5-28-2019

Swarm-Based Techniques for Adaptive Navigation Primitives

Nathan Metzger

Santa Clara University

Department of Mechanical Engineering

Date: May 28th, 2019

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Nathan Metzger

ENTITLED

Swarm-Based Techniques for Adaptive Navigation Primitives

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

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

[illegible]

Thesis Advisor

Dr. Christopher Kitts

Robert A. Marks

Thesis Reader

Dr. Robert Marks

[illegible]

Chairman of Department

Dr. Drazen Fabris

Swarm-Based Techniques for Adaptive Navigation Primitives

By

Nathan Metzger

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, 2019

Santa Clara, CA

Swarm-Based Techniques for Adaptive Navigation Primitives

Nathan Metzger

Department of Mechanical Engineering
Santa Clara University
Santa Clara, CA
2019

ABSTRACT

Adaptive Navigation (AN) has, in the past, been successfully accomplished by using mobile multi-robot systems (MMS) in highly structured formations known as clusters. Such multi-robot adaptive navigation (MAN) allows for real-time reaction to sensor readings and navigation to a goal location not known *a priori.* This thesis successfully reproduces MAN cluster techniques via swarm control techniques, a less computationally expensive but less formalized control technique for MMS, which achieves robot control through a combination of primitive robot behaviors. While powerful for large numbers of robots, swarm robotics often relies on "emergent" swarm behaviors resulting from robot-level behaviors, rather than top-down specification of swarm behaviors. For adaptive navigation purposes, it was desired to be able to specify swarm-level behavior from a top down perspective rather than experimenting with emergent behaviors. To this end, a simulation environment was developed to allow rapid development and vetting of swarm behaviors while easily interfacing with an existing testbed for validation on hardware. An initial suite of robot primitive and composite behaviors was developed and vetted using this simulator, and the behaviors were validated using the existing testbed in Santa Clara University's Robotics System Laboratory (RSL). Of particular importance were the adaptive navigation primitives of extrema finding and contour finding and following. These AN primitives were tested over a variety of experimental parameters, yielding design guidelines for top-down specification of swarm robotic adaptive navigation. These design guidelines are presented, and their usefulness is demonstrated for a Contour Finding and Following application using the RSL's testbed. Finally, possible future work to expand the capability of swarm-based adaptive navigation techniques is discussed.

Acknowledgements

First and foremost, I would like to thank my advisor Dr. Christopher Kitts for his constant guidance, support, and inspiration throughout this process. His undergraduate Controls course (as well as later graduate courses) initially sparked my interest in robotics. The fact that this research grew from a class project in one of those graduate courses speaks to his ability to blend new research into the classroom as well as his ability to encourage students to pursue new avenues within that research.

Additionally, I would like to thank my collaborators in this research, Max Reese and Shae Hart, whose constant insight and hard work cannot be understated. I would also like to thank all those around the lab who were always willing to offer suggestions and help work through roadblocks, including (but certainly not limited to) Dr. Michael Neumann, Robert McDonald, and Danop Rajabhandharaks. I would also like to thank all of my predecessors in the lab whose work made this research possible.

I would also like to thank my family, friends, girlfriend Maggie, and all others who politely smiled and nodded after asking what my research was about.

A portion of this work was submitted for publication [49].

List of Figures

List of Tables

Table P1. List of Variables

Variable Name	**Notation**	**Architecture level**
Current Robot State	(X, Y, Θ, SV)	Swarm Level
Bearing Command for Robot *i*	Φi	Omnibot Level
Velocity vector command for Robot *i*	$\overrightarrow{V_{\imath_{cmd}}}$	Omnibot Level
Angular velocity for wheels on Robot *i*	$\overrightarrow{\omega_{\imath_{cmd}}}$	Omnibot Level
Velocity vector, output, for Robot *i*	$\overrightarrow{V_{\imath_{out}}}$	Omnibot Level
Number of Robots	N	User Interface
Simulation Run Time	T_{sim}	User Interface
Sensor Range	R_s	User Interface
Avoidance Range	R_a	User Interface
Desired Contour Value	SV_{des}	User Interface
Contour Value “Buffer	SV_{buff}	User Interface
Robot Speed (Magnitude)	M_v	User Interface
Velocity vector, output, for Behavior *m* for Robot *i*	$\overrightarrow{V_{m\imath}}$	Behavior Level
Velocity vector, output, go-to Behavior for Robot *i*	$\overrightarrow{V_{g\imath}}$	Go-To Behavior
Velocity vector, output, Attract Behavior for Robot *i*	$\overrightarrow{V_{a\imath}}$	Attract Behavior
Velocity vector, output, Disperse Behavior for Robot *i*	$\overrightarrow{V_{d\imath}}$	Disperse Behavior

Velocity vector, output, flocking behavior for robot *i*	$\overrightarrow{V_{fl\imath}}$	Flocking Composite Behavior
Magnitude of Obstacle Avoidance Behavior Vector for Robot *i*	M_{oi}	Obstacle Avoidance Behavior
Velocity vector, output, Obstacle Avoidance Behavior for Robot *i*	$\overrightarrow{V_{O\imath}}$	Obstacle Avoidance Behavior
Distance from Robot *i* to Robot *j*	d_{ij}	Behavior Laws
Bearing from Robot *i* to Robot *j*	ϕ_{ij}	Behavior Laws
Weighting Gain for Behavior *m*	K_m	Behavior Laws
Sensor Value difference from robot *i* to *j*	ΔSV_{ij}	Sensor Value Comparison primitive
Find Min/Find Max vector from robot *i* to robot *j*	v_{fmij}	Find Min/Find Max Vector
Velocity vector, output, Find Min/Find Max Behavior for robot *i*	$\mp\vec{V}_{fmi}$	Find Min/Find Max Behavior

Table of Contents

1. Introduction

1.1 Robotics, Mobile Robotics, and Mobile Multi-Robot Systems (MMS)

Robotics, primarily in the form of industrial manipulators similar to the articulated arm shown in Figure 1, have had a large impact on manufacturing due their ability to execute tasks with speed and repeatability [1]. Initial applications, and the majority of current applications, consist of some sort of actuation (such as picking and placing parts, welding, and screw driving) in which the end-effector of the robot has some sort of mechanical device capable of actuating parts in the environment, as shown in Figure 1. Recent applications may choose to replace the mechanical end-effector with a sensing suite, shown in Figure 2.

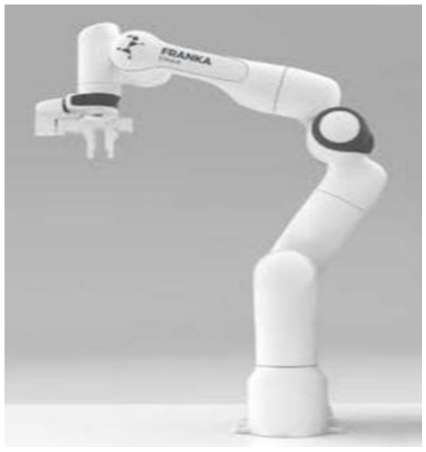


Figure 1: Franka Industrial Manipulator w/gripper [2]

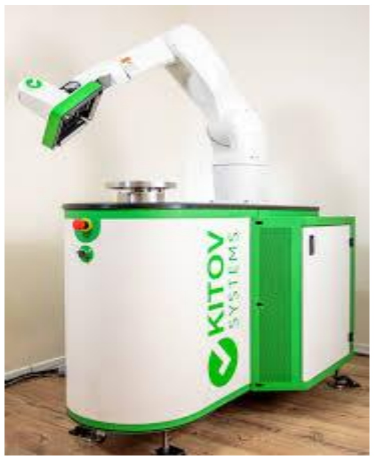


Figure 2: Kitov System - Denso Robot w/ Camera [3]

In Figure 2, rather than having a mechanical gripper like the Franka arm in Figure 1, the robot has an attached high-definition camera. Images from this camera can be used in a vision system to check parts for failures or defects [3]. These types of applications are likely to become more common in the so-called “fourth Industrial Revolution,” a term applied to the movement from automated physical systems to “Smart” Cyber-Physical Systems that incorporate larger numbers of sensors for robustness and predictability of manufacturing processes. [4]

In parallel to the development of industrial manipulators has been the development of mobile robotics. In some cases, mobile robots have industrial manipulators attached (Fetch Mobile Manipulator, Figure 3), which makes the industrial manipulator mobile via its base. In others, the primary role of the mobile robot is as a mobile sensor suite that can take measurements and gather data about its environment (NASA Mars Rover: Opportunity, Figure 4). These mobile robots offer a way to fulfill tasks that industrial manipulators cannot, such as material transport and sensing of a larger environment outside of the industrial manipulator's workspace.

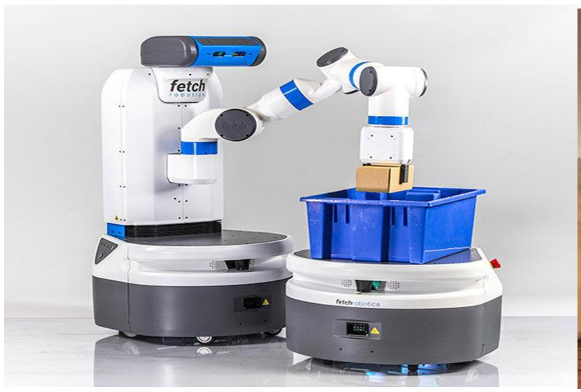


Figure 3. Fetch Mobile Manipulator [5]

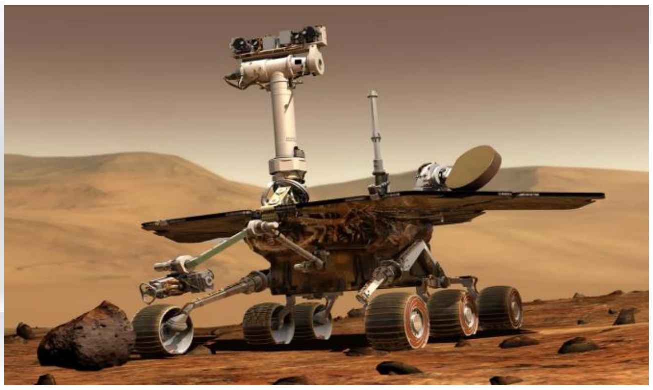


Figure 4. NASA Opportunity, a mobile sensor suite [6]

Growing out of the field of mobile robotics is the field of Mobile Multi-Robot Systems (MMS's). MMS offers many of the same benefits as mobile robotics, while also providing improvements upon applications using a single robot. MMS's utilize multiple smaller, resource limited robots to achieve similar results as single mobile robots. Using an MMS instead of a single large mobile robot can provide a few advantages. First, the individual cost of the robots is reduced, as complexity is often significantly reduced for the individual robots in an MMS. Second, MMS systems can be easily designed to be robust to individual robot failures, so that the success or failure of a mission does not depend on the successful operation of one individual robot. Additionally, there are inherently distributed tasks that require multiple agents, or tasks that are too complex for single robots to complete. Finally, certain applications benefit from the use of multiple distributed robots or sensors rather than one individual robot. Figures 5 and 6 show two examples of MMS, with Harvard's Kilobot swarm of small mobile robots and SCU's Decabot system, featuring more robust individual robots.

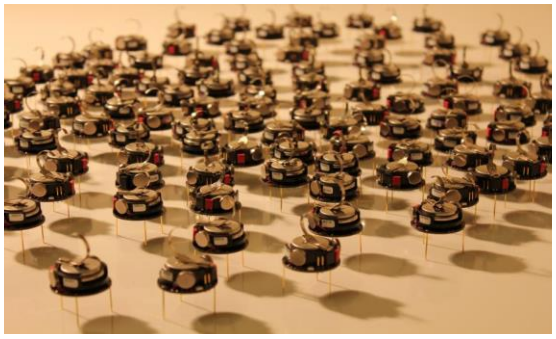


Figure 5. Harvard's Kilobot swarm MMS [7]

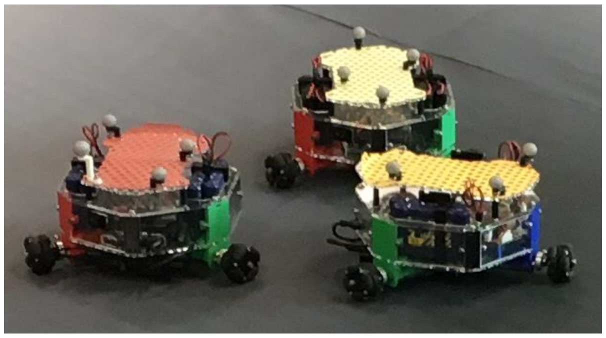


Figure 6. Robots in SCU's Decabot MMS system

1.1.1 Classification of MMS Applications

MMS applications, at a high level, can be categorized mainly by the two following classifications: Heterogeneous robots vs. Homogeneous Robots, and Collaborative vs. Parallel applications. MMS applications involving heterogeneous robots may involve multiple *types* of robots, each performing their own function within the application. For example, in a warehouse, a team of 3 robots could be tasked to gathering materials: one tasked to locating the desired material and broadcasting its location, and another, possibly equipped with a robotic manipulator, picks and places the materials onto a third "Sherpa" robot, which can then transport the materials to a desired location. In contrast to this, a homogeneous MMS system consists of robots that all have roughly the same capabilities, sensor suites, and actuators. For example, a group of homogeneous robots engaged in bathymetric mapping would all be using the same set of actuators and sensors to complete their task.

The second classification is between collaborative and parallel applications. As an example, consider the previous bathymetric mapping application. An MMS system that acts in parallel could send out a team of *N* robots that each have their own area to map out, with the aggregate of their areas fully mapping the desired location. This offers an advantage over a single-robot system that would take much longer to map out the area, simply because there is less area for each individual robot to map. These robots act almost completely independently of each other during operation, and as such are said to act in parallel. In contrast, robots trying to find the maximum depth of the lake floor would constantly be sharing data and comparing with each other in order to move towards that maximum depth, and as such would be acting collaboratively. While there has been research done in multiple areas regarding MMS, this thesis will focus on reviewing research done in homogeneous, collaborative systems.

Of particular interest in the scope of homogeneous, collaborative systems is the idea of adaptive navigation. Whereas conventional navigation involves guiding a vehicle along a predefined path or trajectory, adaptive navigation utilizes environmental data to move towards a target location not known *a priori*. While adaptive navigation for a single robot may be achieved through approaches such as SLAM (Simultaneous Location and Mapping), many approaches to adaptive navigation utilize MMS in order to exploit the collection of simultaneously collected distributed information in order to make navigation decisions. This will be discussed in more depth in section 1.2.

1.1.2 Formation Control for Mobile Multi-Robot Systems

One of the first issues that is addressed within MMS is how to coordinate the motion of multiple agents, often called formation control. Within the field of MMS there are many different approaches to formation control of the group of robots, which can be broadly classified as explicit formation control and implicit formation control.

1.1.2.1 Explicit Formation Control

Explicit formation control strictly specifies where robots are to be located in relation to each other. Typical examples of these are the Leader-Follower model, and an "expanded" version called Leader-Follower chains. These frameworks are illustrated in Figures 7-10. In a simple Leader-Follower (LF) framework, the leader moves with some direction, and the follower moves at some specified distance and bearing behind the leader (Figure 7). This can be expanded to multiple followers, so that multiple followers follow a single leader. Figure 8 depicts a scenario in which there is one leader, and multiple followers at different bearings, while Figure 9 depicts another possible scenario, where robots are given the same bearing angle and different distances. These are explicit frameworks for control as they distinctly specify where follower robots should be located in relation to the leader robot at any point in time.

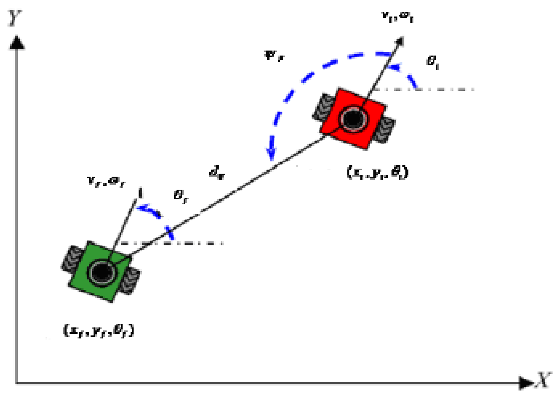


Figure 7. LF formation control for 2 robots [9]

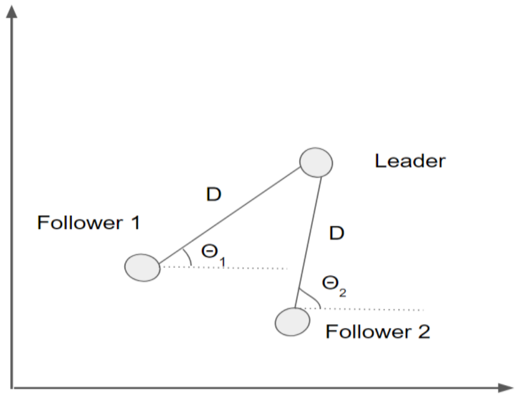


Figure 8. LF for one distance, multiple bearing angles

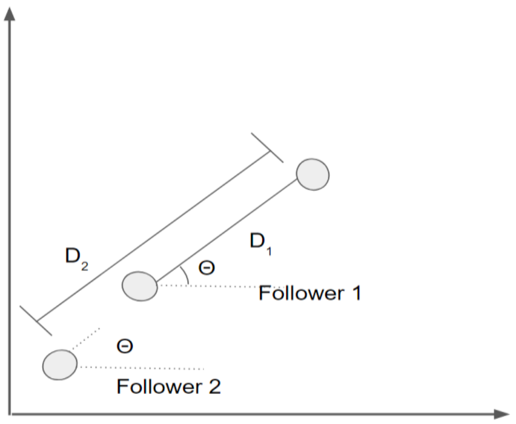


Figure 9. LF for one bearing angle, multiple distances

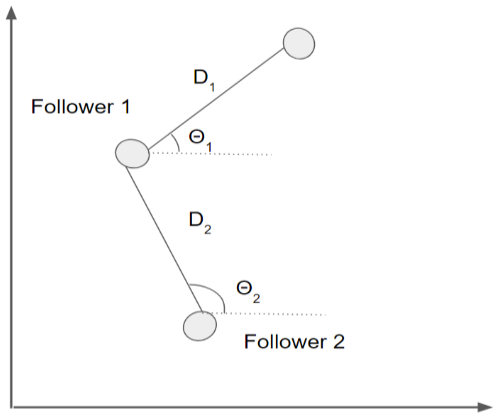


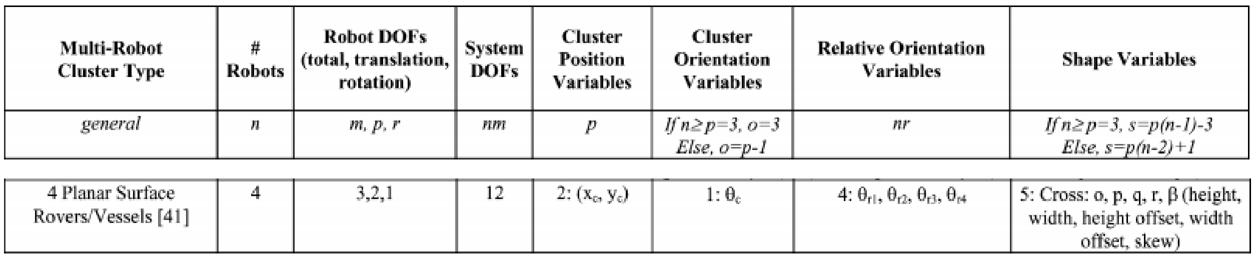
Figure 10. 3-robot LF chain

Another expanded version of this is the Leader-Follower chain, where each follower is then a leader for the robot behind it. This is shown in Figure 10 above. Finally, virtual “leaders” can be specified, so that the follower robot follows a given point, where a physical robot may not actually exist. Though leader-follower organization and leader-follower chains allow an easy way to specify a formation, there are challenges to the approach. Leader-follower chains can be prone to oscillation given delays in communication along the chain, and at large following distances, any requirement to maintain position based on the orientation of the leader can result in unachievable velocities.

Another method of explicit formation control is the Cluster-space formation [10]. In the cluster space, a group of N robots form a “cluster,” which is specified as a combination of geometry, position, and orientation variables. By using clusters, position and orientation can be explicitly specified at any point in time by specifying the motion of the cluster frame. The cluster frame is specified as a function of the robots, such as the centroid of the formation. Individual robots in the cluster may have their position and orientation specified in relation to the cluster frame, and motion of the individual robots can therefore be specified by the movement of the cluster as a whole via the Jacobian and inverse Jacobian relationships of the cluster. The cluster offers the benefit of explicit formation control and allows more sophisticated formations than simple leader-follower or leader-follower chains. This type of formation control is especially important in certain adaptive navigation scenarios, such as ridge or trench following, where relative location of environmental readings is important in order to determine characteristics of the scalar field. The two main challenges that arise when using the cluster space formulation are scalability (due to computational complexity) and singularity avoidance (although several approaches have been developed to address this).

Size limitations arise as a result of cluster configurations. For the example of 3-DOF planar robots (translational DOF’s in x and y, rotational about z), the total number of cluster variables required for an N-robot cluster will be 3N. Cluster variables are split up into four categories: cluster position, cluster orientation, relative robot orientation, and “shape” variables. Table 1 below, presented in [10], indicates the cluster space formulation in general and for 4 planar rovers:

Table 1. Cluster-Space Specification variables for MMS [10]



Multi-Robot Cluster Type	# Robots	Robot DOFs (total, translation, rotation)	System DOFs	Cluster Position Variables	Cluster Orientation Variables	Relative Orientation Variables	Shape Variables
general	*n*	*m, p, r*	*nm*	*p*	*If $n \geq p=3$, $o=3$ Else, $o=p-1$*	*nr*	*If $n \geq p=3$, $s=p(n-1)-3$ Else, $s=p(n-2)+1$*
4 Planar Surface Rovers/Vessels [41]	4	3,2,1	12	2: (x_c, y_c)	1: θ_c	4: $\theta_{r1}, \theta_{r2}, \theta_{r3}, \theta_{r4}$	5: Cross: o, p, q, r, β (height, width, height offset, width offset, skew)

The cluster position and orientation variables are functions of the individual robot degrees of freedom, while relative robot orientations are a function of the rotational degrees of freedom of the individual robots and the number of robots in the cluster. As a result, the cluster variable most affected by scaling the number of robots is the shape variables, which are $s= p(n-2) +1$. For example, 10 planar robots will have 17 shape variables. The specification and regulation of all variables leads to the cluster specification being computationally expensive, particularly when the number of robots in the cluster increases.

Similar to singularities in industrial manipulator robots, singularities occur in clusters at certain cluster configurations that cause the Jacobian or its inverse to become singular. It should also be noted that, similar to industrial manipulators, operating in the vicinity of singularities is undesirable as it causes very high actuator rates. In the context of clusters, this usually means that a certain "cluster motion" specification would require a robot velocity that is at or above the robot's physical capabilities. As such, the cluster specification is a powerful tool for multi-robot location specification, but has certain drawbacks, particularly in relation to easy scalability of the cluster.

1.1.2.2 Implicit Formation Control

While explicit formation control explicitly specifies the location of individual robots, either in relationship to each other or in global coordinates, implicit formation control allows for a "bottom-up" emergence of a formation as a result of individual robot behaviors or control laws. Implicit control includes techniques such as artificial potential fields (APF), behavior based robotics, and swarm control techniques. In APF, potential fields are artificially created around certain points so that the robot will move down the potential field in the desired manner. Figures 11 and 12 show a single potential field and the result of super-imposed potential fields. The combined potential field and a potential path down it shows how this may be used for mobile robot navigation. Superposition of global APF with "mobile" APF affixed to each individual robot leads to an "implicit" formation that balances each of the individual potential fields.

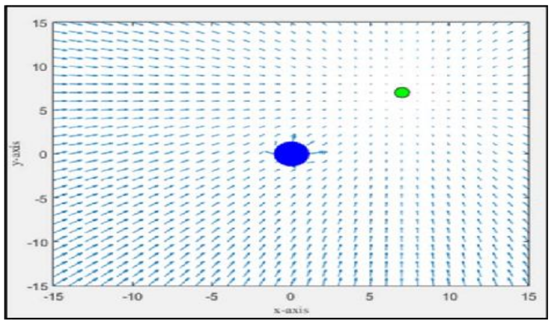


Figure 11. Single repulsive APF [11]

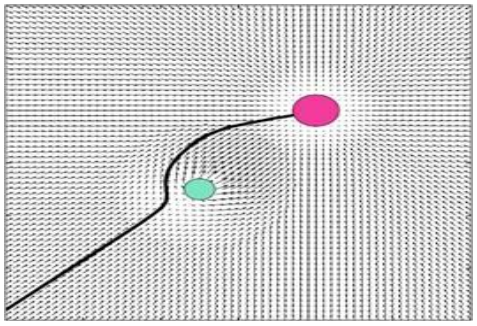


Figure 12. Superposition of APF, expected robot path [12]

This approach could also be accomplished through behavior-based robotics. However, in behavior based-robotics, the resultant behavior would be a result of individual behaviors rather than individual potential fields. For example, the first behavior could be “go east,” a second could be “avoid obstacle *X*” and a third be “avoid other robots.” In this way, a similar implicit formation could be achieved through a different formulation. While it is not explicitly different from the first two approaches mentioned here, swarm control techniques bear mentioning as their own approach to implicit formation control. The main difference between swarm robotics and the approaches mentioned above is that swarm behaviors are geared at very large-scale MMS. As a result of this much larger scale, swarm behaviors often attempt to execute the simplest form of behaviors, rather than more complex behaviors such as APF. Different approaches to swarm control techniques and example applications will be discussed in the next section.

