yc];
r7_pos = Rgc*[d1; -d1] + [xc; yc];
r8_pos = Rgc*[d1; 0] + [xc; yc];
r9_pos = Rgc*[d1; d1] + [xc; yc];

%Scalar value
scalar1 = scalar_surface(r1_pos(2), r1_pos(1), field_shift);
scalar2 = scalar_surface(r2_pos(2), r2_pos(1), field_shift);
scalar3 = scalar_surface(r3_pos(2), r3_pos(1), field_shift);
scalar4 = scalar_surface(r4_pos(2), r4_pos(1), field_shift);
scalar5 = scalar_surface(r5_pos(2), r5_pos(1), field_shift);
scalar6 = scalar_surface(r6_pos(2), r6_pos(1), field_shift);
scalar7 = scalar_surface(r7_pos(2), r7_pos(1), field_shift);
scalar8 = scalar_surface(r8_pos(2), r8_pos(1), field_shift);
scalar9 = scalar_surface(r9_pos(2), r9_pos(1), field_shift);

R1 = [r1_pos; scalar1];
R2 = [r2_pos; scalar2];
R3 = [r3_pos; scalar3];

```

```

R4 = [r4_pos; scalar4];
R5 = [r5_pos; scalar5];
R6 = [r6_pos; scalar6];
R7 = [r7_pos; scalar7];
R8 = [r8_pos; scalar8];
R9 = [r9_pos; scalar9];

%g1 = -grad_calc(R2, R7, R9);
g1 = -grad_calc(R8, R3, R1);
g1_unit = g1/norm(g1);

d_dot = 0; %Can be adjusted for adaptive sizing

c1 = -d1 + 2*d1*(R5(3) - R1(3))/(R7(3) - R1(3));
c2 = -d1 + 2*d1*(R5(3) - R3(3))/(R9(3) - R3(3));
cc = (c2 - 0) - (0 - c1);

if time < 1
    state = 0;
    prev_state = 0;
    x_start = 1;
    y_start = 1;
end

if scalar5 > scalar1 + sens && scalar5 > scalar2 + sens && scalar5 > ...
    scalar3 + sens && scalar5 > scalar4 + sens && scalar5 > scalar6 + ...
    sens && scalar5 > scalar7 + sens && scalar5 > scalar8 + sens && ...
    scalar5 > scalar9 + sens
    on_max = true;
else
    on_max = false;
end

if state == 0
    %%Heading for destination.%%
    vel = Rgc^(-1)*([sign(destination_x - xc), sign(destination_y - ...
        yc)]');
    %xc_dot = sign(destination_x - xc);
    %yc_dot = sign(destination_y - yc);

```



```

xc_dot = sign(vel(1));
yc_dot = sign(vel(2));
thetac_dot = 0;
if scalar5 < signal_limit
    prev_heading = [sign(destination_x - xc), sign(destination_y - ...
        yc)]';
    prev_state = state;
    state = 2;
end

elseif state == 1
    %%Climbing state%%
    des_theta = atan2(g1_unit(2), g1_unit(1));
    thetac_dot = sign(des_theta - theta_c);
    d_dot = 0; %%Can be used to change the size of the cluster
    yc_dot = 0;
    xc_dot = 1;
    des_level = scalar5 - 1;
    if on_max == true && prev_state == 0
        des_level = scalar5 - level_drop;
        x_peak = xc;
        y_peak = yc;
        prev_state = state;
        state = 2;
        start_time = time;
    end

elseif state == 2
    %%Following contour to find a route%%
    vel = Rgc^(-1)*(prev_heading);
    des_theta = atan2(g1_unit(2), g1_unit(1));
    thetac_dot = (des_theta - theta_c);
    d_dot = 0; %%Used to change cluster size over time
    xc_dot = sign(signal_limit - R5(3));
    yc_dot = sign(vel(2));
    dest_vector = [(destination_x - xc), (destination_y - ...
        yc)]/norm([(destination_x - xc), (destination_y - yc)]);
    CosTheta = dot(dest_vector, [g1_unit(1), g1_unit(2)]);
    angle = acos(CosTheta);
    if abs(angle) < pi/4
        state = 0;
    end
end

```

```

elseif state == 3
    %%Following contour%%
    des_theta = atan2(g1_unit(2), g1_unit(1));
    thetac_dot = (des_theta - theta_c);
    d_dot = 0; %Used to change cluster size over time
    xc_dot = sign(des_level - R5(3));
    yc_dot = -1;
    %if (xc > x_start - 20) && (xc < x_start + 20) && (yc < y_start + ...
        20) && (yc > y_start - 20) && (time - start_time > 10)
    v = [xc - x_peak, yc - y_peak];
    angle = atan2(v(2), v(1));
    radius = norm(v);