1.1.3 Robotic Swarms and Research Fields within Robotic Swarms

As was mentioned in section 1.1.2, swarms often employ very simple individual robot behaviors and control laws, and rely on the scale of the MMS (the “swarm”) to achieve complex group behaviors. A more detailed classification of swarm robotics is as follows, adapted from Senanyake et al. [13] In general, a rough set of criteria for swarm robotics is as follows:

1. Swarms consist of autonomous robots, capable of sensing and actuating. As a result, stationary sensor networks or pure software agents are not “swarms.”
2. Swarms are large. Swarm behaviors can be studied at a small scale, but should be made with an aim towards scalability and large numbers of agents.
3. Swarms are homogeneous and redundant. While having heterogeneous robots does not exclude an MMS from being a swarm, the heterogeneity must be very limited so that the swarm is redundant and robust to individual robot failures.
4. Swarms improve performance through cooperation. In particular, the swarm robots would not individually be able to complete the task at hand (or would do it very inefficiently) outside of the context of the swarm. This is sometimes referred to as “swarm intelligence,” or the idea that the swarm is able to “know” things that individual robots themselves do not.
5. Swarm robots do not have extensive individual capabilities. Swarm robots should have limited sensing and communication abilities in an attempt to implement more distributed coordination.

Within the criteria above, there are many different foci in research involving swarms. Some of these research areas are applications based (such as mapping and localization, source seeking, and object transportation), some are focused on a particular design approach (such as bio-inspiration), and others are focused on swarm protocols (such as communication, control protocols, motion coordination, and "learning" *via* swarm intelligence) [14]. These areas are not necessarily exclusive. For example, certain groups may utilize bio-inspiration (design approach) for their communication protocols (swarm protocols) for swarms engaged in source seeking (application-based). Each of these areas is briefly discussed below, with references to studies in each area.

1.1.3.1 Bio-Inspired Swarms

Bio-inspired swarms refers to the explicit effort to re-create group behaviors seen in natural groups of animals with limited individual capabilities, such as ants, bees, birds, and fish. While other research areas within robotics use bio-inspiration for design of actuators and physical robots [15, 16], bio-inspired swarm research focuses on group behavior algorithms and communications to utilize the "swarm intelligence." [17] Perhaps the most common research area within bio-inspired swarms is the attempt to recreate insect swarm behaviors, including such applications as foraging (mimicking ant foraging) [18] and source seeking. While insects are the most common inspiration, researchers have also mimicked fish as well as packs of wolves [19].

1.1.3.2 Communication, Control, and Motion Coordination Protocols

The issues of communication, control, and motion coordination protocols fall within the area of "swarm protocols," or how the swarm behaviors will actually be achieved.

Communication is a necessity given the inter-connectedness and collaboration of robotic swarms, and can be done explicitly or implicitly. Explicit communication involves direct transmittal of robot data (such as position and sensor reading data), while implicit communication is most commonly achieved through stigmergy. Stigmergy involves indirect communication through actuation on the environment. A common implementation of this in swarms is through virtual "pheromones" [20]. For example, it may be desired to share data about an individual robot's sensor values for a certain pollutant. Explicit communication would involve directly sending this information to other robots (or to a centralized location that then transmits it to other robots). In contrast, stigmergy would involve the robots releasing a "pheromone" at a level proportional to their sensor value reading. In this implicit communication, data can be shared more easily across a wider array of robots, but there are other concerns that arise with lag in sharing of data, imprecise data sharing, and additional sensors needed to sense the "pheromone." Despite these concerns, the use of "pheromones" can help reduce computational load by only sharing data with local neighbors.

At the control level, a decision is made about whether control will be done in a centralized or distributed manner. While individual commanded velocities must be distributed to each individual swarm robot, the process of the control is a key decision. Centralized control involves one controller/computer that calculates all robot velocities and then distributes them to the proper agents. In contrast, distributed control involves each robot making its own calculations through a set of control or behavior laws and acting off of those. A blend of distributed and centralized control is also possible for certain applications [30].

1.1.3.3 Swarm Intelligence and Swarm Learning

As was stated in section 1.1.3, all swarm robotic systems utilize swarm intelligence at some level, as they all leverage the knowledge or abilities of simple agents to create a complex group behavior. However, some areas of swarm research are much more focused on this “intelligence” aspect of the swarm and attempt to utilize learning algorithms to maximize the “intelligence” of the swarm. This area is especially relevant for groups focused on implementing swarms in unknown or rapidly changing applications [22]. Swarm learning may take a variety of forms, such as decision trees [24] or neural networks [25]. It should be noted that this area of research is closely linked with AI, Machine Learning, and other research fields, and is not exclusive to swarm robotics. However, the typical architecture of a robotic swarm lends itself to research in this area, and oftentimes may serve as a testbed for these research areas.

1.1.3.4 Applications-Based Swarm Development

Whereas sections 1.1.3.1-1.1.3.2 have discussed particular research areas within swarm robotics, a particular approach to swarm robotics research that places the desired application at the forefront is discussed in this section. It should be noted that this can be executed in conjunction with the other research areas previously stated, but that the main goal of the research is toward a certain application, rather than the development of a capability. The most common applications found in research currently are task allocation, object transportation and manipulation, and source finding. Task allocation is necessary when the swarm must accomplish multiple goals simultaneously. In these instances, it is necessary for the swarm to determine which robots should complete each task [26]. Many different strategies have been proposed for how to most successfully allocate tasks. These include market-based [27], auction-based [28], distributed planning [29], and fault-tolerant techniques [30]. Object transportation and manipulation can encompass “grasping” a physical object [31], pushing the object without grasping [32], or “guarding” a target [33]. Finally, source finding has been explored by multiple researchers. Due to its direct connection to adaptive navigation techniques and this thesis, current research in this area is discussed in section 1.2.2.

1.2 Adaptive Navigation

The focus of this section of the paper is to discuss a particular application - Adaptive Navigation - for which swarm control techniques can be utilized.

Adaptive Navigation (AN) is the process of determining a goal location during transit rather than specifying a goal *a priori*. This stands in contrast to traditional navigation, which often involves goal coordinates, waypoints, or a time-dependent trajectory. Oftentimes, AN implies that the quality of the goal location may be known, rather than the specific coordinate(s) or location(s). A disaster-relief example of AN might involve the mapping of the spread of a pollutant, in order to determine which areas need to be evacuated. In this example, the goal of the AN would be to use sensor readings of pollutant concentrations in order to effectively map and determine pollutant spread. Subsequent sections will discuss approaches to AN, with a survey of techniques in general, and a section that will focus on swarm approaches to implementing AN.

Frequently, adaptive navigation is used in scalar fields, that is, fields that have a discrete scalar value for each position in the field. Adaptive navigation is particularly useful in finding features of interest within scalar fields, such as maxima or minima, contours of a specified value, ridges and trenches, or saddle points, as indicated in Figure 13.

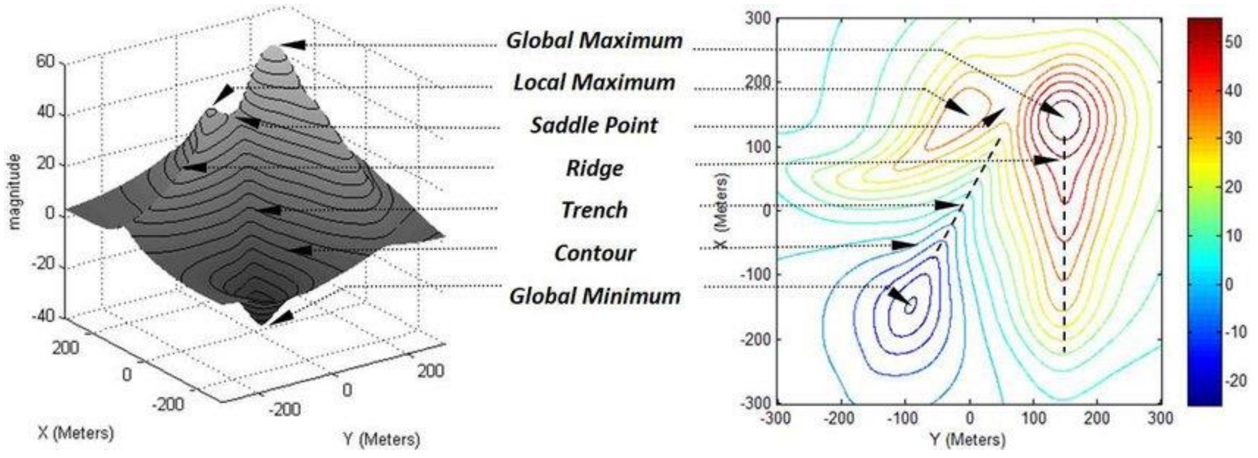


Figure 13. Key Features in a Scalar Field [34]

1.2.1 Approaches to AN

Adaptive Navigation is heavily linked with the field of mobile robotics, including applications in single mobile robot systems as well as MMS. Single mobile robot adaptive navigation systems may utilize AN for similar reasons as MMS, but are often more limited in their abilities or approaches. Single mobile robot systems may utilize differential sensing, but are limited by the fact that the sensors are limited in their distribution by the robot's size. Despite this, efforts have been made to utilize single quadcopters with differential sensing for source-seeking applications [36]. Alternatively, single mobile robots may implement adaptive navigation techniques that are much more dependent on memory and storage of previous conditions, such as Simultaneous Location and Mapping (SLAM) algorithms [37]. Such algorithms, while powerful, require robust systems with much higher computing power and storage capacity.

One of the main advantages of utilizing MMS for adaptive navigation, especially when implementing differential sensing techniques, is that the sensors can be spread over a much wider area. This wider area can be utilized to more accurately estimate relevant gradient information, which is the basis for most scalar field adaptive navigation techniques. Recently, the RSL has successfully implemented such scalar field AN techniques utilizing the cluster-space MMS technique discussed in section 1.1.2.1. By utilizing differential sensing across robots in the cluster, feature information could be estimated in order to successfully accomplish minima and maxima finding, contour following, ridge and trench following, and saddle-point "station keeping." [34] Another advantage of utilizing MMS for AN is the fact that decisions can be based solely off real-time sensor readings, which makes the system more agile than a single robot basing decisions on both current and past states.

1.2.2 Survey of Adaptive Navigation utilizing Swarm Control Techniques

As was mentioned in section 1.2.1, there are many advantages to utilizing MMS for AN. In the area of swarm control techniques, a detailed survey of the research did not indicate that researchers have explicitly been working towards creating "adaptive navigation primitives" similar to those presented in [34]. There is, however, extensive research in utilizing robotic swarms for source finding and tracking. In the context of [34], this would be the problem of minima or maxima finding, depending on the type of the source.

As mentioned in section 1.1.3.1, robotic swarms are often bio-inspired, and this trend is especially prevalent in source-finding applications. For example, efforts have been made to mimic moth behaviors in source finding applications [38]. In [38], a behavioral architecture was used that involved tracking and reacquiring a "plume," which implies a well-defined scalar field ridge. While this architecture is similar to that used in this thesis' behavioral architecture, the individual behaviors are more complicated and involve the storage of previous states. Similar plume-tracking behaviors have also been expanded [40] and have been proven effective for time-variant sources as well. In addition to moth-inspired algorithms, some researchers have based search algorithms off schools of fish [41, 42]. This strategy involves a dual-input velocity determination, wherein velocity is based off formation keeping as well as individual measurements of the scalar field.

In addition to these bio-inspired approaches, research has been done in swarm-based source seeking that attempts to work past a common assumption within other research areas. These are summarized in Table 2.

Table 2. Summary of Assumption-based Swarm Source-Finding Research

Common Assumption	**Description of Work addressing Assumption**	**Author(s)**
Swarm robots have unlimited battery life in the context of their mission	Employ a "layered" behavior architecture that includes behaviors for returning home to recharge. Tested on quadcopters with flight endurance constraints.	Gainer, J. *et al.* [44]
Source will be constant (or if time-variant, always measurable)	Simulated tracking of an intermittent RF Source that may not always be measurable or detectable. Employed an estimator and path-planning optimizer for achieving fastest convergence to source.	Koohifar, F. *et al.* [45]
Swarm Robots have knowledge of their global position	Structured behaviors off of relative bearing angle to other robots in the swarm, without using distance or global position of individual robots	Fabbiano and Garin [46]
Swarm robots are essentially acting as particles, as their size is small relative to the source	Based simulations off of larger robots with non-holonomic kinematic constraints to verify that cooperation algorithms hold for non-particle robots.	Xue Songdong [47]

1.2.3 Working Definition of Swarm Robotics

Given the wide variety of uses and definitions of Swarm Robotics, it is necessary to present a working definition of "swarm robotics." In this thesis, a swarm is:

1. *Homogeneous:* All robots within the swarm should have similar actuation and sensing capabilities to ensure interchangeability of individual robots. The robots can have an "identity," whether this is through numbering or naming of robots.

2. *Complex Group Behavior:* Swarms should achieve behaviors that individual robots are not capable of by themselves.
3. *Simple, Superimposable, Universal Behaviors:* Robot control is achieved through simple behaviors that can be superimposed or “stacked.” All robots within the swarm should be programmed to have the same set of behaviors.
4. *Local, Real-time information:* Robots should only utilize information between local neighbors (and not global information), and the robots should only use real-time information in behavioral calculation, rather than storing past information in a buffer for more complex calculations.
5. *Constant-Speed Robots:* Robots within the swarm should be set to move at a constant speed, with behavioral commands resulting in a bearing command, thus controlling the direction rather than the magnitude of the robot’s velocity.

1.3 Major Contributions

As discussed in previous sections, swarm control techniques have been utilized for a variety of “source-seeking” applications, though no previous research was found that made the explicit jump from source-seeking to adaptive navigation applications. The goal of this thesis is to make this jump, and achieve adaptive navigation primitives with swarm control techniques similar to those achieved using the cluster-space formulation. The three main contributions of this thesis are as follows:

1. *Development of a Simulation Environment in Matlab/Simulink* - An easy-to-use simulation environment was developed in Matlab/Simulink that allows for rapid testing and vetting of swarm behaviors. The simulator was developed so that it can easily interface with the testbed previously developed by RSL [35].
2. *Demonstration of top-down specification of swarm behaviors* - As was mentioned in section 1.1.3, swarm robotics often involves the use of “emergent” behaviors that result from a combination of simple “primitive” behaviors. This paper demonstrates a movement towards more distinctly specifying behaviors from a top-down perspective. This includes specification of design guidelines for achieving certain desired swarm-level behaviors.
3. *Demonstration of Adaptive Navigation Primitives* - This thesis successfully demonstrates adaptive navigation primitives such as minima and maxima finding and contour finding and following. These AN primitives are vetted in simulation and have been validated on the RSL’s testbed.

The development of adaptive navigation behaviors using swarm robotic techniques provides an alternative method to the cluster-space technique that may be more easily applicable in cases of poor communication or high probability of individual mobile robot failures. The adaptive navigation techniques that are developed have the potential to more rapidly complete mapping applications and could also be used in disaster-relief scenarios.

1.3.1 Research Collaboration

This research began as class project for MECH 296A: Mobile Multi-robot Systems at Santa Clara University, where a simulation environment for swarm behaviors was initially developed in Matlab/Simulink, with collaboration between the author and classmates Jimmy Nguyen and Jason Fu. This simulator was built to observe the behavior of 10 swarm robots for swarm behaviors including Attract, Disperse, and Go-To Coordinates. The initial work associated with this thesis involved an expansion of the MECH 296A simulator and was done in collaboration with Santa Clara University graduate students Maximilian Reese and Shae Hart. Much of the initial expansion of the simulator was done by Max Reese, including extensive Matlab/Simulink programming to allow for a variable number of robots and easy manipulation of behavior laws across all robots. The conversion of the simulation environment to interface with the existing testbed environment was done by Shae Hart. The author: (1) developed a GUI to allow rapid variation and testing of simulation and experimentation parameters, (2) developed post-processing plots and metrics to evaluate swarm performance, and (3) conducted the simulation and hardware experiments described in Chapter 4. Adaptive Navigation behaviors such as extrema finding and contour following were developed and refined in collaboration between all 3 students.

2. Simulator Development

One of the main foci of this research has been to develop a Swarm Adaptive Navigation Simulator (SANS) that easily allows for the composition, superposition, and testing of swarm-level behaviors. Initial testing and behavior development is performed in SANS' simulation mode. Once behaviors are fully vetted, they can be tested on hardware using SANS' testbed mode. SANS is segmented into different tiers, which are discussed in the following section from the lowest level to the highest level.

The simulator is structured in a way that it can be used to rapidly run and test different behaviors for different numbers of robots, lengths of simulations, types of behaviors, robot speeds, and scalar fields. Additionally, as is discussed in section 2.6, new behaviors can be added to the simulator and rapidly tested and verified.

A high-level view of the simulator control architecture is shown in Figure 14.

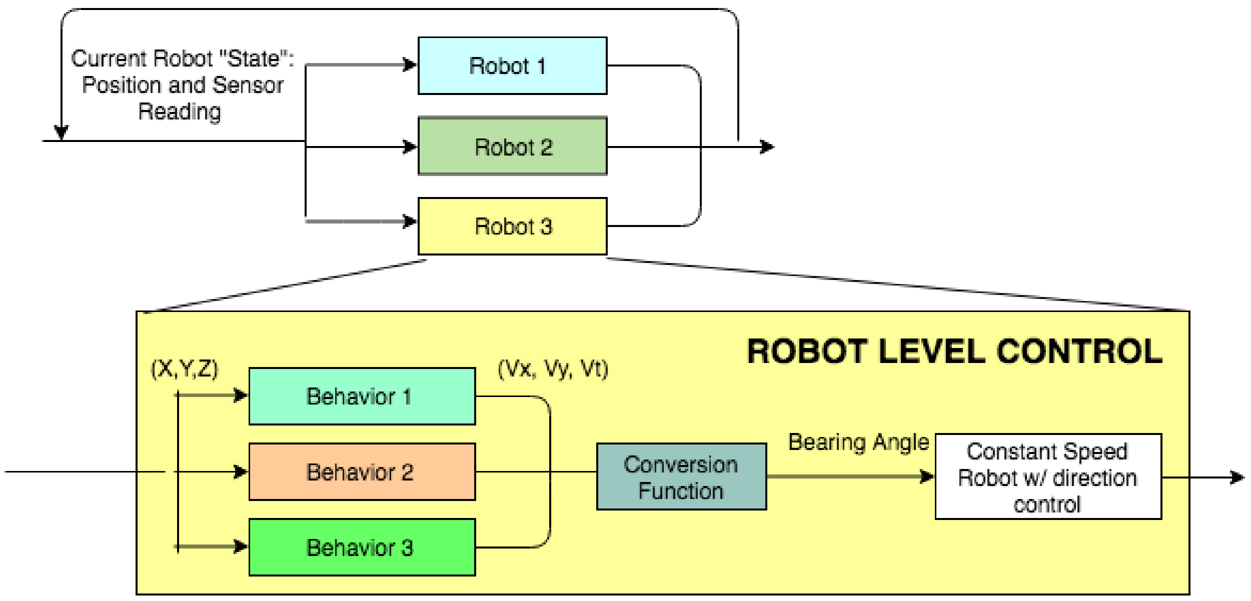


Figure 14. High-level schematic of the SCUSANS control architecture.

At the highest level, each robot shares its current state, which can include position, orientation, and sensor value information. This current state is then fed as inputs to each individual robot at a behavior control level, where each behavior calculates a velocity vector based on the current swarm poses. These velocities are then aggregated and converted into a bearing angle that is given as a command to an omnibot with on-board velocity control. The conversion function is the summation and weighting of the input velocity vectors, as well as the conversion of that information into a bearing command angle for the constant speed robot. The behaviors, conversion function, and constant-speed robots are discussed in the following sections. In the simulation environment, there is a necessary screening of incoming data to simulate the communication range of the individual robots within the swarm.

2.1 Robot-Level: Omnibot

The base level of SANS is the Omnibot, based off the Decabots used in the RSL's mobile robotics testbed, shown in Figure 15. The Omnibots are equipped with three Omni-wheels that allow the robot to move instantaneously in any direction, while also allowing for rotation about the robot's *z* axis. Each Omnibot is also equipped with an on-board sensor that can read the gray-scale value of scalar field printouts that are laid out on the testbed. We also assume that each swarm robot has knowledge of its position, *via* either GPS, or in the case of the testbed, an Optitrack^{TM} system, and that this data can be transmitted to other robots in the swarm.

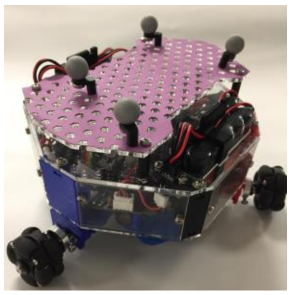


Figure 15. Omnibot used in the RSL's Decabot Testbed System [35]

For the SANS simulation mode, a model of these omnibots is used to calculate the robot-level position response to an input bearing command. For each robot, once the new position has been determined it is fed through a function that determines the robot's "scalar value," which is the value at that (*x*,*y*) location in the loaded scalar field. This simulates the Decabot's reading of a grayscale value from its on-board sensors.

In SANS, the holonomic omnibots are running at a constant speed. Robot control is therefore specified as a bearing angle, which is then converted to individual wheel velocities using the inverse Jacobian for the robot. The robot-level velocity control architecture is shown in Figure 16. The commanded global bearing for robot i, Φ_i , is converted into a velocity in the robot frame, $\overrightarrow{V_{i_{cmd}}}$. For the omnibot in Figure 15, this velocity vector has translational components in the robot's and *x* and *y* axes, and a rotational component about the robot's *z* axis. Using the robot's inverse Jacobian, the robot velocity is converted into wheel velocities, $\overrightarrow{\omega_{\iota_{cmd}}}$. Each wheel is modeled as a first-order transfer function, and is represented by the blocks Wheel *N*. The output from each Wheel *N* is the dynamic response to the commanded wheel velocity. The combined dynamic response is multiplied by the Jacobian to convert back to robot-level velocities and is then transformed back to the velocity in the global frame. Integrating the global velocity over time yields the current pose of the robot, (x_i, y_i, θ_i).

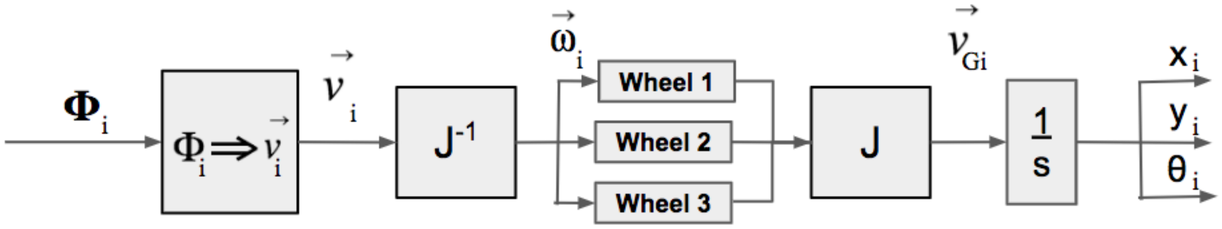


Figure 16. Robot-Level Velocity Control for the Omnibot

2.2 Command-Level: Behavior Commands

Behavior command occurs at the level above robot speed control. At this level, the current robot position and sensor data from local neighbors yields a bearing angle command that is then sent as an input to the constant-speed robot discussed above. In this level, multiple behaviors are specified and can be switched on or off, depending on what behavior or combination of behaviors is desired. Each behavior is a function of the current "robot parameters," which are the current position and sensor value readings of each robot in the swarm. Although global swarm data is available in the simulation environment, only local information may be used in order for the swarm to conform to our working definition in section 1.2.3. As such, data from robots that are not within communication range is filtered out in SANS. The individual behaviors are discussed in depth in Chapter 3.

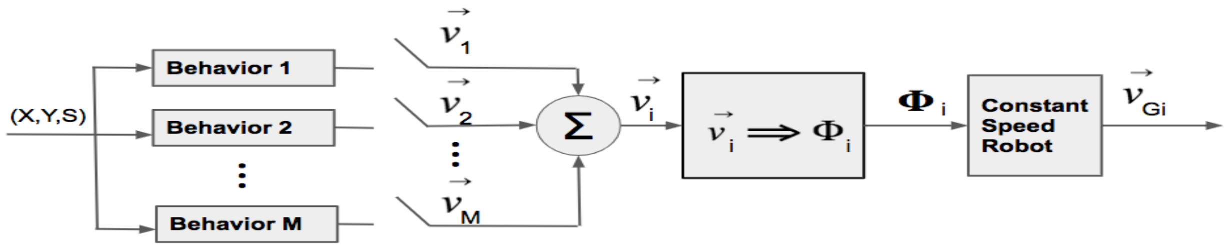


Figure 17. Behavior Level control

The control architecture for the behavior level is shown in Figure 17. The global swarm information, including robot position, orientation, and sensor value data (X, Y, S) is the input to each behavior. Though not shown in Figure 17, this data is filtered to ensure that only local information is used. Each behavior *m* for robot *i* yields a command velocity vector $\overrightarrow{V_{mi}}$ in the global frame. These command velocities are then converted to a bearing command in the global frame, ϕ_i. The conversion function also allows for weighting of behaviors *via* a gain associated with each commanded velocity vector. Generally, behaviors were given equal weighting. However, for certain composite behaviors, such as flocking (Section 3.2.1), utilizing different weighting gains can yield different swarm-level behaviors.

For the purposes of this swarm, all robots are assumed to be identical with the exception of individual robot numbers. This means that each robot has the same behaviors programmed in at the behavior command level, and all robots will have the same behaviors turned on or off. In Figure 17, the "Constant Speed Robot" block is a subsystem that consists of the functions depicted in Figure 16.

2.3 Group Level: Swarm Architecture

Above the behavior control level is the swarm architecture. At the swarm level, *N* identical, numbered robots can be specified, as shown in the block diagram below:

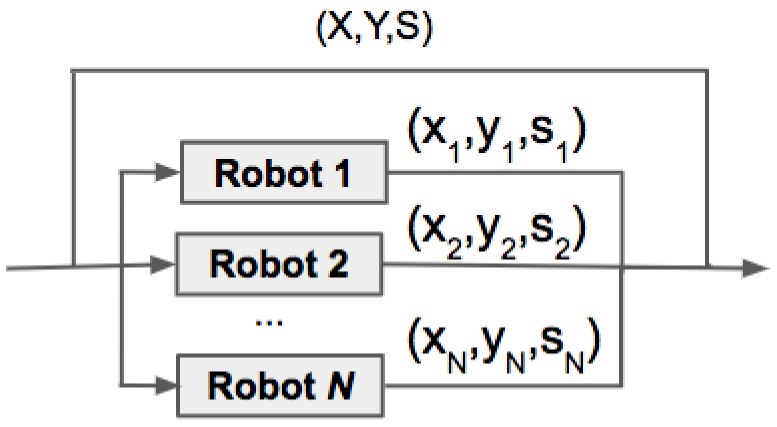
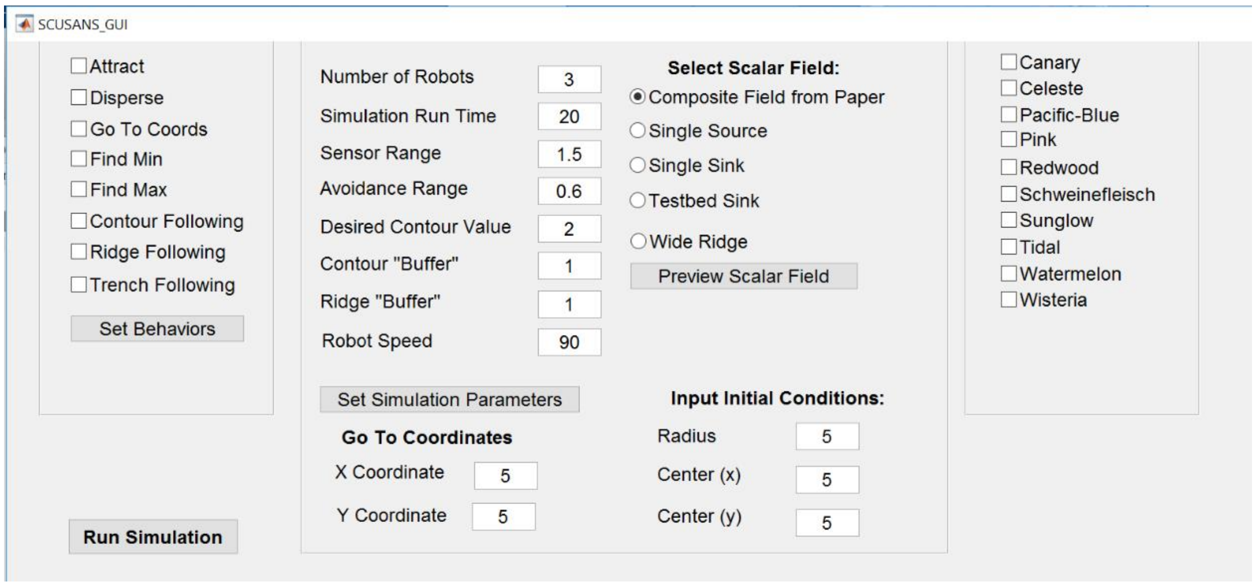


Figure 18. Swarm-Level Architecture

In the block diagram above, each block for robot *i* includes both the omnibot and the behavior commands discussed in the prior two sections. Thus, at the swarm level, the input is the current "swarm state," which includes information about all robot position and sensor data, and the output is the swarm pose at the next time step. This updated swarm pose is then fed back as the input for the robot behaviors of the next iteration. In the context of the simulator, it is important to note that while each robot receives data about every other robot, not all of this data is marked as "usable" by the robot. As will be discussed in the Behavior Algorithms section, we make the assumption that each robot has a finite "Communication Range" in which it would be able to communicate with other robots in the swarm. If robots fall outside of this communication range, although data is received, it is not used in behavior algorithms.

2.4 User Level: User Interface

The top level of the swarm simulator is the user interface, which allows for easy specification of robot behaviors, swarm size, and other factors. The GUI is shown in Figure 19 and is split into five main categories: behavior selection, simulation parameter selection, scalar field selection, mode selection, and robot selection. The robot selection is only necessary for the testbed, and not the simulator.


SCUSANS_GUI

☐ Attract
☐ Disperse
☐ Go To Coords
☐ Find Min
☐ Find Max
☐ Contour Following
☐ Ridge Following
☐ Trench Following

Number of Robots	3
Simulation Run Time	20
Sensor Range	1.5
Avoidance Range	0.6
Desired Contour Value	2
Contour "Buffer"	1
Ridge "Buffer"	1
Robot Speed	90

Go To Coordinates

X Coordinate	5
Y Coordinate	5

Select Scalar Field:

[✓ Composite Field from Paper] Single Source Single Sink Testbed Sink Wide Ridge

Input Initial Conditions:

Radius	5
Center (x)	5
Center (y)	5

☐ Canary
☐ Celeste
☐ Pacific-Blue
☐ Pink
☐ Redwood
☐ Schweinefleisch
☐ Sunglow
☐ Tidal
☐ Watermelon
☐ Wisteria

Figure 19. User Interface for specification of swarm behaviors and parameters

The first category, labeled Select Behaviors, allows the user to specify which behaviors will be turned on for the swarm as whole. These relate to the switches shown in Figure 17 that detail switching of behaviors at the behavior command level.

The second main category is simulation parameters, which has the following components:

- *Number of Robots* - user can specify up to 10 robots to be part of the swarm
- *Simulation Run Time* - allows user to specify how long the simulation will run
- *Sensor Range* - used in determining which robots are "in range" of each other.
- *Avoidance Range* - used in the obstacle avoidance algorithm.
- *Desired Contour* and *Contour "Buffer" Zone* - used in the Find/Follow contour behavior.
- *Robot Speed* - Sets the speed of the constant speed robot at the Omnibot level.

The sensor range, avoidance range, desired contour, and contour "buffer" zone are all discussed in more depth in Chapter 3.

The third section of the GUI allows the user to specify which scalar field to use for adaptive navigation purposes. The user's selection is loaded into the scalar field reading function at the Omnibot level that determines each robot's "scalar value" when in simulation mode. Finally, the GUI allows the user to specify initial conditions in three ways. The first is by using "Default" conditions. Default conditions allow the user to specify the center and radius of a circle, and then spaces the individual robots equally along the circumference of that specified circle. Users can also set robot initial conditions "manually," where the user can specify initial conditions *via* mouse click in a coordinate plane. Finally, the user can select "random" and have the robot initial conditions randomly dispersed throughout the scalar field domain.

The final section of the GUI, Select Mode, allows users to easily transition from simulation to experimental validation on hardware. To use this mode, the user specifies in the Select Robots section which robots will be used for their experiment. Then the SANS backend handles importing and connecting all of the Simulink blocks necessary for communication with the OptiTrack system and the individual robots. During this transition, the SANS control architecture remains unchanged; the only two differences are the use of the OptiTrack system to determine robot pose instead of calculating poses from the Decabot models and the use of the Decabots' onboard sensors to measure scalar values instead of using the provided scalar field function.

2.5 Post-Processing and Visualization of Results

In order to easily visualize the results of the simulation, there are 3 post processing outputs produced by the simulator. The first is a video that displays the robots' positions over a color-coded scalar field, as shown in Figure 20. In this video, each robot is represented as a colored point. The blue line running through the image represents the specified "desired contour value."

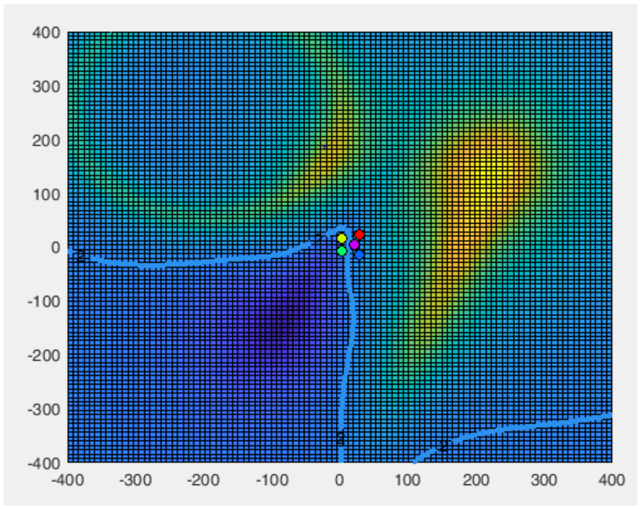


Figure 20. Freeze-frame of video of swarm behavior

In addition to the video, a time history of the robot positions is superimposed on a contour map of the scalar field. In this view, each thick line represents a robot trajectory, while the thin lines represent contours of specific values. Figure 21 shows this functionality for a "Find Max" behavior (left) and for a "Follow Contour" behavior (right). The 3D view of the time history allows for easy visualization of performance of adaptive navigation behaviors.

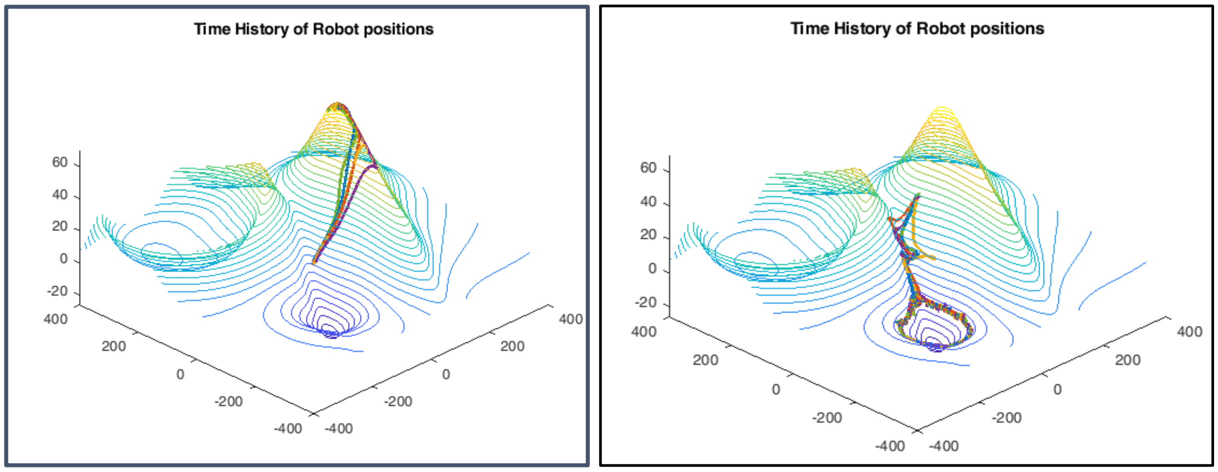


Figure 21. Post-processing plots of time history of robot positions

Finally, for adaptive navigation behaviors (such as find min, find max, or follow contour) it can be beneficial to view the time history of the sensor values, as these readings are being used in the behavior control laws. Two examples are displayed below for "Find Max" and "Follow Contour." These plots include not just the sensor values, but also information that can help show how effectively the swarm has achieved the desired behavior. For example, the Find Max Sensor

Value Time History indicates the known global max value, while the Follow Contour displays the desired contour value. This can be seen in Figure 22, with the left image showing a Sensor Value time history for a Find Max behavior, and the right image showing the time history for a Follow Contour behavior.

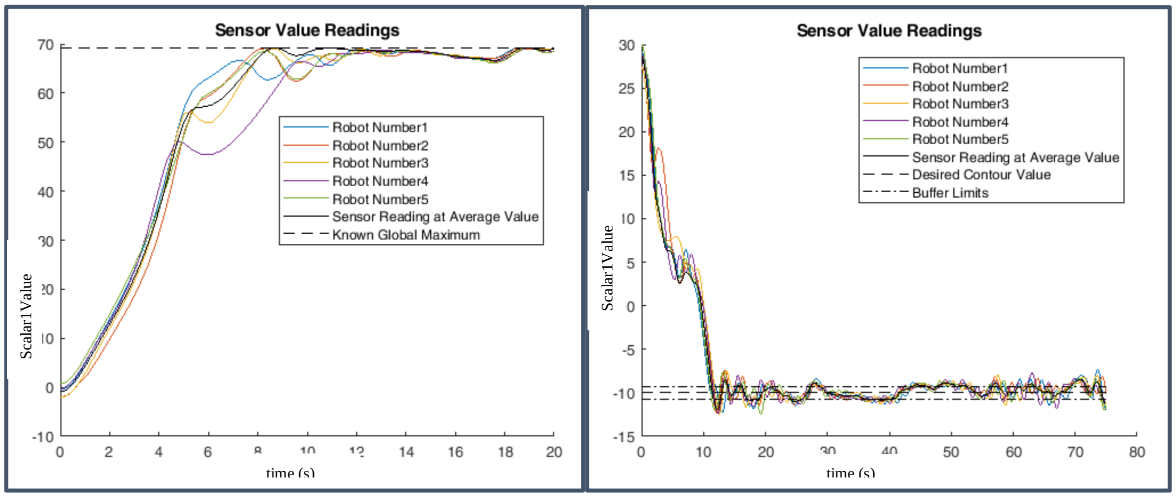


Figure 22. Post processing plots of time history of sensor value readings

These post-processing plots help to give an indication of how well the system is achieving the desired swarm behavior. For example, the left plot of Figure 22 shows that the swarm successfully moves toward locations with higher scalar field values and settles around the known global maximum value. Around t=14 seconds, the swarm drifts from the global maximum but successfully re-locates the max by t=18 seconds.

2.6 Adding Additional Behaviors

The main goal of developing the simulator was to enable rapid testing of individual behaviors and combinations of behaviors. This was accomplished by creating an easily scalable system that allows for user-inputs for system variables (such as number of robots, robot speeds, and simulation times). Additionally, the user interface makes it easy to visualize which behaviors are being utilized. It should also be noted that, because the architecture is structured to allow individual behaviors to be stacked, new individual behaviors can be added relatively quickly. In order to add a new primitive or composite, it must be added in the following locations:

- *Simulink Behavior Block* and *Behavior Switch* - The new behavior block must be added as a "Matlab Function" block in the Simulink file "Swarm_Robot_Base." In order to make this behavior switchable, a switch must be added downstream, with a zero velocity vector set to the "switch off" position.

- *Conversion Block* - the new behavior must be added to the "SwarmSimSum" function, and a gain selected for the conversion function.
- *User Interface* - In order to switch the behavior on or off in the user interface, the interface must be updated (using GUIDE) to have a checkbox associated with the behavior switch in "Swarm_Robot_Base."
- (Optional) *plotRobotHistory* - As discussed in 2.5, the simulator will produce a post-processing video of the robots, displayed over the current scalar field. A separate post-processing plot, if desired, (such as the time history of the robots, or the time histories of the sensor values) can be added in the plotRobotHistory sub-function of SwarmRobotTestSim. It should be noted that in order to do this, a new case must be added for the desired behavior, and the setup for this case can be done in the SCUSANS_GUI function.

2.7 Testbed Environment

After behaviors are developed and vetted in the simulation environment, they are tested and validated on hardware. The hardware used for these swarm robots is the Decabot indoor testbed system in Santa Clara University's Robotics Systems Laboratory (RSL) presented in [35]. The Decabot system can utilize up to 10 of the omnibots shown in Figure 15. These robots are equipped with grayscale sensors that allow for the reading of the magnitude of a grayscale plot. Each robot sends its grayscale sensor value reading to the control computer *via* wireless connection. A sample grayscale plot with 3 omnibots is shown in Figure 23.

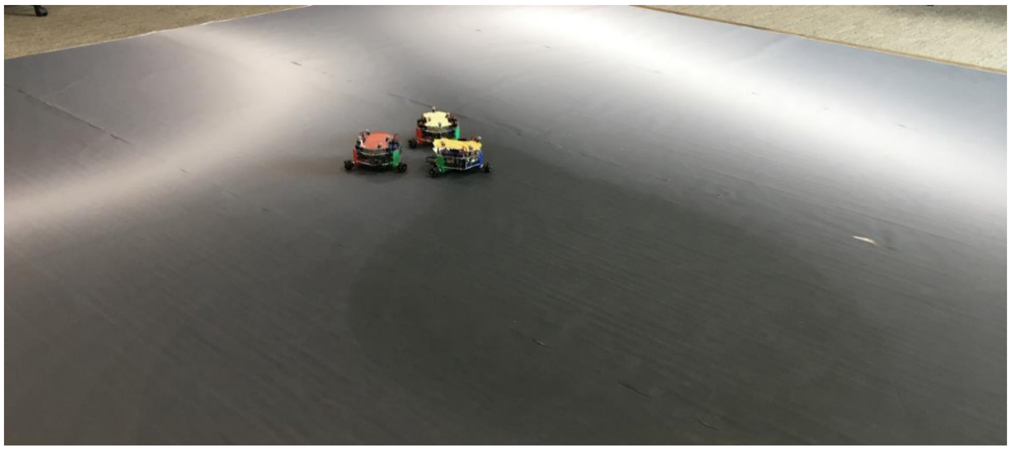


Figure 23. Sample Grayscale Field with 3 Decabots

Robot position and orientation is determined using an OptiTrack system. Each robot has uniquely spaced markers on it that are sensed by the OptiTrack system and are used to define a rigid body. The position of each marker is sensed by the OptiTrack cameras, and the OptiTrack system then converts this raw marker position data to the position and orientation of each of the uniquely defined rigid bodies. The position and sensor value readings are then sent to the Simulink environment, where behavioral control laws are implemented. In this controlled testbed environment, the global swarm data is still available, so filtering of robot data to only utilize local neighbors' information is still necessary. The architecture for the testbed is shown in Figure 24, adapted from [35]. A streaming server, DataTurbine, is used as an intermediary to send and receive data. Via this channel, Matlab/Simulink receives sensor data from the robots and position data from the OptiTrack system. The robot velocity commands are then calculated in Matlab and distributed to the individual robots. Despite using a centralized computer, the swarm control techniques are still a distributed technique due to the control architecture, presented in 2.2-2.3.

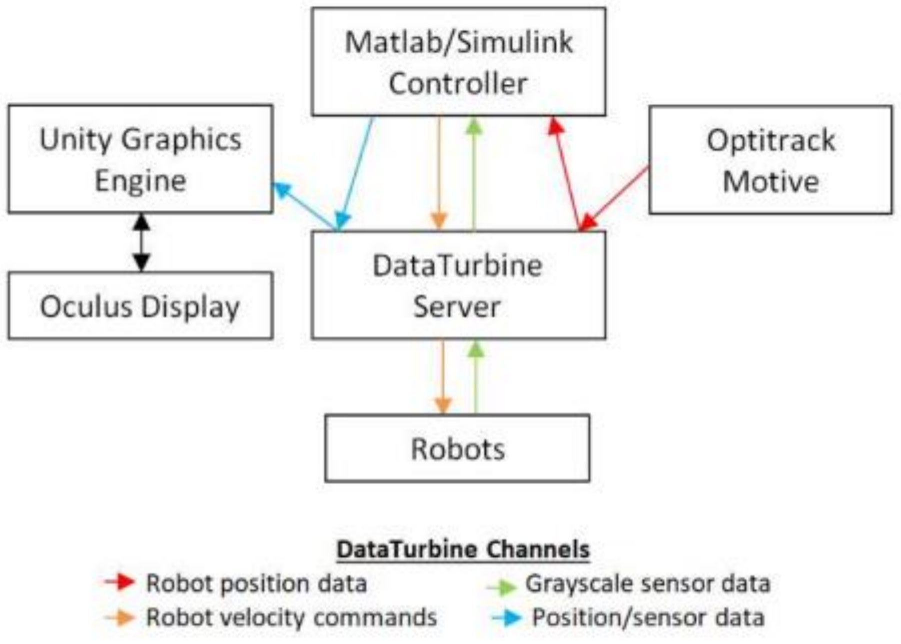


Figure 24. Decabot Indoor Testbed Architecture [35]

3. Behavior Control Laws

As discussed in section 2.2, robot control is achieved through behavior control, where current information about individual robots within the swarm is converted into a velocity command for the omnibots. In general, these behaviors can be split into two main categories of "Primitives" and "Composites." Primitives are base-level calculations or behaviors, while composites are "stacks" of multiple primitives. Behaviors can stack in multiple ways, such as addition, multiplication (or scaling one primitive by another), or some combination of the two. The behavior descriptions below provide a schematic for the intended result of the behavior, a mathematical definition of how the behavior can be implemented, verification of the behavior in simulation, and validation of the behavior in the testbed.

While these behaviors have been split into primitives and composites, it should be noted that some composite behaviors can also be labeled as "adaptive navigation primitives" to be consistent with [34]. For example, the Find Min/Find Max behavior is an adaptive navigation primitive, but is a swarm composite behavior consisting of the swarm primitives of attract and sensor value comparison.

3.1 Supporting Calculations

Swarm behaviors, whether primitives or composites, yield an output velocity vector that can be superimposed with other behaviors. In the process of computing these behaviors, there are two supporting calculations that do not yield output velocities in and of themselves. The first, out-of-range determination, is a condition imposed in simulation to reflect hardware communication conditions. The second, sensor value comparison, can be utilized as a gain in later composite behaviors, but by itself does not result in a commanded velocity.

3.1.1 Out of Range Determination

The "out of range determination" is an artificial constraint imposed in simulation to mimic the communication range constraint of individual robots in the swarm. As was mentioned in section 1.2.3, one of the defining characteristics of swarms is the fact that they are limited to local communication only. However, in simulation, all robot data is available to each other. Therefore, it is necessary to implement a constraint on inter-robot communication so that behaviors are only a function of robots that are within communication range of each other. The range determination is accomplished by calculating the distances, d_{ij}, from each robot *i* to all other robots *j*, by using the (x, y) position of each robot *i* (x_i, y_i) and robot *j* (x_j, y_j) as shown in Equation 1.

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \qquad \text{Equation 1}$$

The distance from each robot *i* to *j* is then compared to the robots' communication range, *Rs*. For all $d_{ij} > Rs$, the robot *j* is considered to be "out of range" of the robot *i*, and vice-versa. These out of range robots are then "discarded" and not used in the determination of the rest of robot *i*'s behavior controls.

3.1.2 Sensor Value Comparison

Sensor value comparison is another supporting calculation, as it does not explicitly affect the swarm behavior unless it is paired with another behavior. The sensor value of robot *i* is compared to all other robots *j:*

$$\Delta SV_{ij} = SV_i - SV_j \qquad \text{Equation 2}$$

For future behaviors, it is important to note that for this convention, positive values of ΔSV_{ij} indicate that robot i has the higher sensor value reading, while a negative value indicates that robot *j* has the higher reading.

3.2 Swarm Primitive Behaviors

The following behaviors are "primitive" behaviors that can be used by themselves, or stacked with other primitives in order to achieve more complex swarm behaviors.

3.2.1 Obstacle Avoidance

An obstacle avoidance algorithm is necessary in any mobile robotics application, and especially in mobile multi-robot systems, so that the individual robots will not collide with each other. Obstacle avoidance is particularly important in adaptive navigation scenarios where the terrain being explored may be unknown and external obstacles may be present.

The obstacle avoidance algorithm constantly monitors the relative positions of the other robots, but does not put out an avoidance velocity component unless the avoidance range has been triggered. In contrast to the other behaviors in the simulator, the obstacle avoidance behavior has a variable velocity magnitude. In this way, the obstacle avoidance, when triggered, has the ability to “override” and dominate other behaviors to insure that inter-robot and robot-object collisions are avoided. The magnitude of the avoidance velocity is:

$$M_{Oij} = \frac{{R_a}^4}{{d_{ij}}^4}, d_{ij} < R_a \qquad \text{Equation 3}$$

$$M_{Oij} = 0, d_{ij} > R_a$$

where M_{Oij} is the magnitude of the obstacle avoidance velocity for robot i in the direction away from robot *j*. R_a is the obstacle avoidance range. Chapter 4 discusses criteria for selecting an obstacle avoidance range, but the range must be at least ~2.5x the radius of the physical robots to ensure collision avoidance. The range must also be significantly smaller than the communication range so that robots do not repel each other out of communication range. The distance from robot *i* to robot *j*, d_{ij}, must also be calculated as described in Equation 1. The obstacle avoidance behavior is not individually validated, but is validated in combination with the other primitive behaviors described in sections 3.1.4-3.1.5.

The magnitude of the obstacle avoidance velocity is determined using methods similar to those presented in [49]. This allows for strong repulsion in the vicinity of obstacles and other robots, while yielding negligible obstacle influence when robots are farther away from obstacles or other robots. The ratio of the avoidance range to the distance between robots is raised to the 4th power because this was found to yield effective collision avoidance without negatively affecting other swarm behaviors.

3.2.2 Go-To Coordinates

A Go-To Coordinates behavior was implemented for the swarm. In this behavior the user specifies the goal pose’s *x* and *y* coordinates and each robot attempts to travel to it. It should be noted that this is not necessarily a “swarm” behavior, as each robot can perform the behavior individually, without needing robot-to-robot communication. This can be seen by the fact that each robot’s velocity vector is a function of its own pose and the desired pose only:

$$V_{gxi} = K_g(x_{des} - x_i) \qquad \text{Equation 4}$$

$$V_{gyi} = K_g(y_{des} - y_i)$$

where V_{gxi} is the x-component and V_{gyi} is the y-component of the velocity output vector for the go-to behavior that is fed into the conversion function at the behavior level (see section 2.2). K_g

is a scaling gain, nominally set to 1. Though the Go-To Coordinates behavior doesn't utilize information from other robots, it is an important behavior to implement for multiple reasons. First, this simple behavior can be easily verified when moving from the simulator to the testbed. Second, it can be coupled with other primitives to achieve more complex composites. A general schematic for the go-to behavior is shown in Figure 25. The left image shows the robots at time t=0, while the right image shows the expected behavior once they have settled, with the robots reaching an equilibrium position that balances the go-to velocity vectors with the obstacle avoidance the robots feel against each other (see section 3.2.1 for details on obstacle avoidance).

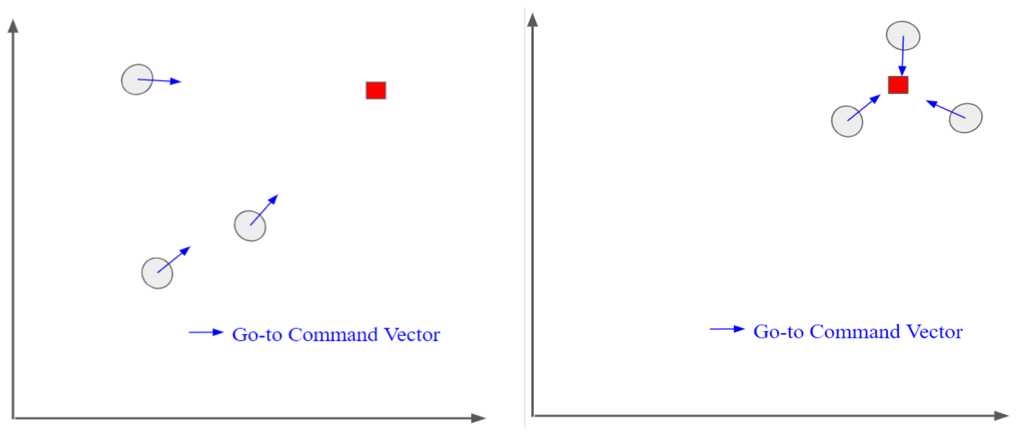


Figure 25: Schematic of go-to behavior for *N=3* robots

The go-to behavior is first verified in simulation and then validated using the testbed. To validate the effectiveness of the go-to behavior, the centroid of the swarm is calculated by taking the average position of the robots, and comparing this to desired goal location, as can be seen in Figure 26. The left figure indicates the go-to functionality in the simulation environment, and the figure on the right indicates that the behavior is repeated effectively in the experimental testbed. In the figures below, x's indicate the initial robot positions, and 0's indicate final robot positions.