    %if v_unit(1) < x_comp + .1 && v_unit(1) > x_comp - .1 && v_unit(2) ...
        < y_comp + .1 && v_unit(2) > y_comp - .1 && (time - start_time > ...
        10) &&(xc > x_start - 20) && (xc < x_start + 20) && (yc < ...
        y_start + 20) && (yc > y_start - 20)
    if angle > start_angle + .7
        passed_once = true;
    end
    if radius < start_radius + 25 && radius > start_radius - 25 && ...
        angle > start_angle + .2 && (time - start_time > 200) %&& ...
        passed_once == true
        des_level = des_level - level_drop;
        passed_once = false;
        prev_state = state;
        state = 2;
    end

elseif state == 4
    %%Following ridge until saddle is reached%%
    v2 = scalar1 - scalar3;
    v1 = scalar7 - scalar9;
    thetac_dot = -1*vmax/d1*sign(v2 - v1);

    if (scalar2 > scalar3 + .01) && (scalar2 > scalar1 + .01)
        xc_dot = sign(scalar1 + scalar3 - scalar7 - scalar9);
        yc_dot = -sign(scalar1 - scalar3 + scalar7 - scalar9);
    else
        xc_dot = -1*sign(scalar1 + scalar3 - scalar4 - scalar5);
    end

```

```

        yc_dot = -1*sign(scalar1 - scalar3 + scalar7 - scalar9);
    end

    if scalar5 > scalar4 + .01 && scalar5 > scalar6 + .01 && scalar5 + ...
        .01 < scalar2 && scalar5 + .01 < scalar8 + .01
        prev_state = state;
        state = 1;
    end
else
    %%Just tells everything to stop if we are in an unknown state.%%
    xc_dot = 0;
    yc_dot = 0;
    thetac_dot = 0;
end

%%Transformation into global frame
vg_dot = Rgc*([xc_dot yc_dot] ');
xg_dot = vg_dot(1);
yg_dot = vg_dot(2);

%%Variable numbers can be used to plot the appropriate values
Vcmd = [xg_dot; %1
        yg_dot; %2
        thetac_dot; %3
        d_dot; %4
        field_shift_rate(1); %5
        field_shift_rate(2); %6
        field_shift_rate(3); %7
        field_shift_rate(4); %8
        xc; %9
        yc; %10
        theta_c; %11
        r1_pos(1); %12
        r1_pos(2); %13
        r2_pos(1); %14
        r2_pos(2); %15
        r3_pos(1); %16
        r3_pos(2); %17

```

```

r4_pos(1); %18
r4_pos(2); %19
r5_pos(1); %20
r5_pos(2); %21
r6_pos(1); %22
x_peak; %23
y_peak(1); %24
angle; %25
passed_once; %26
start_time; %27
prev_heading(1); %28
prev_heading(2); %29
cc; %30
des_level; %31
state; %32
prev_state; %33
];
Vcmd = Vcmd';
end

```

Function for Mapping the Contours Around an Extrema

```
function [Vcmd] = contour_mapping( position )

xc = position(1);
yc = position(2);
theta_c = position(3);
d = position(4);
time = position(5);
des_level = position(6);
state = position(7);
prev_state = position(8);
start_angle = position(9);
start_radius = position(10);
start_time = position(11);
angle = position(12);
passed_once = position(13);
x_peak = position(14);
y_peak = position(15);

sens = .01;
level_drop = 10;

field_shift_rate = [0 0 0 0];
field_shift = field_shift_rate*time;

%From cluster to global.
Rgc = [cos(theta_c), -sin(theta_c);
       sin(theta_c), cos(theta_c)];

%Cluster formation
%      1  2  3
%      4  5  6
%      7  8  9

%%Cluster 1 Variables (downhill cluster)%%
dl = 5; %Distance from cluster center to bot

%Robot positions in global frame
```

```

r1_pos = Rgc*[-d1; d1] + [xc; yc];
r2_pos = Rgc*[-d1; 0] + [xc; yc];
r3_pos = Rgc*[-d1; -d1] + [xc; yc];
r4_pos = Rgc*[0; d1] + [xc; yc];
r5_pos = Rgc*[0; 0] + [xc; yc];
r6_pos = Rgc*[0; -d1] + [xc; yc];
r7_pos = Rgc*[d1; -d1] + [xc; yc];
r8_pos = Rgc*[d1; 0] + [xc; yc];
r9_pos = Rgc*[d1; d1] + [xc; yc];

```

```

%Scalar value

```

```

scalar1 = scalar_surface_old(r1_pos(2), r1_pos(1), field_shift);
scalar2 = scalar_surface_old(r2_pos(2), r2_pos(1), field_shift);
scalar3 = scalar_surface_old(r3_pos(2), r3_pos(1), field_shift);
scalar4 = scalar_surface_old(r4_pos(2), r4_pos(1), field_shift);
scalar5 = scalar_surface_old(r5_pos(2), r5_pos(1), field_shift);
scalar6 = scalar_surface_old(r6_pos(2), r6_pos(1), field_shift);
scalar7 = scalar_surface_old(r7_pos(2), r7_pos(1), field_shift);
scalar8 = scalar_surface_old(r8_pos(2), r8_pos(1), field_shift);
scalar9 = scalar_surface_old(r9_pos(2), r9_pos(1), field_shift);

```