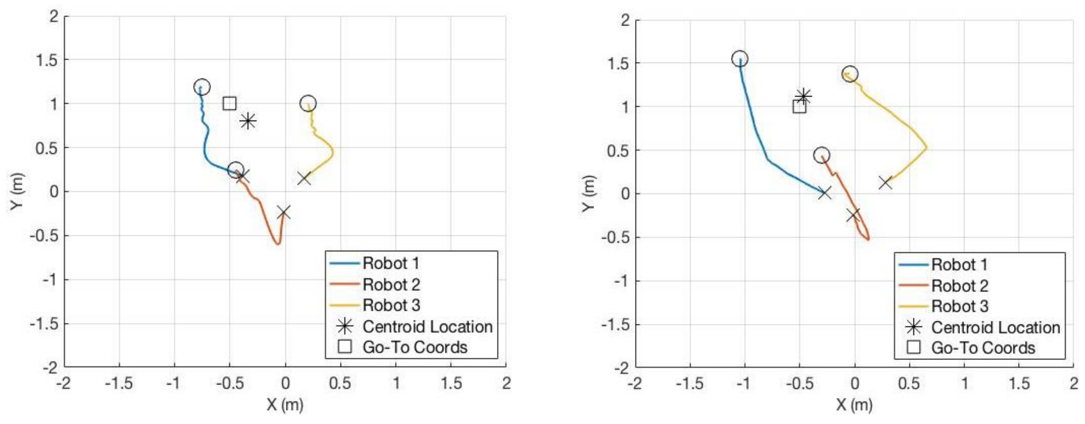


Figure 26: Behavior simulation (left) and validation in testbed (right) for go-to coordinates behavior.

3.2.3 Attract/Disperse

The attract behavior causes robots within the swarm to move closer to other robots, while the disperse behavior causes them to spread out from each other. While these are useful functionalities on their own, it should also be noted that the attract behavior is an important part of the AN strategy as it allows for the swarm to stay within communication range of each other while performing the desired AN task. As will be shown in the results section, the attract function can dramatically improve the performance of AN behaviors.

The first step of the attract behavior is a response based on the distance and bearing of a robot *i* relative to all other robots *j:*

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \qquad \text{Equation 5}$$

$$\phi_{ij} = \arctan\left(\frac{y_j - y_i}{x_j - x_i}\right) \qquad \text{Equation 6}$$

Where d_{ij} is the distance from robot *i* to robot *j* and ϕ_{ij} is the bearing angle from robot *i* to robot *j*. The distance d_{ij} is used to determine which robots are in range, as discussed in section 3.1. The bearing angle is then used to determine a velocity in the direction of each in-range robot, with constant speed *V*:

$$v_{axij} = Vcos(\Phi_{ij})\ ;\ v_{ayij} = Vsin(\Phi_{ij}) \qquad \text{Equation 7}$$

where v_{axij} is the x-component and v_{ayij} is the y-component of the attraction velocity of robot *i* with respect to *j*. The attract/disperse function is then a simple summation of the velocity components in (7) for all robots that lie within the designated sensor range R_s, or:

$$\text{For all robots } j \text{ s.t. } d_{ij} < R_s\text{: } V_{ax} = \sum_{j=1}^{N} v_{axij}\ ;\ V_{ay} = \sum_{j=1}^{N} v_{ayij} \qquad \text{Equation 8}$$

This behavior is illustrated in Figure 27. In this 5-robot configuration, note that all robots are within range of each other with the exception of the leftmost robots. Therefore the aggregate “attraction” for the bottom left robot is based off the information from the 3 robots to its right, and not the robot directly above it.

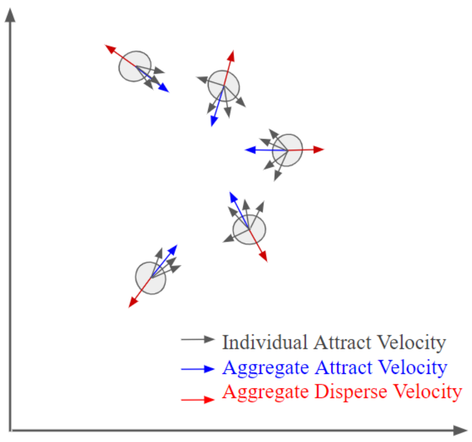

Individual Attract Velocity
Aggregate Attract Velocity
Aggregate Disperse Velocity

Figure 27. Schematic of Attract/Disperse behavior for *N=5* robots

The disperse behavior can be thought of as the direct opposite of the attract function. The same aggregate vector in Equation 7 is calculated, with the disperse behavior simply going in the negative sense of that vector.

The attract behavior was first verified in simulation and validated in the testbed for varying obstacle avoidance and sensor range values. Figure 28 indicates the position history for the robots when using an obstacle avoidance range of 1 unit (left) and 0.5 units (right). In both cases, the robots, as a result of the selected behaviors, achieve a formation that minimizes the distance between themselves and all other robots.

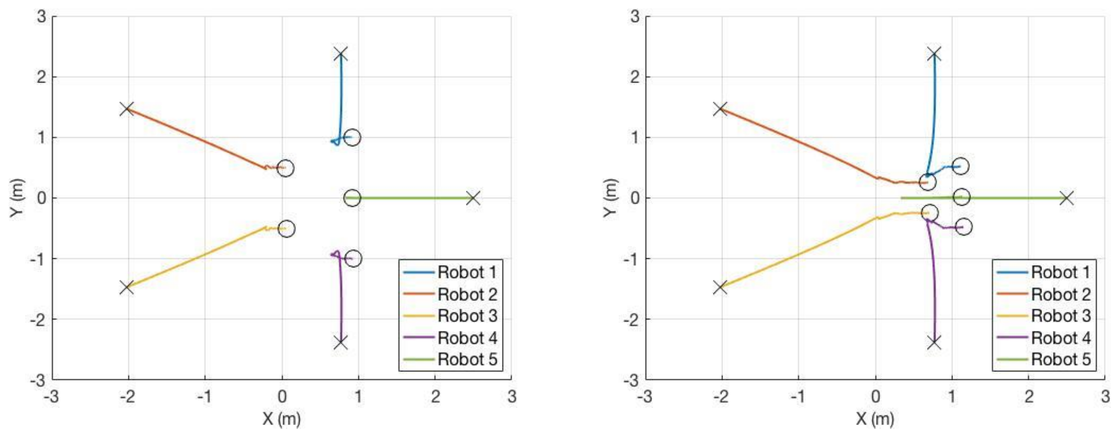


Figure 28. Simulation of attract with obstacle avoidance range of 1 m (left) and 0.5 m (right)

After vetting the attract behavior in simulation, it could be validated using the experimental testbed, the time histories of which are shown in Figure 29.

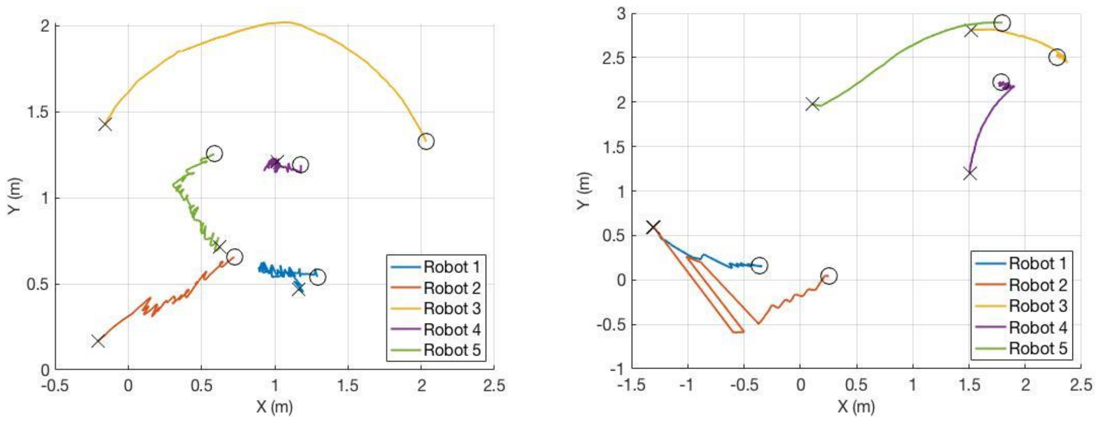


Figure 29. Testbed Validation of Attract Behavior

The testbed results show two key results. The first, as seen in the left image of Figure 29, is that the initial conditions can have a dramatic effect on individual robot behaviors, as is seen in the case of robot 3. While robot 3 still successfully completes the attract behavior, it does so by "orbiting" robots 4 and 5, rather than having the roughly linear behavior observed in the simulation. The second image on the right indicates a "clumping" that happens when the robots are out of communication range of each other. Because robots 1 and 2 are out of range of 3, 4, and 5, they separately complete the attract behavior without use of the other robots' position information.

The "opposite" behavior of the attract behavior is the disperse behavior, which is similarly tested and verified, as shown in Figure 30.

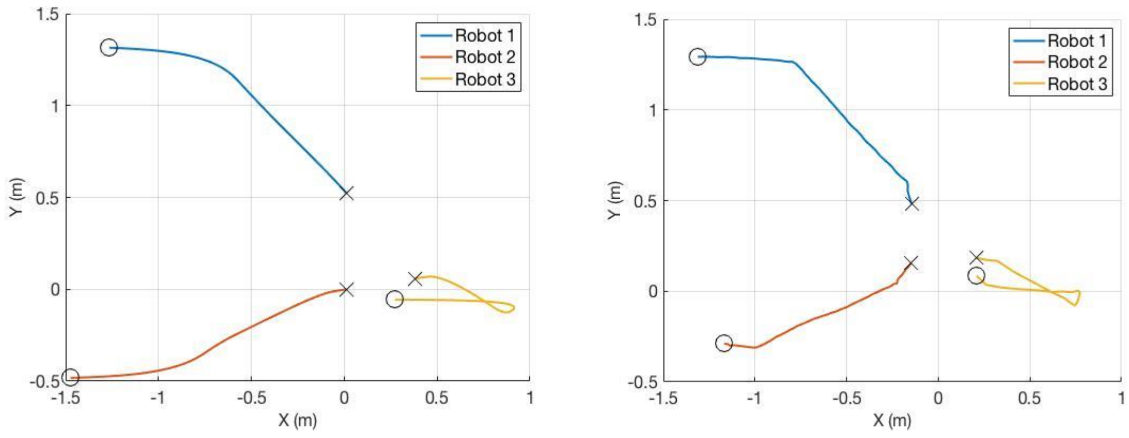


Figure 30: Simulation (left) and testbed validation (right) of disperse behavior

In both cases, the robots move away from each other until they are all out of range of each other, at which point they "wander" in the negative x direction. As seen in the figures, the testbed validation closely matches the simulated behaviors.

3.3 Swarm Composite Behaviors

Composite behaviors are formed by combining multiple primitives together. In most cases, this involves simple multiplication or addition, or a combination of the two. The main exception to this rule is the case of Out-of-Range Determination, which is implemented only in simulation to conform to the swarm definition presented in 1.2.3. Though not listed in the formulation of any of the composites, it should be noted that this is used for all behaviors, so that only agents that are "in-range" are used when calculating the behaviors. Obstacle Avoidance is also assumed to be "turned on" for all behaviors, and is added to the final resultant velocity of the behavior control laws.

3.3.1 Flocking

The natural response of the swarm in go-to-coordinates mode is to attract to a certain point and space out around that point based on the robot obstacle avoidance range. However, for some applications it may be desired that the swarm group together and travel as a pack rather than meeting at the specified coordinates. For such an application, the robot bearing can be commanded as a sum of the attract and go-to behaviors, as shown in equation 9:

$$V_{fli} = K_a V_{ai} + K_g V_{gi} \qquad \text{Equation 9}$$

where K represents a gain that can be scaled depending on desired weighting of behaviors. For example, a large K_a would result in the swarm grouping together and then moving towards the goal coordinates, while a smaller K_a would lead to a more gradual convergence. A simple schematic for flocking is shown in Figure 31, and compared to the primitive go-to coordinates.

Rather than going straight to the desired coordinates as shown in Figure 25, the flocking behavior is designed to cause the robots to group together and then head towards the goal coordinates, or to do so simultaneously. The preliminary simulation verification of this is shown in Figure 32.

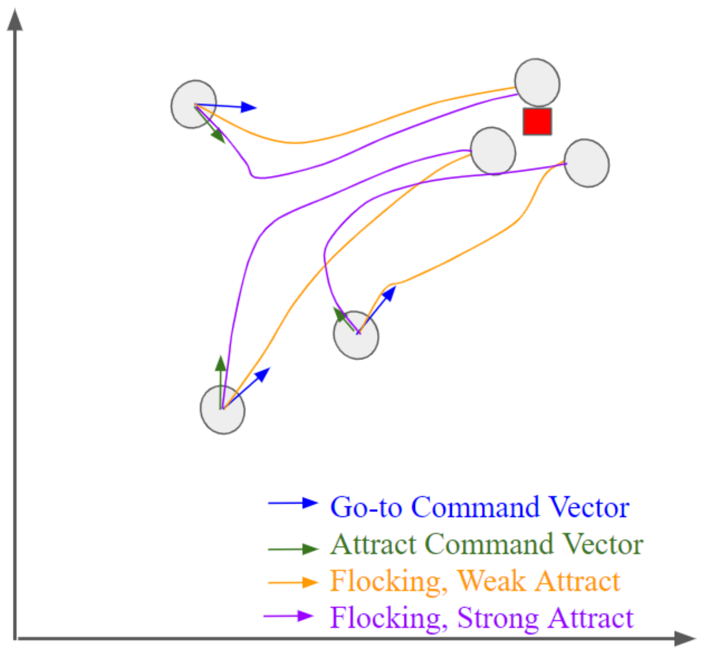


Figure 31. Schematic of Flocking behavior for varying weighting of attract behavior

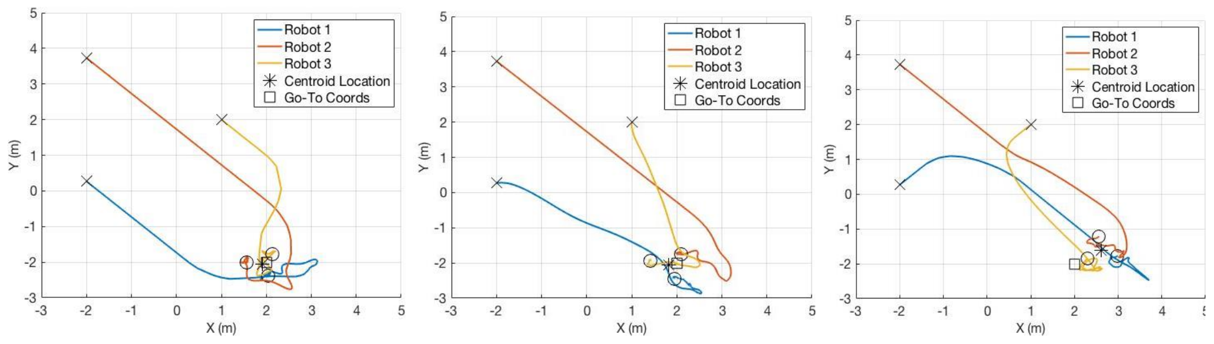


Figure 32. Simulation verification of Flocking behavior for varying weighting of Attract

The image on the left is a “pure” go-to primitive, while the middle image has added a weak attract (K_a= 0.5) to the behavior composite. The paths of robots 1 and 3 are influenced by robot 2’s location, as they are attracted back to the group rather than directly to the goal coordinates. This is seen even more clearly when using a strong attract (K_a=1.5) as shown in the right image.

3.3.2 Source-Seeking (Find Min / Find Max)

As was discussed in the background on adaptive navigation (section 1.2), it is often desired to find the local or global extrema (minima or maxima) within a scalar field. In SANS, this is accomplished by implementing a source-seeking composite behavior. The source-seeking composite behavior is based off differential sensing, and implicitly utilizes an approximation of the local gradient sensed by the swarm. The goal of the source-seeking composite behavior is to compare robot i's sensor value to the sensor values of all other in-range robots *j*, and then move toward the robots with higher values (for find max) or lower values (for find min). To accomplish this, the attract vector (equation 7) is scaled by the sensor value comparison scalar (equation 2), yielding the following behavioral composite:

$$v_{fmij} = \Delta SV_{ij} \ * v_{aij} \qquad \text{Equation 10}$$

where v_{fmij} is the resultant source seeking velocity for the robot pair *ij*, ΔSV_{ij} is the sensor value comparison as defined in equation 2, and v_{aij} is the attract velocity for the robot pair *ij* as defined in equation 7. The source-seeking velocity is calculated from robot *i* to all other robots *j*, and then summed as shown in equation 8. A visual schematic for this is shown in Figure 33.

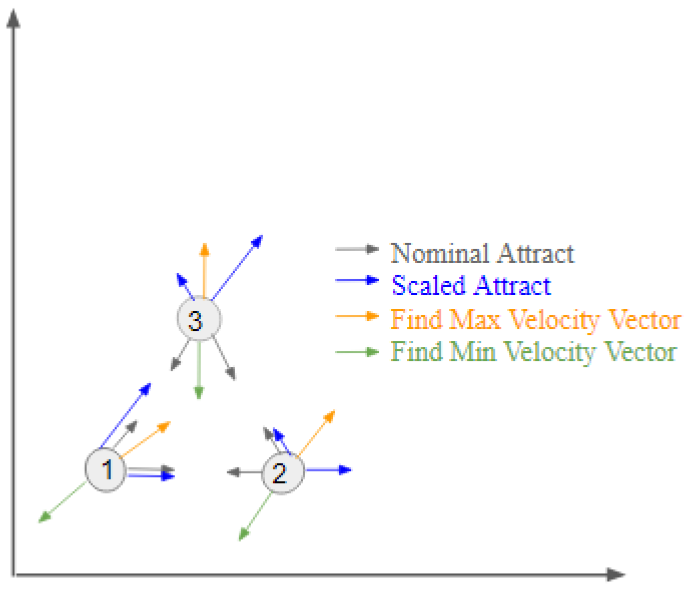


Figure 33. Schematic of Find Min/Find Max behavior for *N=3* Robots

Figure 33 depicts a source-seeking behavior calculation for a single time-step of a simulation. Robot 1 (bottom left) senses a scalar field value of 1, Robot 2 (bottom right) a value of 2, and Robot 3 (top center) a value of 3. The nominal attract vectors are of equal magnitude and show how the robots would be drawn together without scaling, while the scaled attract vectors show how this is scaled by the sensor value difference. The find max aggregate velocity V_{fij} is then the sum of the individual scaled attract vectors. This figure highlights two key characteristics of the source finding algorithm. The first is that all robots are using the same behavior. This means that while the robots with smaller sensed scalar field readings are drawn towards the robots with higher readings, the robots with higher readings are equally “pushed” away from the robots with lower readings. The second is that the find min behavior is the same as the find max behavior, with the exception being the commanded velocity V_{fmij} is in the negative sense of the vector. This again relates to the formulation of the sensor-value comparison primitive (Equation 2).

The find min and find max are both verified in simulation, as seen in Figures 34 and 35. The figure on the left indicates the find min behavior for 3 robots on a “single sink” scalar field- there is only one minimum in the scalar field. The time history of the sensor values is shown to the right, and indicates that the robot sensor values move down gradient towards the minimum. The same behavior is run for find max on a “single source” behavior, and the simulation results are shown in Figure 35.

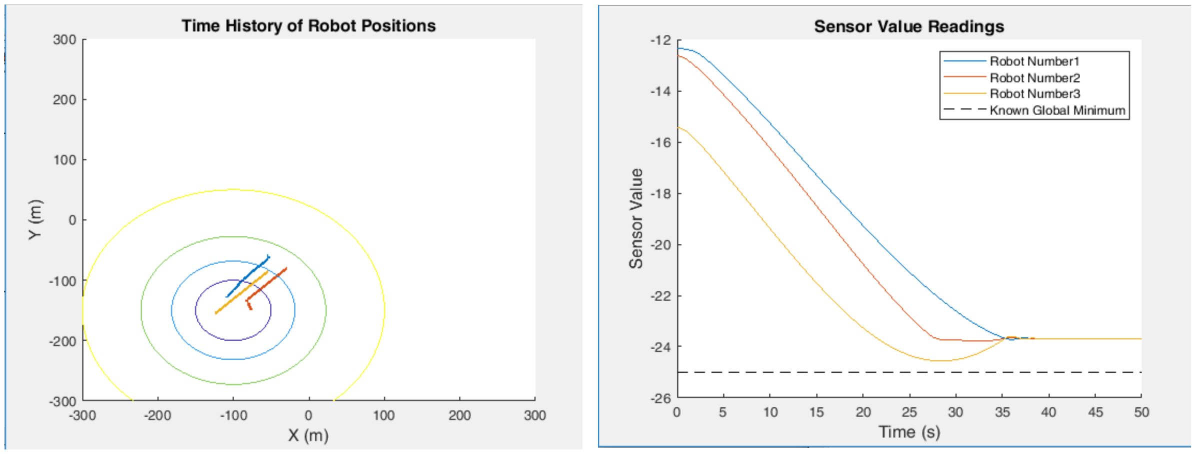


Figure 34. Simulation: Find Min position (left) and sensor value history (right) for single sink

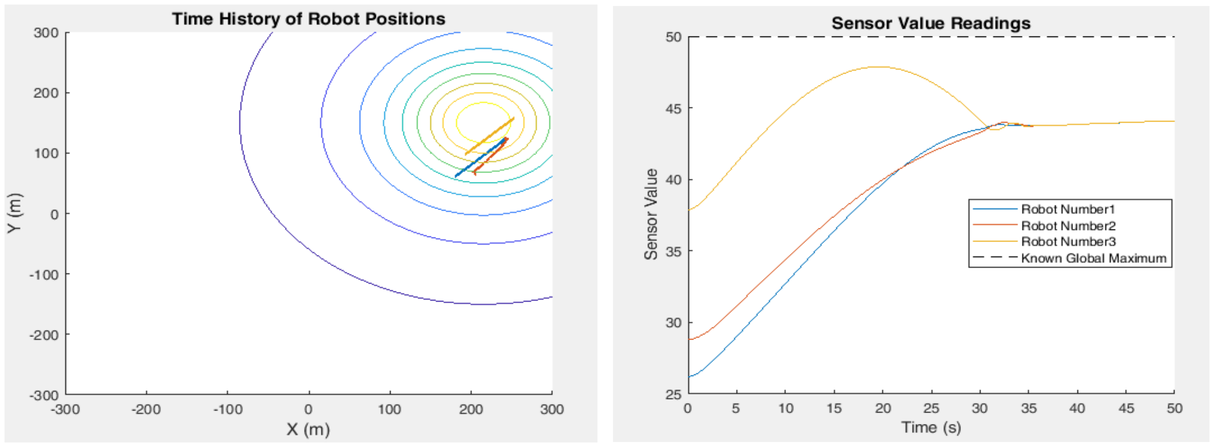


Figure 35. Simulation: Find Max position (left) and sensor value history (right) for single source

The find min behavior has been validated on the testbed with *N=3* robots, shown in Figure 36. The left image shows the robots successfully moving down the gradient, while the right image shows the sensor value readings of the individual robots approaching the known global minimum.

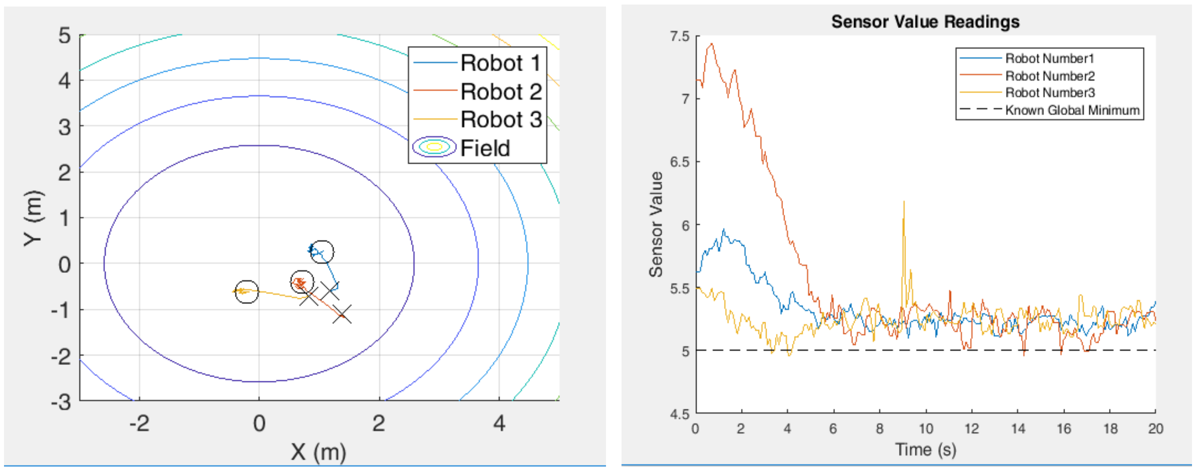


Figure 36. Testbed validation of find min behavior

Figures 34-36 demonstrate the functionality of the find min and find max composite behaviors, both in simulation and on hardware. They also indicate the steady-state behavior of the extrema finding behavior, particularly in the right image of Figure 35. While robots do move up the gradient, they eventually reach a point where they lose gradient definition due to all being situated on the same contour. Approaches to improve this steady-state behavior are discussed in Chapter 4, which include increasing the number of robots and stacking of additional behaviors.

3.3.3 Find/Follow Contour

During AN, it may be desired to find and follow a contour in the scalar field. For example, tracing and mapping the contour of a pollutant may indicate how far a pollutant has spread, and can therefore indicate areas that may be safe or unsafe for human occupation. With this in mind, a composite behavior to find and follow a contour was developed.

The contour finding and following composite behavior is a two-step process. The first step is for robot *i* to self-identify whether it is on the contour, above the contour, or below the contour. This is determined by comparing the robot's sensed scalar field value to the desired contour value, SV_{des}. Each robot first checks if it is "on the contour." This could be done by simply checking if the robot's sensed scalar field value is equal to SV_{des}. However, this would require that the sensor value exactly match the desired contour value. Therefore, a "buffer range," SV_{buff}, is established. The robot then checks if it is on the contour by calculating its sensed scalar field value in reference to the buffered contour value. Namely, robot i is designated as on the contour if:

$$SV_i < SV_{des} + SV_{buff} \text{ and } SV_i > SV_{des} - SV_{buff} \qquad \text{Equation 11}$$

If the robot is not on the contour, then the robot must determine if it is above the contour or below the contour. If the robot's sensed scalar field value is greater than SV_{des}, then the robot is above the contour, and if the robot's sensed scalar field value is less than SV_{des}, then the robot is below the contour. At the end of this process, robot i has successfully established its "contour state" as either on contour, below contour, or above contour. Once robot i's contour state has been established, the robot moves in the direction of the contour, or along the contour if it is on the contour. The formulation of the commanded velocity is summarized in Table 3. It should be noted that for V_{fci} for the "On Contour" contour state in Table 3, the formulation indicates a rotation of vector V_{fi} by $\frac{\pi}{2}$ radians.

Table 3. Contour State Determination

Robot *i* Sensor Value	**Contour State**	**Behavior**	**Formulation**
$SV_i > SV_{des} + SV_{buff}$	Above Contour	Find Min	$V_{fci} = -V_{fmi}$
$SV_i < SV_{des} - SV_{buff}$	Below Contour	Find Max	$V_{fci} = V_{fmi}$
$(SV_{des} - SV_{buff}) < SV_i < (SV_{des} + SV_{buff})$	On Contour	Follow Contour	$V_{fci} = V_{fi} \pm \pi/2$

In the above equations, v_{fci} is output velocity for robot i, and V_{fi} is the summation source-seeking velocity as defined in equation 8. As was discussed in section 3.2.2, the positive sense of V_{fi} results in gradient ascent, or find max behavior, while the negative sense results in gradient descent, or find min behavior. In order to move along the contour, it is not desired to move up or down gradient, but perpendicular to the gradient. Therefore to move along the desired contour, it is required to move perpendicular to the gradient direction approximated by V_{fi}, which is accomplished by adding $\pm\frac{\pi}{2}$ radians to the commanded bearing angle. The choice of positive or negative $\pm\frac{\pi}{2}$ will affect the direction of encirclement of the contour, with positive relating to counterclockwise travel about the contour, and negative relating to clockwise travel. A simplified schematic in Figure 37 depicts the contour finding/following composite behavior for 3 robots.

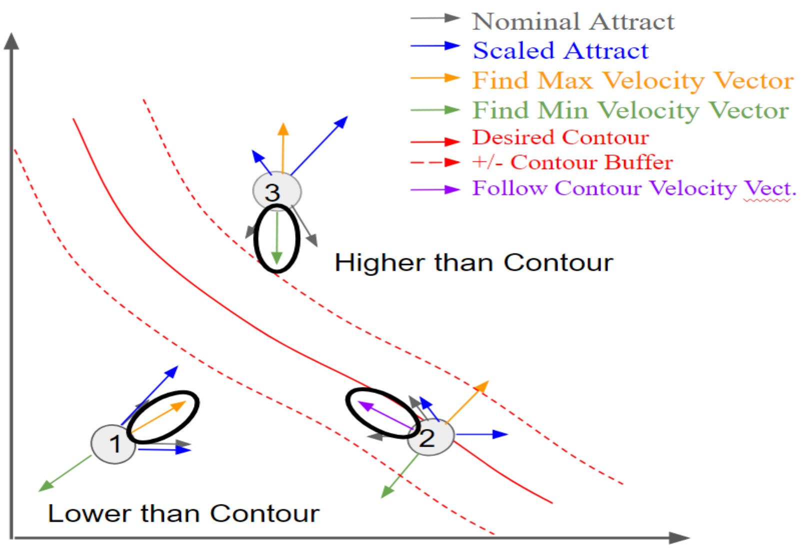


Figure 37. Schematic of Find Contour Behavior

In the schematic of Figure 37, there are again 3 robots with their sensor values indicated, similar to Figure 33. In the given scenario, robot 2's sensor value lies within the range of the buffered desired contour, robot 3 is above the contour, and robot 1 is below the contour. The circled vector indicates the final velocity that each robot will use. Robot 3 is above the contour and therefore uses a find min vector, robot 1 is below the contour and uses find max, and robot 2 is within the contour bounds and uses the follow contour vector. As with other behaviors, the contour behavior was simulated and validated in the testbed, as shown in Figures 38 and 39.

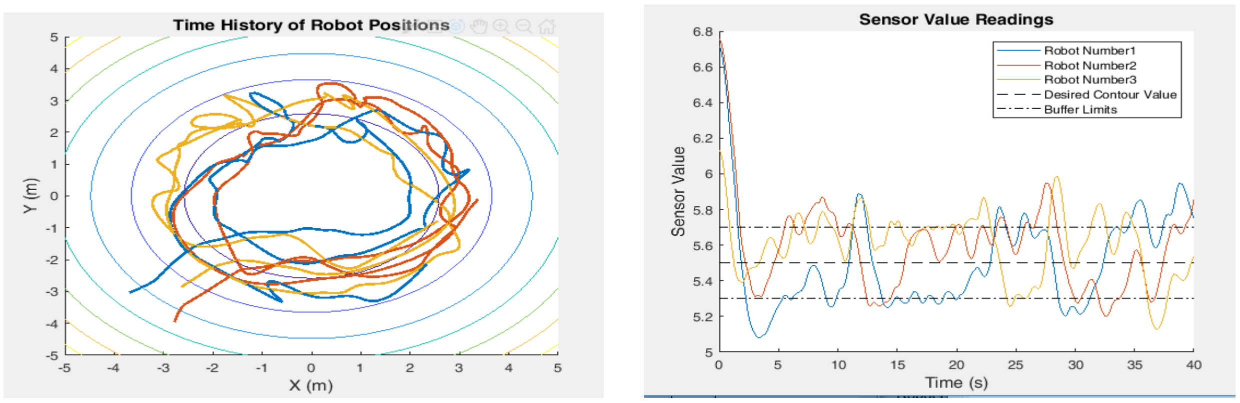


Figure 38. Simulation position (left) and sensor value history (right) for Contour Following

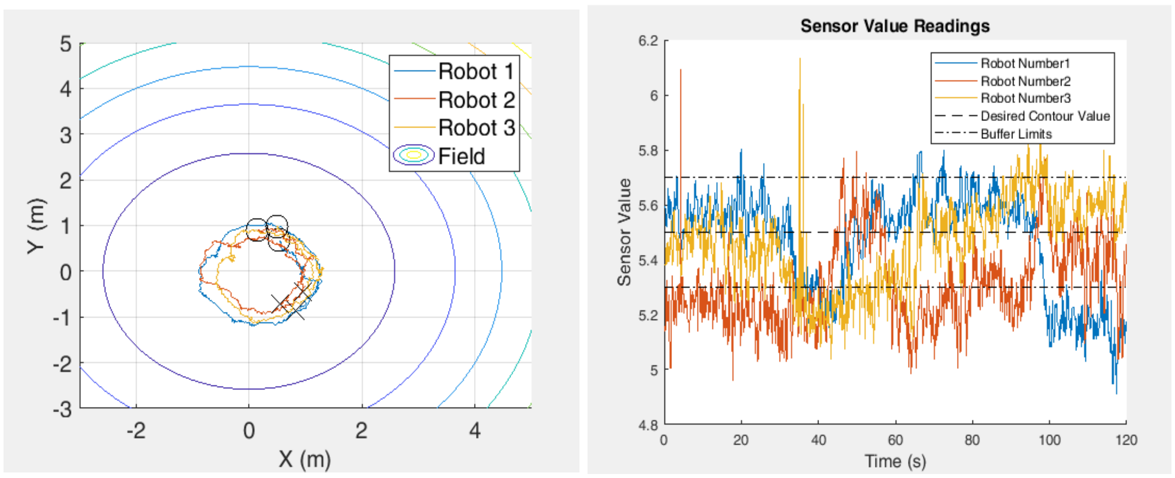


Figure 39. Testbed Validation position (left) and sensor value history (right), Contour Following

In both simulation and the testbed, the swarm successfully follows the contour, and traces a full circle that maps the contour value. The testbed experiment in particular begins to indicate a few design parameters for use of the contour following, namely:

1. *Contour buffers*- the contour buffers must be large enough to allow for the variation in the sensor value signal due to noise.
2. *Avoidance Range*- The avoidance range must be balanced with the distance over which the contour buffer occurs. For example, if an avoidance range of 5 m is set, the dictated contour buffer should occur over a distance of greater than 5 m, or the avoidance will override the contour behavior before the robots can converge on the contour.
3. *Robot Speed* - The robot speeds must be set low enough that they do not overshoot the desired contour value before they can “continue on” along the contour that they have found.

These preliminary design parameters are tested and expanded in Chapter 4.

4. Experimentation with Adaptive Navigation Behaviors

Chapter 3 demonstrated the functionality of primitive and composite behaviors in simulation and confirmed that the simulated behaviors were reproducible when used on the Decabot indoor testbed system. The intent of Chapter 4 is to describe the effects of changing design parameters and to apply the behaviors to a more complicated scalar field. The scalar field used in this section is shown in Figure 40, adapted from [34]. This scalar field is desirable for experimenting with adaptive navigation behaviors because it has multiple extrema, as well as ridges and trenches.

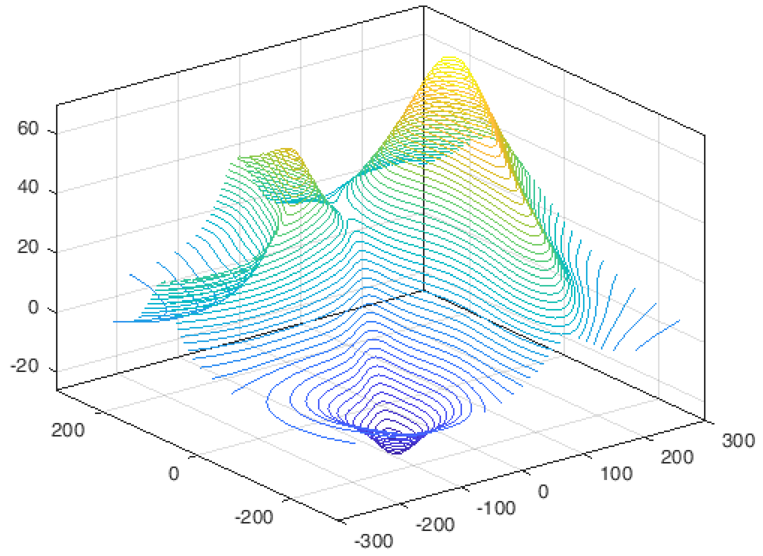


Figure 40. Scalar Field used for experimentation of behaviors [34]

4.1 Extrema Finding

Chapter 3.2.2 indicated the capability of the swarm to move up or down a gradient based on local, real-time scalar field readings. As was seen in Figure 34, it is possible that the swarm robots may be arranged such that they are on a contour, and therefore all have the same scalar field reading. This scenario yields swarm stagnation, and the swarm may not reach the scalar field extrema in this case. For this reason, Extrema Finding entails stacking the Find Min/Find Max behavior with the Attract behavior. This serves two purposes. First, in the case of the robots aligning on a contour, the attraction force would cause them to “dither” off the contour, which allows for differential readings. Second, the attract function serves as an implicit low degree of freedom formation control method to keep the swarm within communication range of each other.

In order to quantitatively measure the effects of changing parameters on the swarm's overall extrema finding capability, performance metrics are introduced that are similar, but not identical, to those used to characterize 2nd-Order system dynamics.

- *Rise Time* - the rise time for the swarm is defined as the amount of time it takes the average sensor value reading of the swarm to reach 90% of the extreme value. For example, if the initial average reading is 5, and the known maximum is 70, then the rise time is the time at which the average sensor value first reaches 0.9*(70-5) + 5 = 63.5. The rise time gives an indication of the speed of response of the swarm.

- *Steady-State Error* - The steady-state error is found as the difference between the known global extreme value and the swarm's average steady-state sensor value. The steady-state error indicates how well the swarm has found the extreme value.

These quantitative metrics are used in addition to qualitative analysis of the system response to show how extrema finding is affected by design parameters such as number of robots in the swarm, obstacle avoidance range, and variation of initial conditions. Because rise time and steady-state error may be dependent on the size of the feature and how far the swarm must travel to reach the feature, they are more useful when comparing small changes to a single design parameter while keeping the other design parameters constant. By keeping the other parameters constant, the rise time can indicate how efficiently the swarm is moving towards the extrema, or how direct its path is.

While these performance metrics are limited, they help quantify the effect of changing design parameters to achieve top-down specification of swarm parameters. Further possible performance metrics are presented in Chapter 5. To the author's knowledge, this is the first published presentation of quantitative metrics for swarm performance.

4.1.1 Effect of Number of Robots on Extrema Finding

The extrema finding behavior outlined in 4.1 was tested for both maxima and minima finding on the scalar field presented in Figure 40. As discussed in Chapter 1, one of the main defining characteristics of swarm robotics is that the swarm utilizes a large number of robots, and that the overall swarm behavior is improved with an increased number of individual swarm robots. For extrema finding, performance is improved for an increasing number of swarm robots because the increased number of robots yields a better estimation of the local gradient. This is demonstrated in simulation by scaling the number of robots from 3 to 10 and observing the behavior of the swarm. All other parameters were held constant, as shown in Table 4. The initial conditions of

(*R*,*x*,*y*) indicate the swarm robots are initially placed such that they are evenly spaced along the circumference of a circle with radius *R* centered at (*x*, *y*).

Table 4. Simulation Parameters for Scaling Number of Robots for Extrema Finding

# of Robots	*Simulation Time(s)*	*Sensor Range(m)*	*Avoidance Range(m)*	*Robot Speed (m/s)*	*Initial Conditions*
N	20	150	10	45	(5,5,5)

Qualitatively, it can be shown that the swarm extrema finding performance improves as the number of robots increases. For example, Figures 41 and 42 show the swarm response for Maxima Finding with 3 robots and 10 robots, respectively. During the given simulation interval, the larger swarm in Figure 41 locates the maximum faster and more effectively, which can be seen in the position history, as well as the fact that all robots in the swarm have a higher sensor value reading than at the end of the simulation for 3 robots.

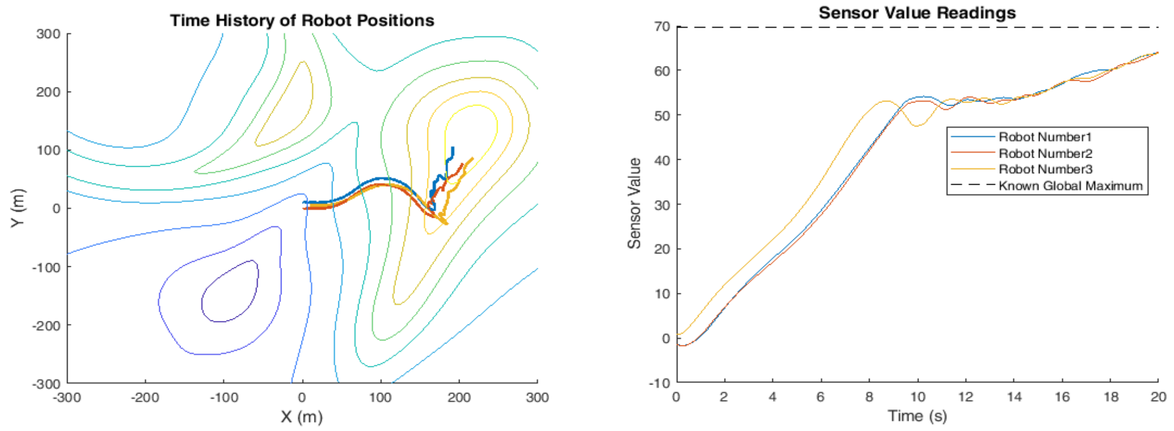


Figure 41. Max Finding for *N=3* Robots

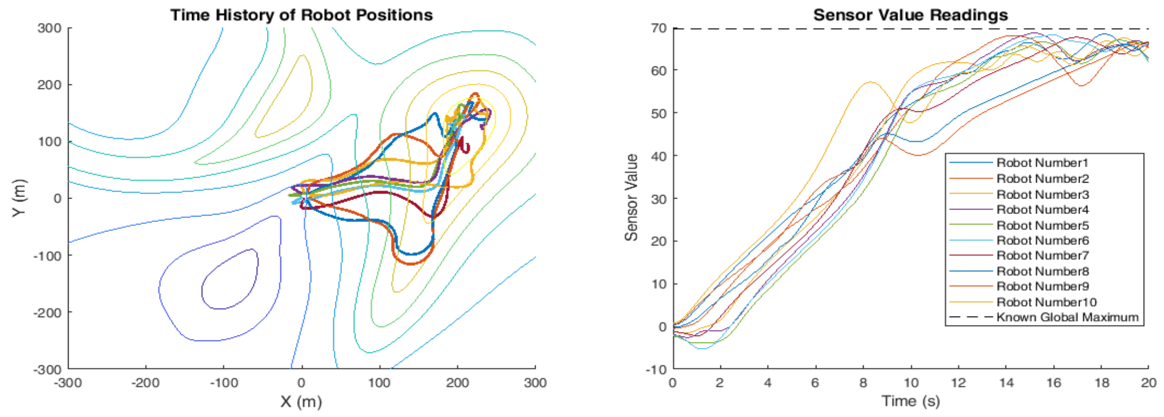


Figure 42. Max Finding for *N=10* Robots

The same phenomenon can also be seen for Minima finding, shown in Figures 43 and 44.

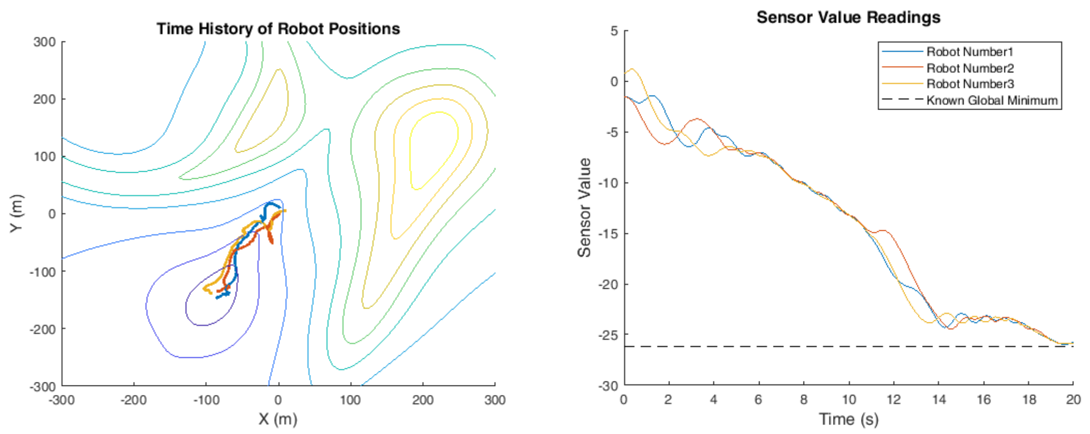


Figure 43. Min Finding for *N=3* Robots

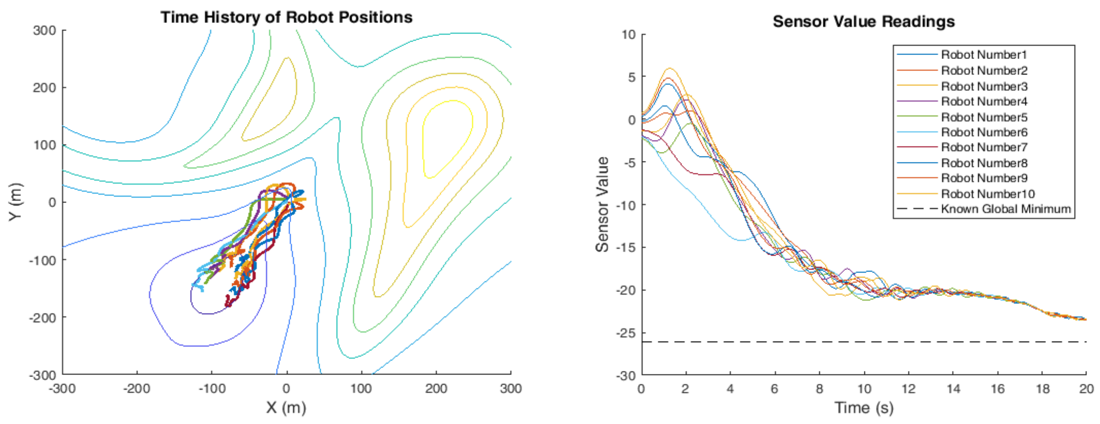


Figure 44. Min Finding for *N=10* Robots

The experimental results for increasing number of robots for maximum finding and minimum finding are shown in Table 5 and 6. Time histories for position and sensor values are shown in the indicated figures, with numbers starting with an A indicating figures in Appendix A.

Table 5. Quantitative Results for Max finding with increasing number of robots

Figure	**Number of Robots**	**Rise Time (s)**	**Steady-State Error**
41	3	19.5	6.42
A1	4	14.17	3.54
A2	5	18.59	6.31
A3	6	14.27	5.42
A4	7	13.57	6.22
42	10	13.21	3.95

Table 6. Quantitative Results for Min finding with increasing number of robots

Figure	Number of Robots	Rise Time (s)	Steady-State Error
43	3	14.87	0.89
A5	4	19.8	2.02
A6	5	12.96	0.67
A7	6	N/A	N/A
A8	7	12.26	0.51
44	10	12.16	3.1

As can been seen in Figure A7, the find min for N=6 robots and the parameters given in Table 4 does not converge to the global minimum during the simulation time. This can be attributed to the obstacle avoidance causing the robot's convergence to slow down. The rest of the data in Tables 5 and 6 indicates that the speed of response of the swarm increases as the number of robots increases, but that the steady state error does not necessarily decrease as the number of robots in the swarm increases. As was mentioned above, the speed of response is improved with more robots due to the better definition of the local gradient estimation. This better definition means that each individual robot's commanded bearing is closer to the location of the extrema than it is with less robots. The lack of improvement of steady-state response can be accounted for by the fact that the robots are limited in their ability to locate themselves directly on the location of the extrema due to their avoidance of other robots. This can be seen in Figure 44 for N=10 robots. The final position of the swarm is roughly centered on a local minimum, but the obstacle avoidance prevents individual robots from positioning themselves directly on the extremum.

4.1.2 Effect of Obstacle Avoidance Range on Extrema Finding

In addition to varying the number of robots in the swarm, the obstacle avoidance range of the robots can also be varied as a form of implicit formation control. The lower limit of the obstacle avoidance range is a function of the individual robot sizes and dynamics and must be sufficiently high to avoid robot collisions. The upper limit of the obstacle avoidance must be lower than the communication range of the robots to ensure that they do not repel to a point where they cannot communicate with each other. Simulations were run to show the effect of the obstacle avoidance range on the swarm steady-state error, using the parameters defined in Table 7.

Table 7. Simulation Parameters for varying Obstacle Avoidance Range for Extrema Finding

# of Robots	*Sim Time(s)*	*Sensor Range (m)*	*Avoid Range(m)*	*Robot Speed(m/s)*	*I.C.*
5	30	150	*variable*	45	(5,5,5)

The obstacle avoidance range was varied from 20m to 2m, with the results summarized in Tables 8 and 9. Figures 45 and 46 show the time histories for the two extremes of this range for maximum finding, and Figures 47 and 48 indicate the swarm response for minimum finding. Qualitatively, comparing Figures 45 and 46 shows that increased obstacle avoidance range does not allow the robots to group as tightly around the maxima, and as a result the individual robot sensor value readings are farther away from the known maximum. Figure 46 also indicates one of the shortcomings of using low degree of freedom formation control. A swarm formation with one robot on the extremum and the other four surrounding it would more effectively locate the extremum. The use of a low degree of freedom formation controller does not allow for this type of formation specification, and can yield final formations similar to that seen in Figure 46.