```

R1 = [r1_pos; scalar1];
R2 = [r2_pos; scalar2];
R3 = [r3_pos; scalar3];
R4 = [r4_pos; scalar4];
R5 = [r5_pos; scalar5];
R6 = [r6_pos; scalar6];
R7 = [r7_pos; scalar7];
R8 = [r8_pos; scalar8];
R9 = [r9_pos; scalar9];

```

```

%g1 = -grad_calc(R2, R7, R9);
g1 = -grad_calc(R8, R3, R1);
g1_unit = g1/norm(g1);

```

```

d_dot = 0; %Can be adjusted for adaptive sizing

```

```

c1 = -d1 + 2*d1*(R5(3) - R1(3))/(R7(3) - R1(3));
c2 = -d1 + 2*d1*(R5(3) - R3(3))/(R9(3) - R3(3));
cc = (c2 - 0) - (0 - c1);

```

```

if time < 1
    state = 0;
    prev_state = 0;
end

if scalar5 > scalar1 + sens && scalar5 > scalar2 + sens && scalar5 > ...
    scalar3 + sens && scalar5 > scalar4 + sens && scalar5 > scalar6 + ...
    sens && scalar5 > scalar7 + sens && scalar5 > scalar8 + sens && ...
    scalar5 > scalar9 + sens
    on_max = true;
else
    on_max = false;
end

if state == 0
    %%'Patrolling' state. Straight line for now.%%
    xc_dot = 1;
    yc_dot = 0;
    thetac_dot = 0;
    if scalar5 > 20
        prev_state = state;
        state = 1;
    end

elseif state == 1
    %%Climbing state%%
    des_theta = atan2(g1_unit(2), g1_unit(1));
    thetac_dot = sign(des_theta - theta_c);
    d_dot = 0; %%Can be used to change the size of the cluster
    yc_dot = 0;
    xc_dot = 1;
    des_level = scalar5 - 1;
    if on_max == true && prev_state == 0
        des_level = scalar5 - level_drop;
        x_peak = xc;
        y_peak = yc;
        prev_state = state;
        state = 2;
        start_time = time;
    end
end

```

```

elseif state == 2
    %%Descending to correct contour%%
    des_theta = atan2(g1_unit(2), g1_unit(1));
    thetac_dot = (des_theta - theta_c);
    d_dot = 0; %Used to change cluster size over time
    xc_dot = sign(des_level - R5(3));
    yc_dot = -1;
    if abs(des_level - scalar5) < .1 && time - start_time > 100 && ...
        abs(thetac_dot) < .1
        v = [xc - x_peak, yc - y_peak];
        start_angle = atan2(v(2), v(1));
        start_radius = norm(v);
        start_time = time;
        prev_state = state;
        state = 3;
    end

elseif state == 3
    %%Following contour%%
    des_theta = atan2(g1_unit(2), g1_unit(1));
    thetac_dot = (des_theta - theta_c);
    d_dot = 0; %Used to change cluster size over time
    xc_dot = sign(des_level - R5(3));
    yc_dot = -1;
    v = [xc - x_peak, yc - y_peak];
    angle = atan2(v(2), v(1));
    radius = norm(v);

    if angle > start_angle + .7
        passed_once = true;
    end
    if radius < start_radius + 25 && radius > start_radius - 25 && ...
        angle > start_angle + .2 && (time - start_time > 200) %&& ...
        passed_once == true
        des_level = des_level - level_drop;
        passed_once = false;
        prev_state = state;
        state = 2;
    end

else

```



```

%%Just tells everything to stop if we are in an unknown state.%%
xc_dot = 0;
yc_dot = 0;
thetac_dot = 0;
end

%Transformation into global frame
vg_dot = Rgc*([xc_dot yc_dot] ');
xg_dot = vg_dot(1);
yg_dot = vg_dot(2);

%Variable numbers can be used to plot the appropriate values
Vcmd = [xg_dot; %1
        yg_dot; %2
        thetac_dot; %3
        d_dot; %4
        field_shift_rate(1); %5
        field_shift_rate(2); %6
        field_shift_rate(3); %7
        field_shift_rate(4); %8
        xc; %9
        yc; %10
        theta_c; %11
        r1_pos(1); %12
        r1_pos(2); %13
        r2_pos(1); %14
        r2_pos(2); %15
        r3_pos(1); %16
        r3_pos(2); %17
        r4_pos(1); %18
        r4_pos(2); %19
        r5_pos(1); %20
        r5_pos(2); %21
        r6_pos(1); %22
        x_peak; %23
        y_peak(1); %24
        angle; %25
        passed_once; %26

```

```
start_time; %27
start_angle;%28
start_radius; %29
cc; %30
des_level; %31
state; %32
prev_state; %33
];
Vcmd = Vcmd';
end
```