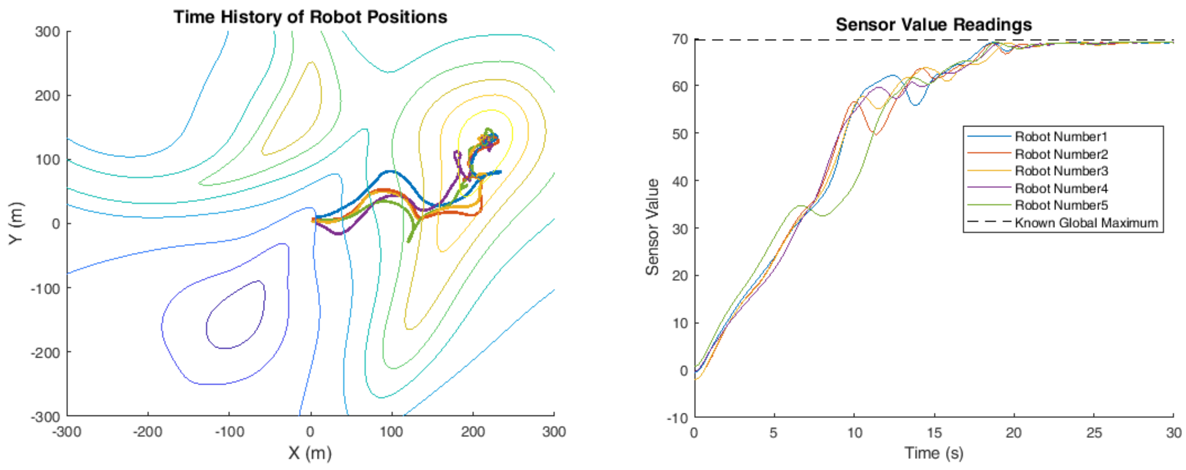


Figure 45. Max Finding for Obstacle Avoidance Range = 2m

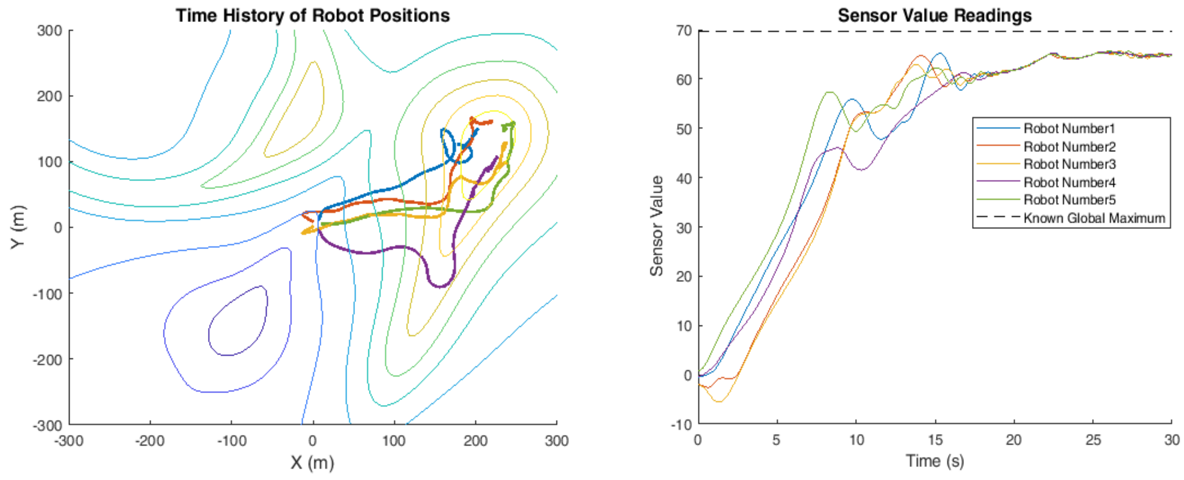


Figure 46. Max Finding for Obstacle Avoidance Range =20m

Similarly, comparing the swarm behavior for minima finding shows the same result of greater steady-state error for higher obstacle avoidance. Despite these changes in system performance, the swarm is able to locate the extrema for the variety of obstacle avoidance ranges presented in Tables 8 and 9.

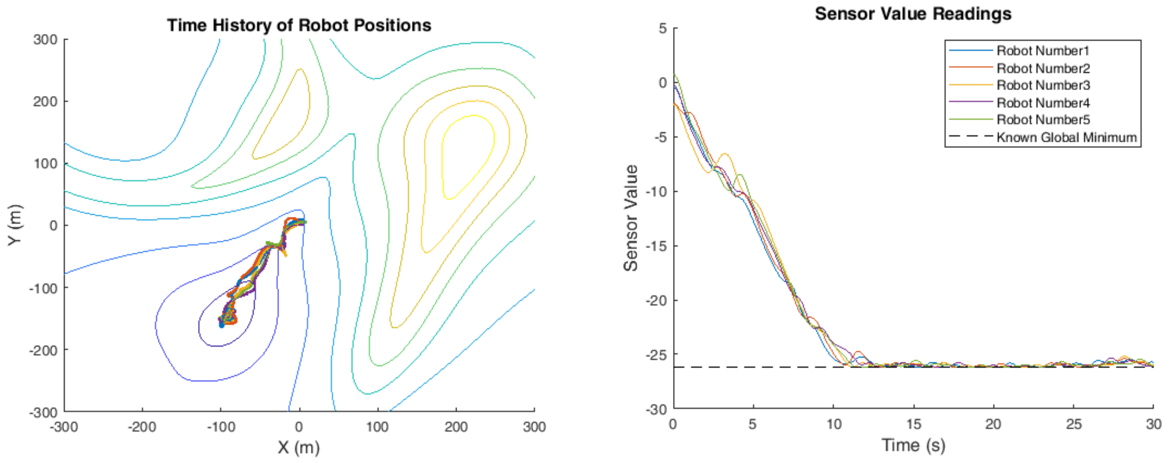


Figure 47. Min Finding for Obstacle Avoidance Range = 2m

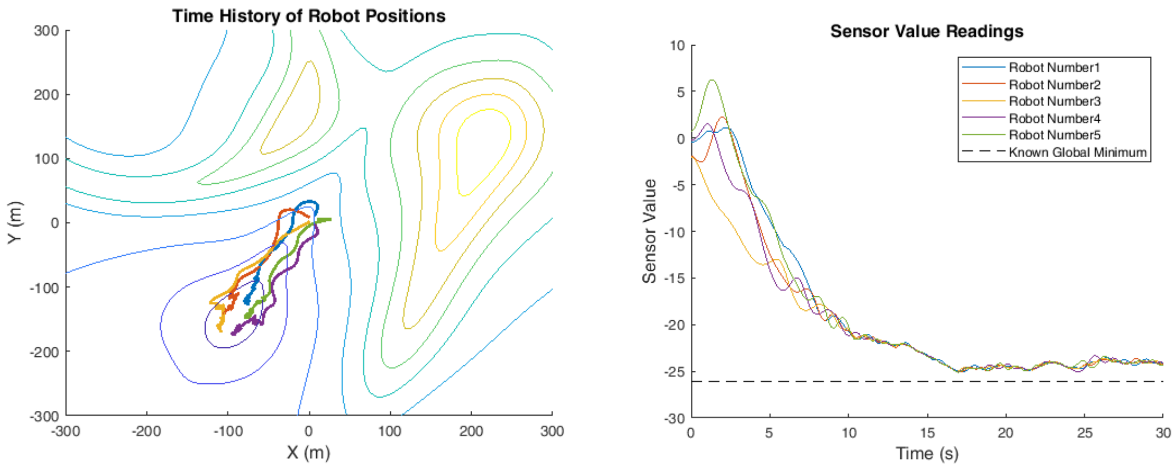


Figure 48. Min Finding for Obstacle Avoidance Range =20m

Table 8. Quantitative Results for Max finding with increasing obstacle avoidance range

Figure	Obstacle Avoidance Range (m)	Rise Time (s)	Steady-State Error
45	2	15.85	0.42
A9	5	13.75	1.15
A10	10	18.59	6.31
A11	15	19.26	2.21
46	20	21.27	4.72

Table 9. Quantitative Results for Min finding with increasing obstacle avoidance range

Figure	Obstacle Avoidance Range (m)	Rise Time (s)	Steady-State Error
47	2	9.732	0.38
A12	5	13.34	0.52
A13	10	15.21	1.18
A14	15	13.95	1.31
48	20	15.35	1.99

As was expected, the steady-state error of the system increased as the obstacle avoidance range increased. In addition, the swarm response was slowed, as seen by the rise time increasing as the obstacle range increased, because the obstacle avoidance acts as another "constraint" on the swarm robot motions that prevents them from moving directly toward the extrema location. As a result, lowering the obstacle avoidance range can help the swarm more precisely locate the extrema location, but a range that is too low may cause lack of gradient definition and collisions.

4.1.3 Repeatability of Extrema Finding for Varying Initial Conditions

Because AN involves moving to an unknown goal location, it was necessary to validate that the swarm could locate extrema from a variety of initial locations in the field. In addition, placing robots in varying locations with respect to each other (in opposition to the default spacing along the circumference of a circle) demonstrated that the swarm was capable of locating extrema for different starting locations and relative orientations in the scalar field. To achieve non-circular placing in the scalar field, initial conditions were selected via the manual selection rather than by using the default initial conditions. Figures 49 and 50 below show the position history for two such initial conditions.

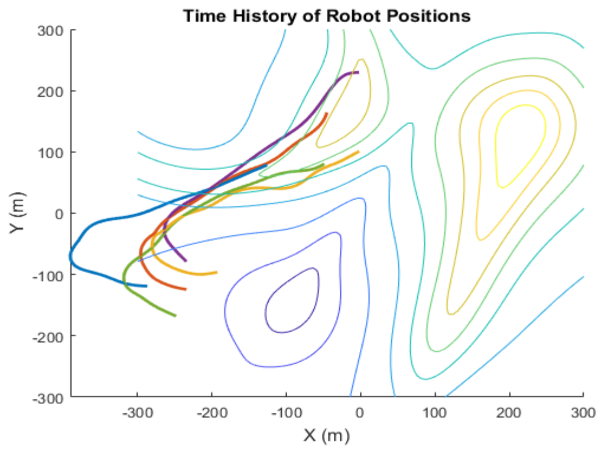


Figure 49. Max Finding for non-circular initial conditions in bottom left quadrant

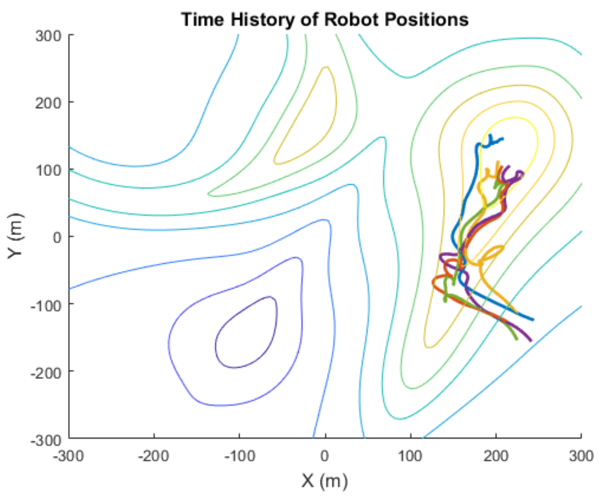


Figure 50. Max Finding for non-circular initial conditions in bottom right quadrant

Figure 49 shows that the swarm will converge to a local maximum, but this is not necessarily the global maximum value. Despite this, the swarm history in Figure 49 shows the successful climbing of the gradient to the local maxima for a different initial condition than was used in sections 4.1.1-4.1.2. In addition, comparing the swarm behaviors in Figures 49 and 50 reveals that they exhibit the same swarm-level behavior. Both swarms start up the steepest path of the gradient until they reach a ridge. The swarms then oscillate about the ridge while progressing up the gradient to their respective maxima. This behavior is expected from the design of the extrema-finding behaviors, which causes the individual robots to move in the direction of the gradient. Including the trials shown in Figures 49 and 50, a total of 10 trials were run for varying initial conditions, summarized in Table 10. The “random” initial condition selection places the robots randomly in the field, as shown in Figure 51. For the purposes of these tests, the communication range was set to the total field width (600m) so that robots are initially within communication range of each other.

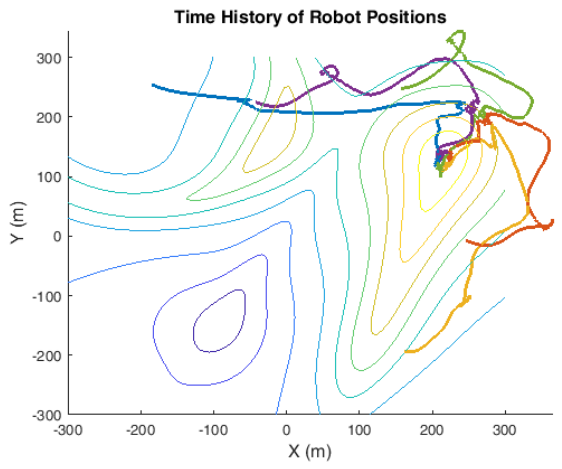


Figure 51. Max Finding for Random Initial Conditions

Table 10. Simulation Parameters for varying initial Conditions, Maxima Finding

Figure	*# of Robots*	*Simulation Time (s)*	*Sensor Range(m)*	*Avoidance Range(m)*	*Robot Speed(m/s)*	*Initial Conditions*
A13	5	60	600	10	45	(5,5,5)
49	5	60	600	10	45	Manual
50	5	60	600	10	45	Manual
A27	5	60	600	10	45	(100,5,5)
A28	5	60	600	10	45	(100,350,350)
51	5	60	600	10	45	Random
A29	5	60	600	10	45	Random
A30	5	60	600	10	45	Random
A31	5	60	600	10	45	Random
A32	5	60	600	10	45	Random

For all conditions in Table 10, the swarm successfully located a maximum, as is visible in the position and sensor value histories presented in each accompanying figure. Of particular interest is the swarm's success for random initial conditions, such as in Figure 51. The successful location of the maximum from random initial conditions indicates that the extrema finding behavior is not dependent on specific initial conditions. Additionally, using the attract behavior in conjunction with extrema finding helps the swarm locate the maximum despite an initial distribution that is large relative to the dominant spatial wavelength of the scalar field. Chapter 5 will discuss additional work that could be done to further guarantee the functionality of the swarm to varying initial conditions.

4.2 Contour Finding and Following

Contour finding and following is an important adaptive navigation behavior for applications where it is desired to map how far a pollutant has spread, or to maintain a constant desired value. Whereas the two main design criteria for swarm performance in extrema finding are the number of robots and the obstacle avoidance range, contour finding has additional design criteria in the form of desired contour and contour buffer range. The effect of varying each of these is shown in the subsequent sections, and the contour finding behavior is shown to be robust to varying initial conditions and desired contours. As in 4.1, a set of performance evaluation metrics are introduced, namely:

- *Time to Contour* - The time to contour is the time it takes for all robots to reach the desired contour value, plus or minus the contour buffer. This gives an indication of the speed of response of the swarm, with lower times indicating faster responses. As with the Rise Time presented in 4.1, this can help indicate how directly the swarm moves to the contour for swarms starting from the same initial conditions.
- *Percent of time in Contour Buffer*- The percentage of time, after initially finding the contour, that the robots are within the contour buffer range. This is a measure of the ability of the swarm to closely follow the contour, with higher percentages indicating more precise following of the contour. The percent of time in the contour buffer is determined by first taking the amount of time that each robot is within the buffer after initially reaching the contour, and then taking the average of each of these percentages for all robots in the swarm.
- *Distance travelled along contour*- Expressed as a percent of the contour length, the distance travelled along the contour gives an indication of how quickly the swarm is progressing along the contour. A larger distance indicates that the swarm has progressed farther along the contour, which could indicate more efficient mapping of the contour in a real-world scenario.

4.2.1 Effect of Number of Robots on Contour Finding and Following

As discussed in section 3.2.3, the contour finding and following behavior utilizes the gradient estimation that is used in extrema finding, and then goes up, down, or perpendicular to the gradient depending on the robot's current sensor value reading. As a result, it is expected that a greater number of robots will result in better swarm performance, because the gradient estimation is improved with more robots. This was tested in simulation, using the following simulation parameters:

Table 11. Simulation Parameters for Contour Following with varying number of robots

# of Robots	Sim Time(s)	Sensor Range(m)	Avoid Range(m)	Des. Contour	Cont. Buffer	I.C.
N	60	150	5	-15	1	(5.5.5)

The result of increasing the number of robots can be seen qualitatively by comparing the position and sensor value history for *N=3* robots in Figure 52 to the position and sensor value history for *N=10* robots in Figure 53. In general, the time to reach the contour is decreased by using more robots, and the distance along the contour is increased. The percent of time on the contour is roughly constant. The quantitative results of the simulations are summarized in Table 12 below, using the performance metrics outlined in 4.2.

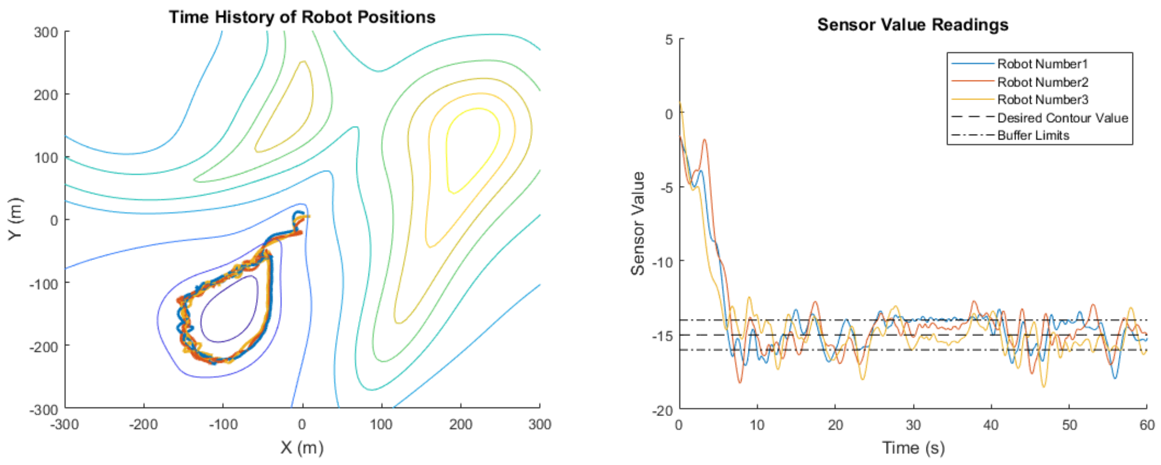


Figure 52. Contour Following and Attract for *N=3* robots

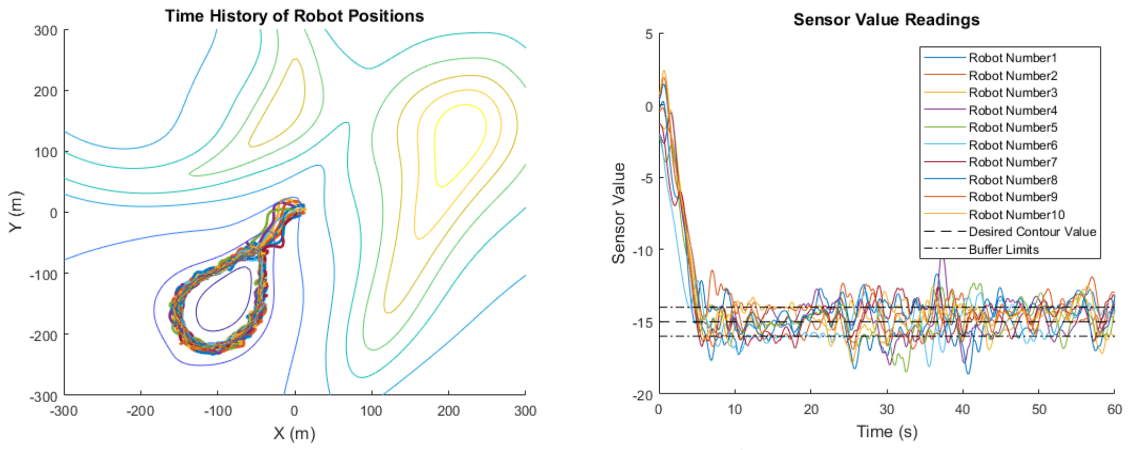


Figure 53. Contour Following and Attract for *N=10* robots

Table 12. Quantitative Results for Contour Following with increasing number of robots

Figure	Number of Robots	Time to Contour (s)	% of time on Contour	Distance along Contour
52	3	6.51	73.5	1.47
A15	4	7.012	71.3	1.56
A16	5	5.71	76.1	1.54
A17	6	5.65	74.6	1.67
A18	7	5.41	68.2	1.57
53	10	4.62	72.4	1.72

As was expected, the time to reach the contour is decreased when using a greater number of robots, and the distance travelled along the contour is also increased for a larger number of robots. This is due to the better gradient estimation from a larger number of robots. The percentage of time within the contour buffer is largely unchanged for increasing numbers of robots. This metric does not improve for larger numbers of robots because the obstacle avoidance "pushes" the individual robots out of the contour buffer range, similar to the phenomenon seen in extrema finding. This again indicates that further refined metrics may be necessary, and will be discussed in Chapter 5.

4.2.2 Effect of Obstacle Avoidance Range on Contour Finding and Following

To observe the effect of the obstacle avoidance range on contour finding and following, the obstacle avoidance range was varied while other simulation parameters were kept constant. The simulation parameters used are shown in Table 13 below:

Table 13. Simulation Parameters for Contour Following with varying obstacle avoidance

# of Robots	Sim Time(s)	Sensor Range(m)	Avoid Range(m)	Des. Contour	Cont. Buffer	I.C.
5	45	150	*variable*	-15	1	(5.5.5)

The position and sensor value history for an obstacle avoidance range of 2 units are shown in Figure 54, and the history for an obstacle avoidance range of 20 units is shown in Figure 55. Figure 55 clearly illustrates that the larger obstacle avoidance range can cause decreased performance in the swarm, but that the swarm is still able to roughly follow the contour. The quantitative results are summarized in Table 14.

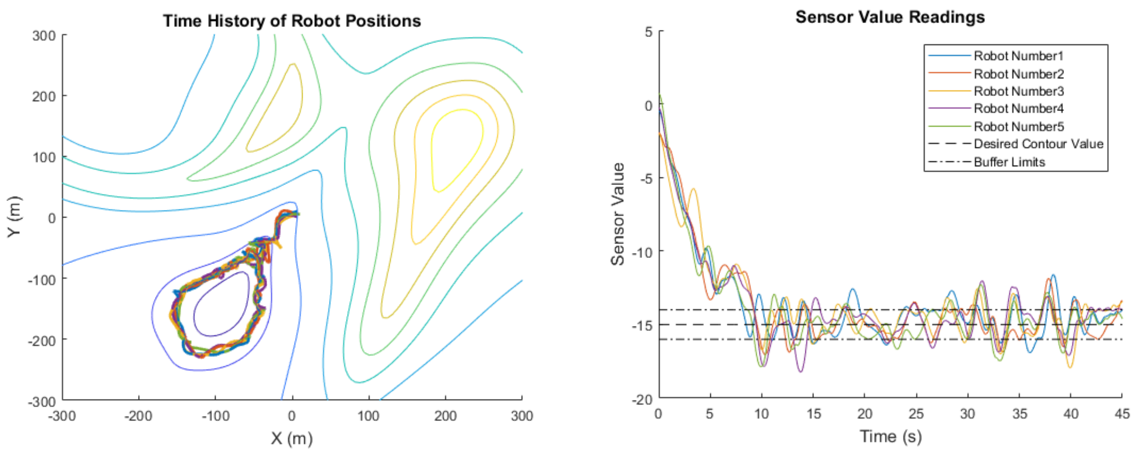


Figure 54. Contour Following for Obstacle Avoidance range of 2 m

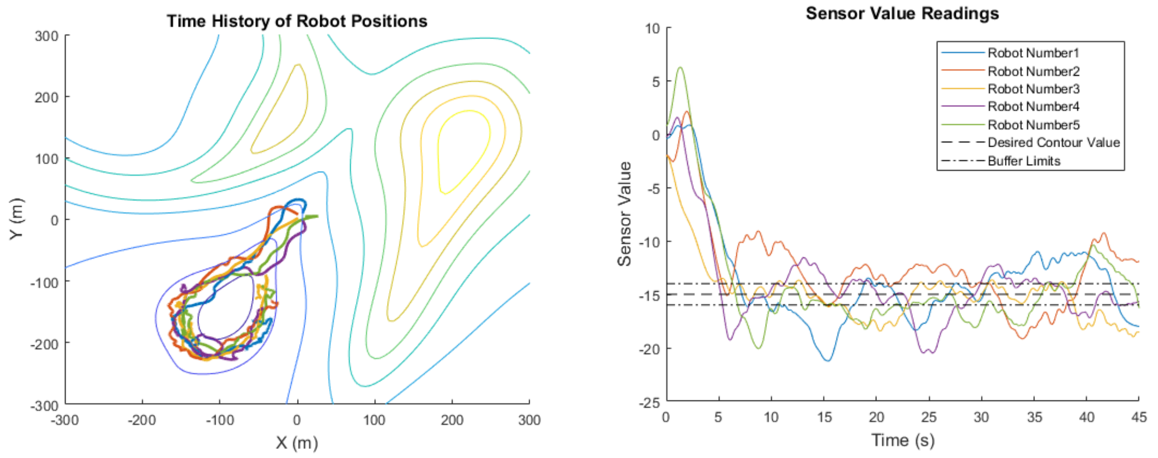


Figure 55. Contour Following for Obstacle Avoidance range of 20 m

Table 14. Quantitative Results for Contour Following with increasing Avoidance Range

Figure	Avoidance Range	Time to Contour(s)	% of time on Contour	Distance along Contour
54	2	8.82	75.2	1.31
A19	5	5.71	76.1	1.08
A20	10	4.81	69.4	0.92
A21	15	5.01	64.7	0.71
55	20	5.22	59.7	0.95

Increasing the obstacle avoidance range causes the swarm to follow the contour less precisely, but also increases the speed of response, as it allows the swarm to initially find the contour more quickly. The larger obstacle avoidance range causes the swarm to spread out more, which in turns yields gradient information over a larger area and can improve the quality of the swarm's gradient estimation. However, if obstacle avoidance range is larger than the distance over which the contour buffer occurs, as can happen in particularly steep gradients, the robots continually repel each other outside of the contour buffer range as seen in Figure 55. This indicates that it may be advantageous to have a varying obstacle avoidance size depending on whether the swarm is finding or following a contour. This will be further discussed in Chapter 5.

4.2.3 Effect of Contour Buffer Size on Contour Finding and Following

The contour buffer, as defined in 3.2.3, is the sensor value range for which the individual robots will be considered "on the contour." As such, a smaller contour buffer can be used for more precise following of the contour, while a larger contour buffer can be used if a rougher estimation is acceptable. Table 15 details the simulation parameters used to test the effect of the contour buffer range on the swarm performance.

Table 15. Simulation Parameters for Contour Following with varying contour buffer

# of Robots	Sim Time(s)	Sensor Range(m)	Avoid Range(m)	Des. Contour	Cont. Buffer	I.C.
5	45	150	2	40	*variable*	(5.5.5)

Figures 56 and 57 show the position and sensor value history for a contour buffer of 0.5 and 5, respectively. The time histories indicate that the lower contour buffer causes the swarm to more precisely follow the contour, but that the swarm does not progress as far along the contour. This is expected, as the smaller contour buffer causes the robots to re-enter contour finding rather than contour following, as their sensor value reading is outside the contour buffer.

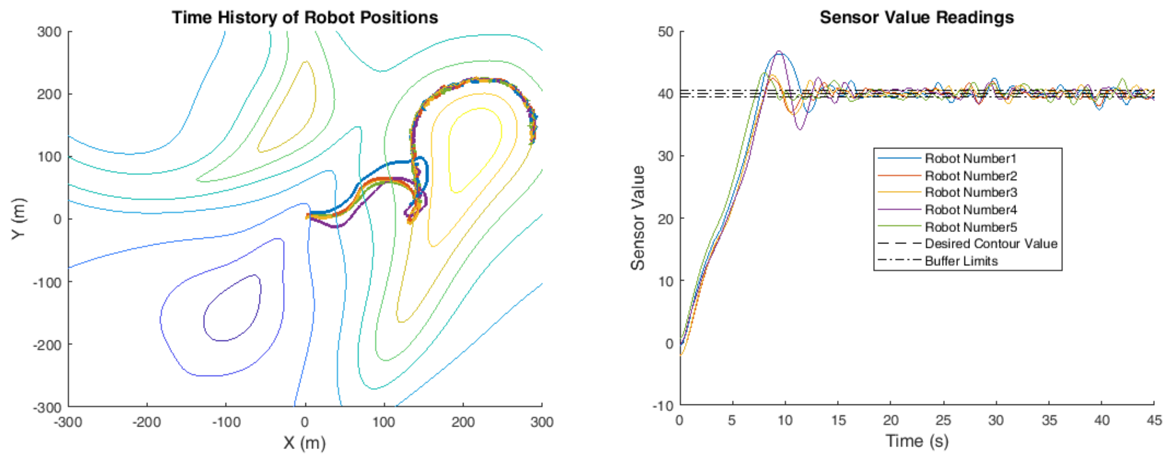


Figure 56. Contour Following for Contour Buffer of 0.5

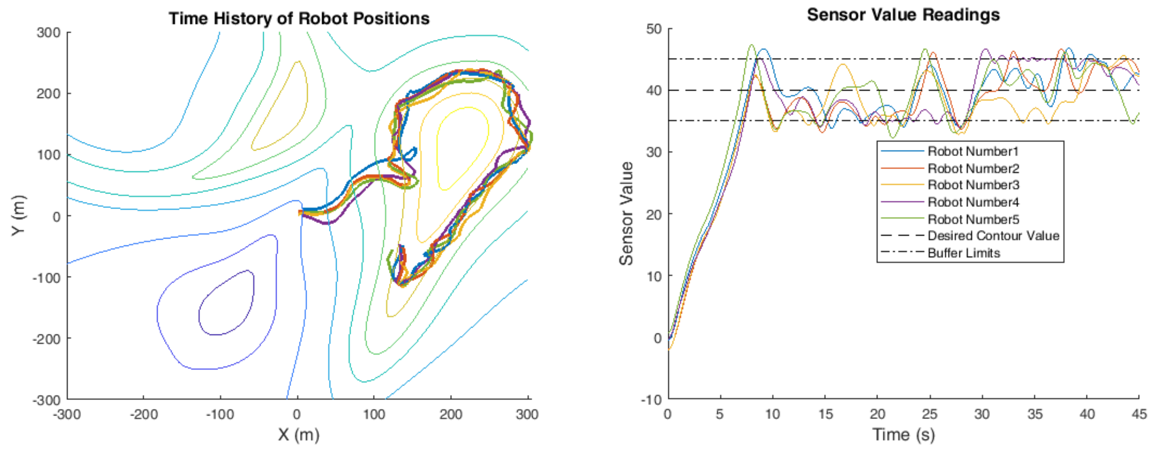


Figure 57. Contour Following for Contour Buffer of 5

The results in Table 16 indicate that decreasing the contour buffer range causes the robots to more closely follow the contour, while larger contour buffers allow for faster mapping and movement along the contour. A larger contour buffer also causes the percentage of time on the contour to be higher because there is more sensor value range that is considered “on contour.”

Figure 57 also shows a situation in which a lower contour buffer may be desirable. On contours that span ridges or trenches, there will be tighter “turns” that the robots must make to stay on the contour. The larger contour buffer used to produce Figure 57 causes the robots to “cut the corner” of this contour as it passes over the ridge. A higher contour buffer also causes the swarm to progress farther along the contour because there is more range for which robots are considered “on the contour” and moving along the contour rather than re-locating it using find max or find min.

Table 16. Quantitative Results for Contour Following with increasing Contour Buffer

Figure	Contour Buffer	Time to Contour(s)	% of time on Contour	Distance along Contour
56	0.5	7.82	84.2	0.52
A22	1	7.71	81.3	0.59
A23	2	7.62	79.7	0.68
A24	3.5	7.39	83.6	0.82
57	5	7.23	92.1	0.86

4.2.4 Effect of Robot Speed on Contour Finding and Following

As discussed in Chapter 2, robot control is achieved via specifying a bearing angle while maintaining a constant robot speed. Because each individual robot has its own dynamics, the robot does not immediately move in the direction of the commanded bearing angle, but has to dynamically respond to the command. This results in individual robot dynamics within the swarm. As is often the case with dynamic systems, simply going slower generally reduces overshoot to commands. In the context of contour finding and following, this overshoot is linked to the percentage of time that the swarm is within the contour buffer, and affects how well the robots can follow the contour. Simulations were run to show the effect of varying robot speed, with the simulation parameters used shown in Table 17.

Table 17. Simulation Parameters for Contour Following with varying robot speed

# of Robots	Sim Time(s)	Sensor Range(m)	Avoid Range(m)	Des. Contour	Cont. Buffer	I.C.	Robot Speed(m/s)
5	45	150	2	20	1	(5.5.5)	*variable*

Figures 58 and 59 show the response of the swarm when using a robot speed of 15 and 70, respectively. The position history shows that the higher robot speed allows the swarm to progress along more of the contour during the simulation, which is expected because the robots are moving faster. However, the tradeoff for this increased distance along the contour is more individual overshoots, and less percentage of time within the contour buffer.

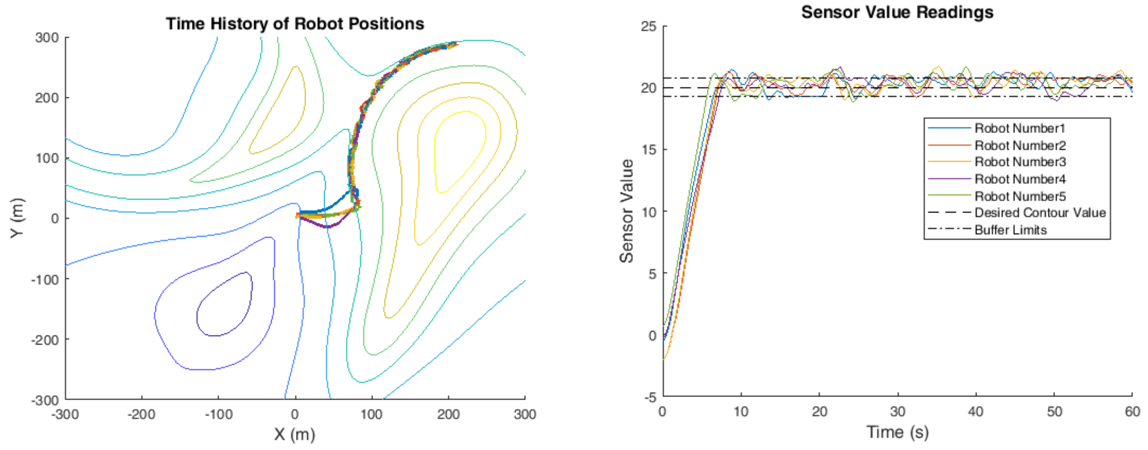


Figure 58. Contour Following for Robot Speed of 15

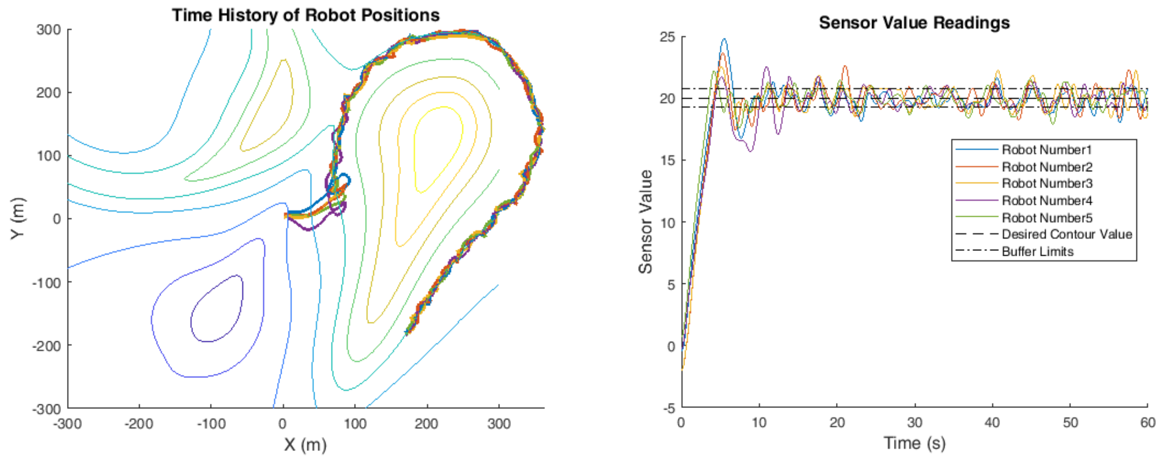


Figure 59. Contour Following for Robot Speed of 70

Table 18 summarizes the quantitative results for contour following with increasing robot speed, from robot speeds of 15 m/s to 70 m/s. The results in Table 18 confirm the expected results, namely that the time required to reach the contour is lowered with increased speed, and that the swarm is able to progress more quickly along the contour. The results in Table 18 indicate that these relationships are not proportional, which suggests a limited benefit to increasing the robot speed. Therefore the benefits of increasing robot speed need to be evaluated on a case-by-case basis. Generally, the increased progression along the contour is not significant enough to warrant using drastically higher robot speeds, especially considering that these higher speeds may require more robust and expensive motors on the physical swarm robots. Additionally, the swarm follows the contour less precisely at increased robot speeds, which is expected due to the individual robot dynamics.

Table 18. Quantitative Results for Contour Following with increasing Robot Speed

Figure	Robot Speed(m/s)	Time to Contour(s)	% of time on Contour	Distance along Contour
58	15	6.61	93.4	0.23
A25	30	4.6	84.2	0.48
A26	45	3.9	83.5	0.74
59	70	3.82	73.7	0.81

4.2.5 Repeatability of Contour Finding and Following for Varying Desired Contours

To validate the contour finding and following behavior, it is necessary to test the behavior for a variety of contours. To accomplish this, simulations were run for a variety of desired contours, using the simulation parameters in Table 19.

Table 19. Simulation Parameters for Contour Following with varying desired contour

# of Robots	Sim Time(s)	Sensor Range(m)	Avoid Range(m)	Des. Contour	Cont. Buffer	I.C.
5	45	150	2	*Variable*	1	(5.5.5)

The contour finding and following behavior was validated for a variety of contours ranging from -15 to 55, and successfully found and tracked the contours in all cases. The contour following for desired values of -15 and -5, shown in Figures 60 and 61, respectively, indicate that the swarm is able to travel down gradient and locate a contour and follow it. Figure 62 shows the swarm successfully following a desired contour of 55, indicating that the swarm can travel up the gradient to locate and follow the contour.

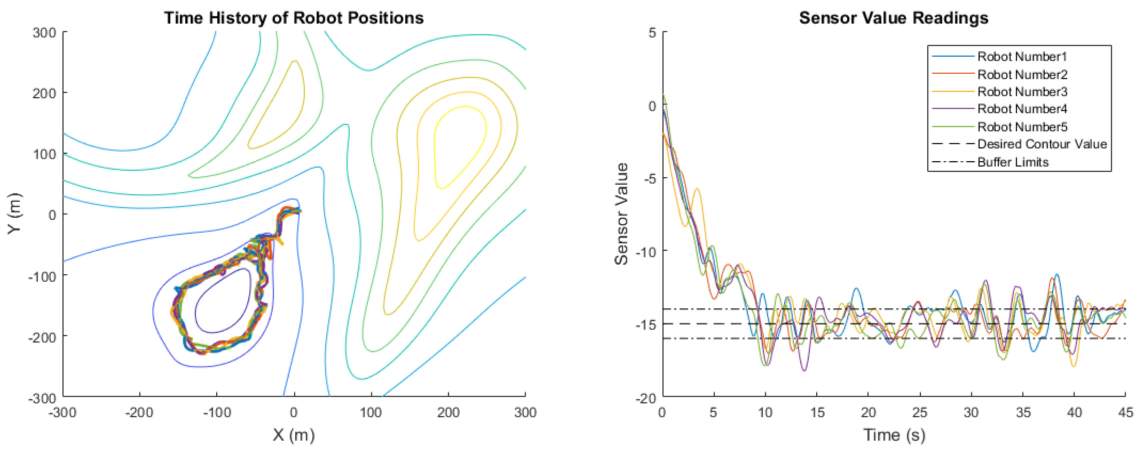


Figure 60. Contour Following for Desired Contour of -15

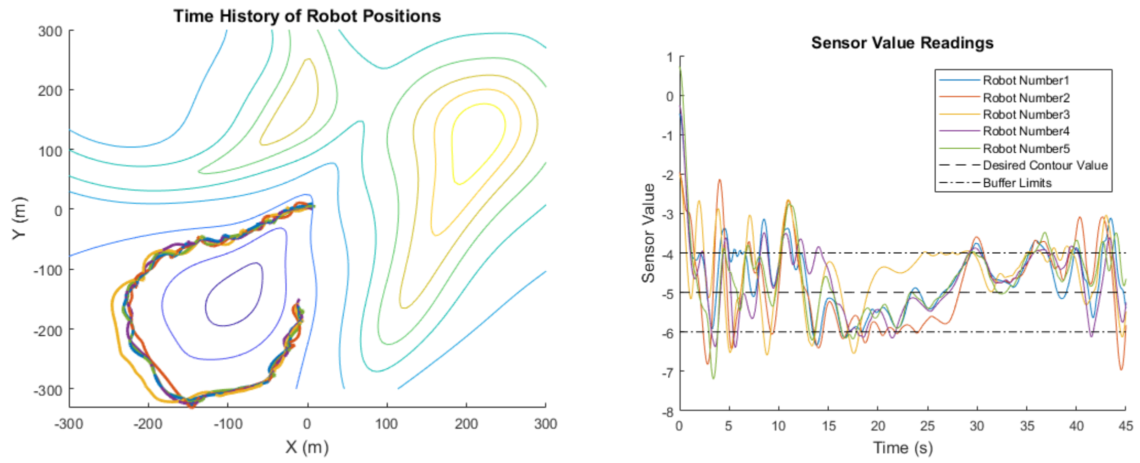


Figure 61. Contour Following for Desired Contour of -5

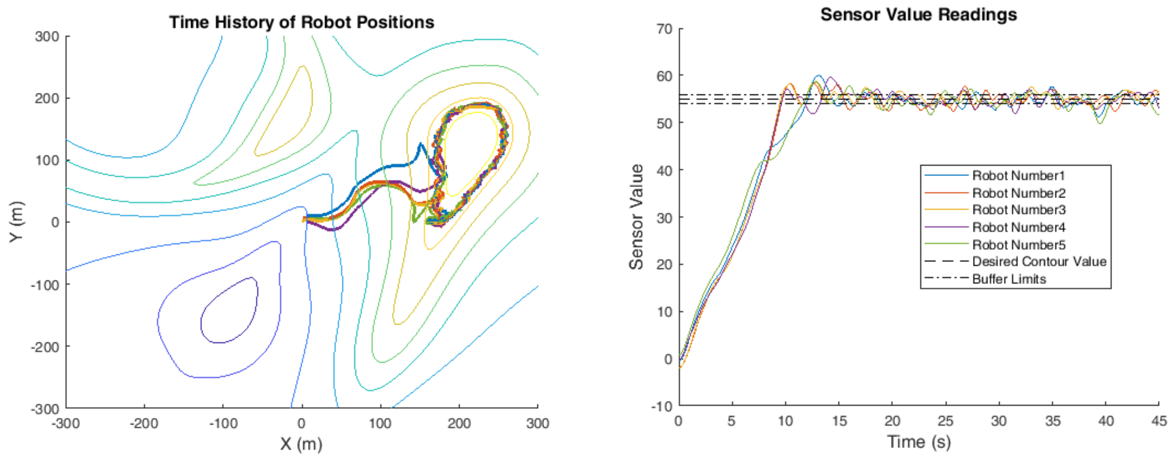


Figure 62. Contour Following for Desired Contour of 55

Figure 63 below shows the swarm following a contour of 40. It should be noted that the contour level of 40 is non-unique in this scalar field, as another contour of 40 exists in the local maxima in the upper left of the field. This indicates that while the swarm can successfully locate and follow a contour of desired value, it does not necessarily locate all contours of that desired value. This is similar to the issue of global or local extrema discussed in section 4.1.3.

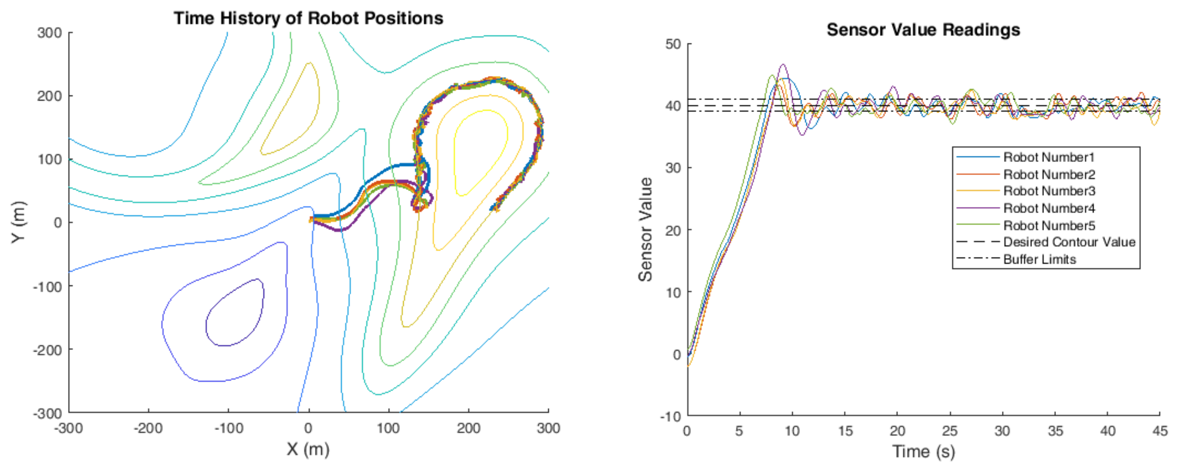


Figure 63. Contour Following for Desired Contour of 40

Finally, Figure 64 shows the swarm following a contour at a desired value of 20. The contour value of 20 is interesting because it takes the swarm to the saddle point at the top of the trench. For the given set of simulation parameters, the swarm continues over the saddle point and follows the contour on the other side of the saddle point. The sensor value history does not indicate large errors in sensor values that would indicate losing the contour, which means that the simulation parameters may need to be adjusted for this particular contour, as there is a very sharp turn in the contour. In particular, the contour buffer may need to be reduced for successful following of this specific contour. This will be revisited in section 4.3, which discusses design decisions for certain characteristics of the desired swarm behavior.

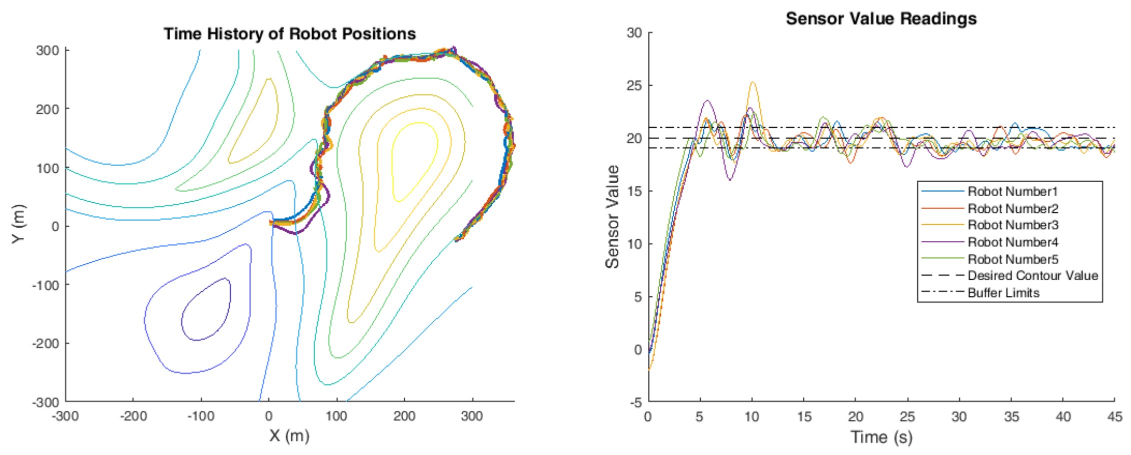


Figure 64. Contour Following for Desired Contour of 20

4.3 General Design Guidelines and Application to Testbed

One of the main goals of this research was to move toward "top-down" specification of swarm behaviors. Using the information presented in 4.1 and 4.2, the effect of different design parameters can be seen for adaptive navigation using swarm techniques. The effect of each design parameter is summarized in Table 20 below.

Table 20. Effect of parameters on Swarm Adaptive Navigation Performance

Parameter	Effect on Extrema Finding	Effect on Contour Finding and Following
Increasing Number of Robots	Yields a better gradient estimation, causing faster response but possibly higher steady-state error	
Decreasing Obstacle Avoidance	Lowers steady-state error but may decrease speed of response	Allows robots to more precisely follow the contour
Varying Initial Conditions	May affect whether robots locate local or global maxima	May affect which contour is tracked if there are multiple contours of the desired value in the field.
Decreasing Contour Buffer	N/A	Allows robots to more precisely follow the contour, but slows the speed of procession around the contour.
Decreasing Robot Speed	Slows speed of response but reduces system "overshoot."	Slows speed of response but allows swarm to track contour more precisely, and spend less time re-locating the contour.

Using the design guidelines outlined in Table 20, the indoor Decabot testbed system was used to validate the swarm behaviors on a basic paraboloid scalar field with a single minimum. Additionally, the scalar field presented in Figure 40 was printed in grayscale to facilitate testing the finding of local maxima, global maximum, the global minimum, and to follow a contour that requires the swarm to reverse the sense of commanded bearings as it passes through a trench.

The first test run using the more complicated scalar field was to find a local maxima. The swarm was started in the upper left quadrant of the field as shown in Figure 65. The swarm successfully navigated to the local maximum location, but, as expected, did not find the global maximum as indicated in the sensor value history of Figure 65. The results achieved in the testbed environment closely match those for local maximum finding shown in simulation in Figure 49. Using the design guidelines above, robot speed and obstacle avoidance range were set at low values for extrema finding applications. The experimental parameters are outlined in Table 21.

Table 21. Experimental Parameters for Extrema Finding using Decabot Indoor Testbed System.

Number of Robots	Run Time(s)	Avoidance Range	Robot Speed	Initial Conditions
3	25	0.4 m	0.2 m/s	User-placed

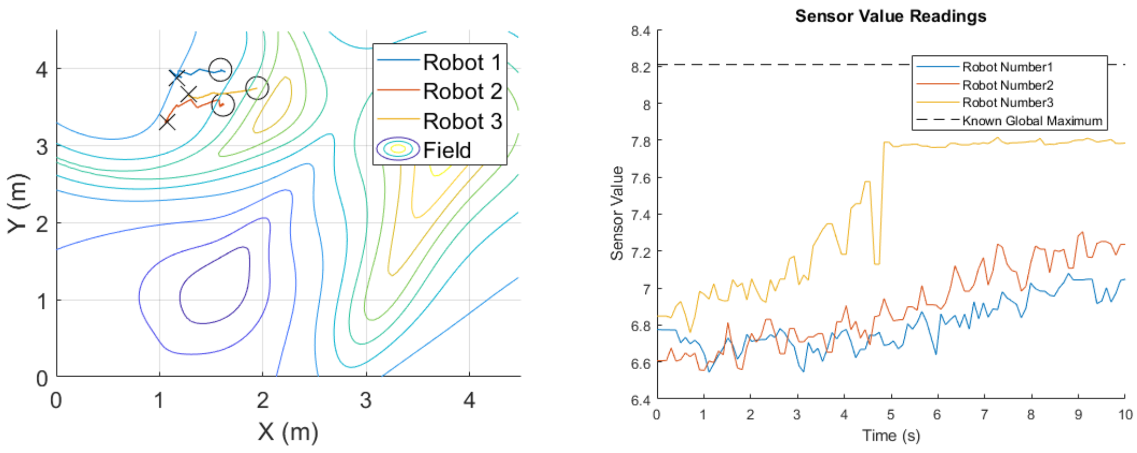


Figure 65. Local Maxima Finding using Decabot Indoor Testbed System

The swarm was subsequently positioned in the right half of the field to allow it to reach the global maximum. The swarm successfully travelled up-gradient to reach the location of the global maximum, as shown in Figure 66.

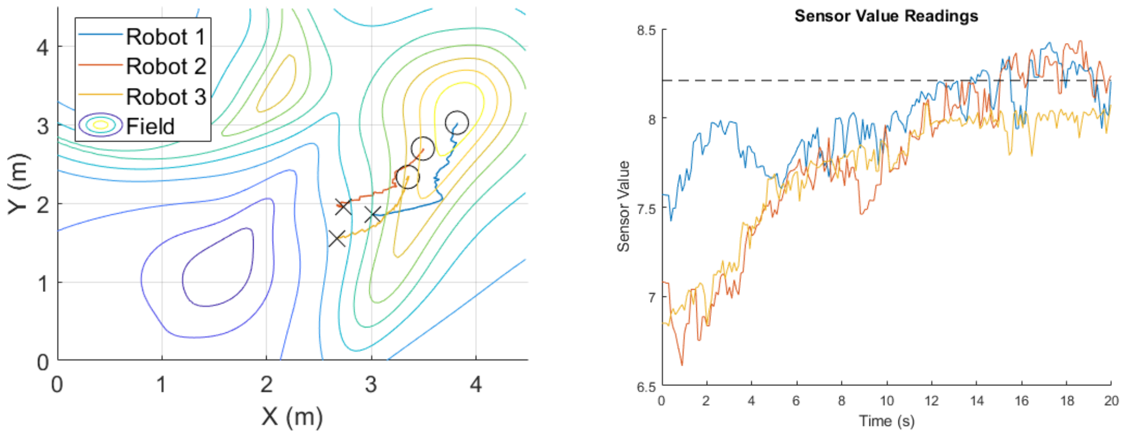


Figure 66. Global Maximum Finding using Decabot Indoor Testbed System

The sensor value readings in Figure 66 show readings that are above the "known global max" value. This is due to sensor noise as well as inconsistencies when calibrating the simulation scalar field values, which range from -25 to 70, to the Decabot's sensor range, which is ~5.5-8.3. In particular, the grayscale sensors on the Decabots vary slightly from robot to robot, causing imprecise calibration. The system response shown in the position history of the swarm indicates that the robots successfully move up gradient to the location of the global maximum.

Figure 67 depicts the swarm in a minimum-finding behavior, starting from a position near the global maximum. The robots move down gradient until they reach a trench, and then continue to move down gradient to follow the trench down to the minimum. Figure 68 shows the swarm starting from a different location and moving down gradient to the minimum.

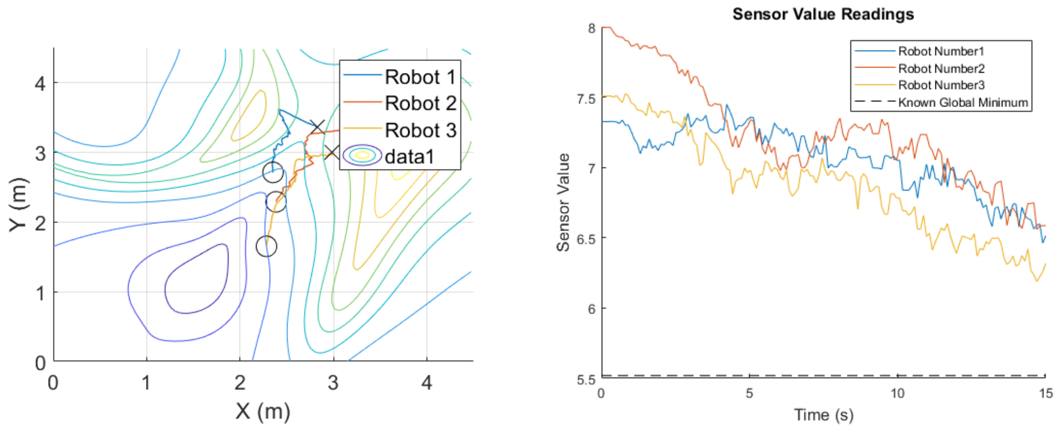


Figure 67. Robot Global Minimum Finding using Decabot Indoor Testbed System

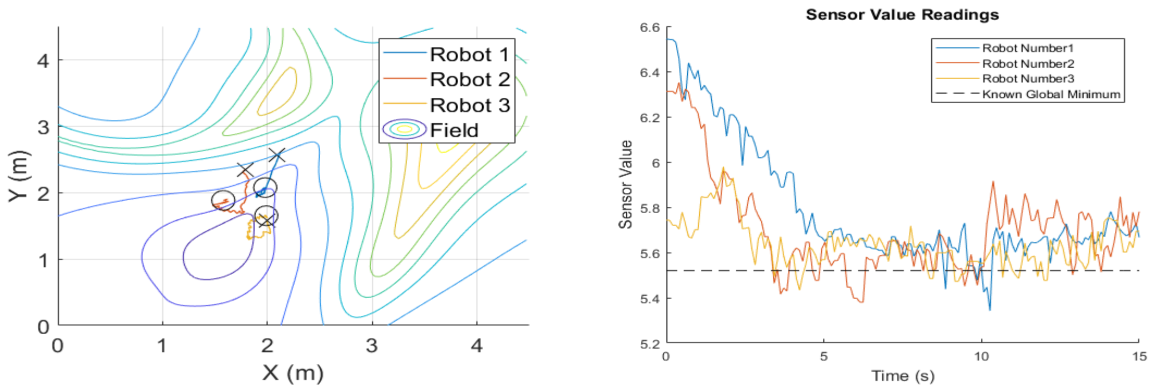


Figure 68. Part 2 of Global Minimum Finding using Decabot Indoor Testbed System.

In addition to extrema finding, contour finding and following was run on the indoor testbed system. Because the scalar field used exhibits fairly steep gradients, a low obstacle avoidance, robot speed, and contour buffer were used. The contour buffer had to still be sufficiently large to allow for noise in the sensor value signal, and as such was selected to be at 0.2. The full experimental parameters used are outlined in Table 22 below. The experimental results from this run are shown in Figure 69.

Table 22. Experimental Parameters for Contour Finding and Following using Decabot Testbed System.

Number of Robots	Run Time(s)	Avoidance Range(m)	Desired Contour	Contour Buffer	Robot Speed
3	120	0.4	6.5	0.2	0.2

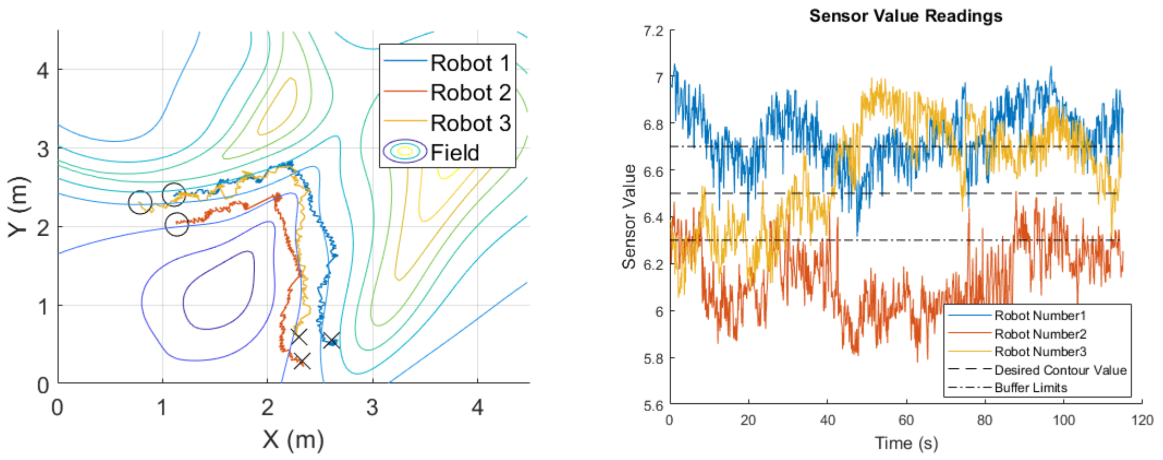


Figure 69. First run of Contour Following using Testbed

Figure 69 indicates the swarm's ability to successfully follow a contour despite the contour's sharp "corner" located around (2.2, 2.2). However, observing the sensor value readings shows that the swarm does not follow the contour very precisely, and also does not progress very far along the contour given the length of the experimental run. As a result, the obstacle avoidance range and robot speed were decreased to allow more precise tracking of the contour, as shown in Table 23 below. The results of the experiment using the parameters in Table 23 are shown in Figure 70.

Table 23. Revised Experimental Parameters for Contour Finding and Following, Decabot System.

Number of Robots	Run Time (s)	Avoidance Range (m)	Desired Contour	Contour Buffer	Robot Speed
3	90	0.3	6.5	0.2	0.15

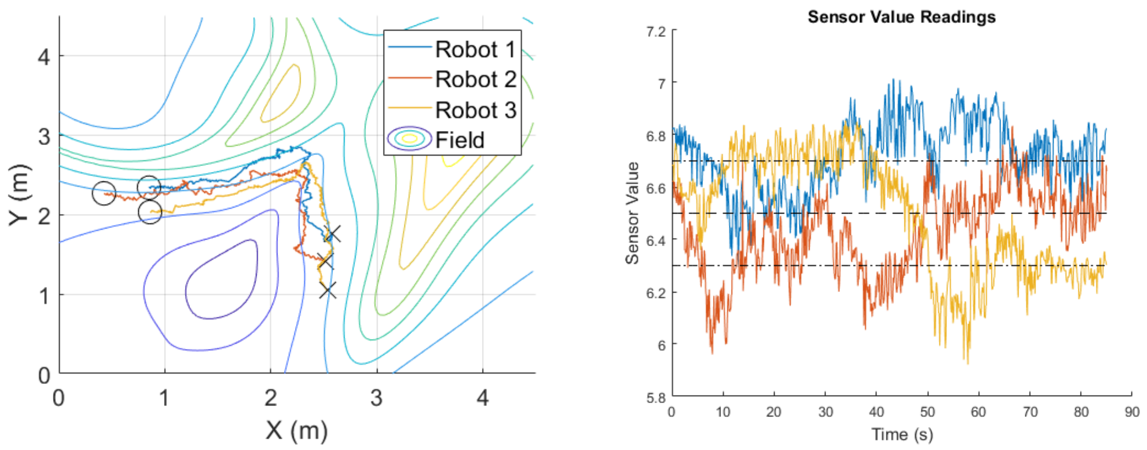


Figure 70. Contour Following with Revised Parameters

While it should be noted that the initial conditions for Figure 70 and Figure 69 are not identical, it is clear that even with a reduced robot speed the swarm is able to progress further along the contour than in Figure 69, due the fact that the swarm more closely follows the contour and therefore spends less time in the contour finding modes. This is supported by the fact the sensor values are consistently closer to the contour buffer, and the fact that the swarm does not “cut the corner” of the contour as Robot 2 does in Figure 69.

4.4 Comparison to Cluster-Based techniques

As mentioned in Chapter 1, one of the main goals of this research was to re-produce the adaptive navigation behaviors presented in [34] while utilizing a swarm-based architecture rather than a cluster-based control architecture. This section will compare the results of the swarm-based adaptive navigation primitives to the results of the cluster-based adaptive navigation primitives, discuss issues that are common across both techniques, and finally discuss the advantages and disadvantages of the respective techniques.

4.4.1 Comparison of Functionality

[34] presents adaptive navigation primitives for extrema finding, contour finding and following, and ridge and trench following. This research has successfully recreated the extrema finding behaviors presented in [34]. Figure 71 indicates the cluster performance for ascending to a local maximum (path B), a global maximum (path A), and the global minimum (path C). The right image in Figure 71 shows the same behaviors repeated using a swarm architecture and starting from similar initial conditions. The swarm-based portion of Figure 71 was produced using a 3-robot swarm, as extrema finding in [34] is accomplished using a 3-robot cluster.

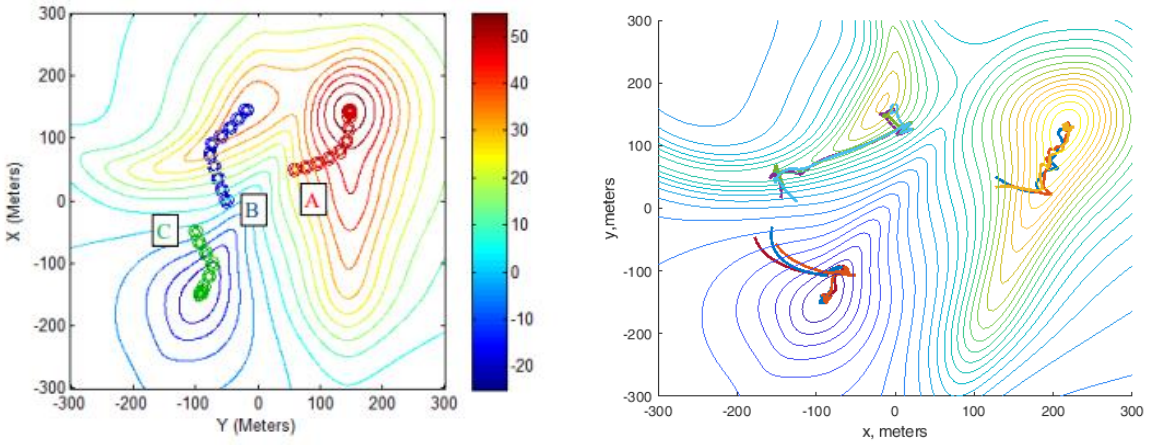


Figure 71. Cluster-based (left) and swarm based (right) extrema finding

For the same scalar field, the swarm architecture successfully navigates to the expected extrema, and follows roughly the same paths demonstrated by paths A, B, and C. One major difference between the two system responses is the "smoothness" of the system response. This can be seen in the swarm-based global maximum finding, as the swarm oscillates back and forth across the ridge as it gradually works its way up to the global maximum, rather than going directly up as in the cluster-based technique. One major reason for this is that the swarm-based architecture uses an "implicit" gradient estimation that is a combination of local slopes to other robots, while the cluster-based architecture employs an explicit feature estimation that utilizes that known relative positions of other robots within the cluster and their sensor value readings. Another reason for these oscillations is the formation control technique. The low degree of freedom formation controller used in the swarm causes inter-robot oscillations, whereas the cluster formation controller maintains fixed relative positions between robots.

In addition to extrema finding, the contour following results from [34] were successfully reproduced. Figure 72 shows the cluster-based and swarm-based contour following for multiple contours, with movement both up and down gradient to find the contours and counter-clockwise and clockwise travel of the contours. In the cluster-based adaptive navigation, path A is up-gradient and counter clockwise, path B is down gradient and counter-clockwise, and path C is up gradient and clockwise.

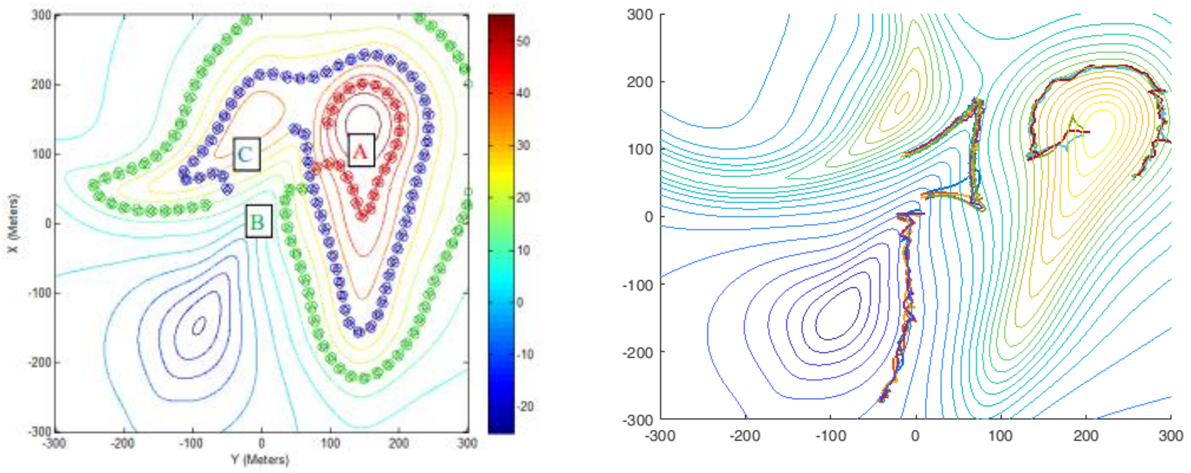


Figure 72. Cluster-based Contour Following (Left) and Swarm-based Contour Following (Right)

In the swarm-based contour following of Figure 72, the top right path moves down gradient and begins clockwise contour following, the middle path moves up gradient and begins clockwise contour following, and the bottom path starts on the correct contour and remains on it by traveling counterclockwise. In similar fashion to the extrema finding, the swarm-based contour following exhibits the same capabilities as the cluster-based primitive but with less smooth tracking, as the swarm will tend to oscillate about the desired contour while progressing along it.

The final capabilities of ridge and trench following have not yet been successfully implemented, and are discussed further in Chapter 5.

4.4.2 Comparison of Design Considerations

The two main design considerations that are discussed in [34] are the size of the cluster and the speed of the individual robots. The size of the cluster refers to the spacing of the robots within in the cluster, and not the specific number of robots within the cluster. As such, these design considerations can be compared to the swarm-based design considerations of obstacle avoidance and individual robot speed. [34] draws similar conclusions to those reached in 4.3, and notes that the size of the cluster must be sufficiently large to yield an accurate estimation, but that a cluster that is too large may fail to sense rapid changes in the scalar field. Additionally, [34] notes that robot speed is an important design consideration, especially when the feature to be mapped requires rapid changes in robot bearing, such as the contours in 4.4.1.

4.4.3 Disadvantages & Advantages of Swarm-Based Adaptive Navigation

The comparison of functionalities in section 4.4.1 highlights some of the main disadvantages of swarm-based adaptive navigation, namely oscillations and reduced resolution of results. The oscillations in extrema finding and contour finding and following do not dramatically influence the system performance, as the swarm-based techniques were able to replicate the results of the cluster-based techniques. However, the swarm oscillations may cause the swarm-based technique to take longer to locate the extrema or map the contour. In real-world applications such as disaster relief, where time is a crucial factor, this may make the swarm-based adaptive navigation techniques less desirable. Another disadvantage of the swarm-based technique is that it offers only implicit formation control and not explicit formation control. This is a disadvantage for more complicated scalar field features, such as ridges, trenches, and fronts, which require precise control of the location of robots in relation to each other to assure that the ridge, trench, or front is maintained.

Despite these shortcomings, the swarm-based control architecture has some desirable advantages for adaptive navigation, most importantly:

- *Decentralized Architecture:* In contrast to the centralized architectures, where a centralized controller distributes control bearings to each robot within the MMS, each individual robot in the swarm calculates its own command bearing based off local communications. This offers advantages in situations where global communication may be limited or unavailable.

- *Robustness to Individual Robot Failures:* 4.1 and 4.2 showed the increasing capability of the swarm with increasing number of robots in the swarm, but also indicated that the swarm can successfully complete adaptive navigation primitives for lower numbers of robots. This indicates robustness to individual robot failures within the swarm without altering control laws. In contrast, the cluster-based architecture utilizing 4 robots would require a different cluster definition for 3 robots if an individual robot fails.
- *Easily Scalable:* As was shown in 4.1 and 4.2, the swarm behaviors can be scaled easily without dramatically increasing required calculations or requiring new formation control definitions. The cluster-based technique is not as easily scaled, as adding robots to the cluster results in rapidly increasing numbers of cluster shape variables, and therefore of control laws and calculations. This relates back to the fact that the cluster is a full degree-of-freedom controller, while the implicit formation control for the swarm is a much lower degree of freedom controller.
- *Adaptable Size:* 4.4.2 discussed the fact the cluster and swarm size is a large consideration in adaptive navigation techniques. In the cluster formation, the "size" of the cluster may actually be dependent on a variety of cluster variables, but the implicit formation control of the swarm is dependent on obstacle avoidance range and attract gain only. This simplified structure, while not offering the refined formation control of the cluster technique, offers easy variation of the size of the swarm for different fields. This will be further discussed in Chapter 5.

These advantages make the swarm-based adaptive navigation techniques desirable for applications that may require decentralized control due to communication limitations. The swarm-based techniques also reduce the impact of increasing numbers of individual agents on the required number of calculations, as each agent calculates its own commanded bearing, rather than relying on a centralized controller. For example, a larger swarm might consist of 50 robots, but each individual robot would only communicate with its nearest neighbors. Therefore the computational load for each robot would be equivalent of that of a much smaller swarm. While cluster control offers more refined control through a full degree-of-freedom controller, the swarm-based adaptive navigation techniques presented in Chapters 3 and 4 offer adaptive navigation capabilities with a low degree of freedom controller at a reduced computational cost. Therefore these techniques may be useful when computation cost is a large concern, or full degree of freedom control is not required.

5. Conclusions and Future Work

This paper has presented a new approach to adaptive navigation primitives that reduces calculation load without dramatically sacrificing system performance. A simulation environment was developed that easily interfaces with the existing indoor testbed, allowing for rapid vetting and validation of swarm primitive and composite behaviors. These swarm composite behaviors were combined to yield Adaptive Navigation primitives such as extrema finding and contour finding and following. In simulation and in experimentation, the effect of different design parameters were shown, which yielded design guidelines to facilitate top-down specification of swarm behaviors. Finally, the swarm-based adaptive navigation behaviors were compared to existing cluster-based adaptive navigation techniques to show comparable system performance, and to compare advantages and disadvantages of each technique. Future work in this area includes the following:

1. *Development of Additional Adaptive Navigation Primitives:* As mentioned in Chapter 4, it is desired to develop more adaptive navigation primitives to allow for ridge and trench following, saddle point station keeping, and front detection and following.
2. *Adaptive Sizing for Increased Performance:* Currently, certain parameters such as avoidance range and contour buffer range are set as constants by the designer. Future development could yield adaptive sizing of the swarm that determines appropriate avoidance ranges and contour buffer ranges based off local gradient estimation and scalar field measurements.
3. *Numerical Validation:* Chapter 4 began to investigate the impact of different design parameters on swarm performance, but a more thorough validation could be performed using Monte Carlo simulations. Such validation could yield more precise information about system convergence to desired behavior for variations in design parameters.
4. *Adaptive Navigation Composites:* The extrema finding and contour following behaviors are presented as adaptive navigation primitives, with the understanding that these could be combined with other behaviors for adaptive navigation composite behaviors. Examples include maintaining a certain wireless connection (contour following) while travelling across campus (Go-To), or mapping all maxima in a region (utilization of extrema finding).
5. *Increased Fidelity of Simulation:* Running the vetted behaviors on the testbed revealed a few possible additions to the simulation that would increase the fidelity of the simulation. Possible additions include a white noise addition to the sensor signals, lag (or a "transport delay" in Simulink terminology), and randomized robot failures within simulation.
6. *Expanded Performance Metrics:* Expanding performance metrics to compare the swarm behavior to known gradient information may help further evaluate the swarm's behavior. For example, [34] presents a cluster-based metric that evaluates the cluster's bearing in comparison to the gradient bearing for contour following. Additional metrics could also include expected computational load and how these compare to cluster-based techniques.

REFERENCES

1. Craig, J. *Introduction to Robotics: Mechanics and Control.* 4th Edition. Pearson, 2018.
2. Franka Emika. "The Franka Emika Panda Robot." Franka Emika Site, 23 April 2019, https://www.franka.de/panda.
3. Kitov Systems. "The Kitov Automated Inspection System." Kitov System Site, 23 April 2019, https://www.kitovsystems.com/.
4. Jazdi, N. *Cyber physical systems in the context of Industry 4.0.* 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, Cluj-Napcoa, 2014, pp 1-4.
5. Wise, M., Ferguson, M., King, D., Diehr, E., and Dymesich, D. *Fetch and Freight: Standard Platforms for Service Robot Applications.* Fetch website, 23 April 2019, https://fetchrobotics.com/robotics-platforms/fetch-mobile-manipulator/.
6. NASA JPL. "The Mars Rover – Exploration" NASA Site, 23 April 2019, https://www.jpl.nasa.gov/missions/mars-exploration-rover-opportunity-mer/.
7. Harvard Robotics. "The Harvard Kilobot Swarm", Harvard website, 23 April 2019, https://wyss.harvard.edu/media-post/kilobots-a-thousand-robot-swarm/.
8. Tomer, S., Kitts, C., et al. *A low-cost Indoor Testbed for Multirobot Adaptive Navigation Research.* 2018 IEEE Aerospace Conference, pp 1-12.
9. Bazoula, A., Djouadi, M.S., and Maaref, H. *Formation Control of Multi-Robots via Fuzzy Logic Technique.* International Journal of Computers, Communication, and Control. May 2008.
10. Kitts, C. and Mas, I. *Cluster Space Specification and Control of Mobile Multirobot Systems.* 2009 IEEE/ASME Transactions on Mechatronics, Vol 14, Issue 2. April 2009.
11. Galvez, R., Faelden, G., Maningo, J., and Nakano, R. *Obstacle Avoidance Algorithm for Swarm of Quadrotor Unmanned Aerial Vehicle using Artificial Potential Fields.* 2017 IEEE Region 10 Conference, November 2017.
12. Safadi, H. *Local Path Planning using Virtual Potential Field.* COMP 765 Lecture Notes, April 2007.
13. Senanyake, M., Barca, J., Senthooran, I., and Chung, H. *Search and Tracking Algorithms for Swarms of Robots: A Survey.* Robotics and Autonomous Systems 75, Part B. pp 422-434, January 2016.
14. Mohan, Y. and Ponnambalam, S.G. *An extensive Review of Research in Swarm Robotics.* In *IEEE Conference on Nature and Biologically Inspired Computing.* January 2010.
15. Boston Dynamics. "The Atlas Robot" Boston Dynamics website, 23 April 2019. https://www.bostondynamics.com/atlas.
16. Festo Robotics. "Festo bio-inspired actuation systems." Festo website. 23 April 2019. https://www.festo.com/group/en/cms/10157.htm ,
17. Sharkey, J.C.A. *Robots, Insects and Swarm Intelligence*. *Artificial Intelligence Review*, vol. 26, pp. 255-268. 2006.

18. Labella T.H. , Dorigo M. and Deneubourg, J.L. *Division of Labor in a Group of Robots Inspired by Ants' Foraging Behavior.* in *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, no. 1, September 2006, pp. 4-25.

19. Tomlinson B. and Blumberg, B. *Using Emotional Memories to Form Synthetic Social Relationships,* Proceedings of AAMAS 02, Bologna, Italy, July 15, 2002.

20. Payton, D., Daily, M., Hoff, B., Howard, M. and Lee, C. *Pheromone Robotics.* in *Autonomous Robots*, vol.11, no. 3, pp 319-324, 2001.

21. Parker, L. *Multiple Mobile Robot Systems.* http://faculty.engineering.asu.edu/acs/wp-content/uploads/2016/08/Multiple-Mobile-Robot-Systems-2008.pdf .

22. Pugh, J. and Martinoli, A. "Multi-robot Learning with Particle Swarm Optimization", AAMAS, Hakodate, Hokkaido, Japan, May 8-12, 2006, pp. 441-448.

23. Sedighizadeh, D. and Mazaheripour, H. *Optimization of multi-objective vehicle routing problem using a new hybrid algorithm based on particle swarm optimization.* Alexandria Engineering Journal, Vol 57, Issue 4. December 2018.

24. Quinlan, J. R., *Induction of Decision Trees.* Machine Learning, Boston: Kluwer Academic Publishers, 1986, vol. 1, pp. 81-106.

25. Pomerleau, D. *Neural Network Based Autonomous Navigation.* In C. Thorpe, Vision and Navigation: The CMU Navlab. Kluwer Academic Publishers, 1990.

26. Khashayar R. , Baghaei, K.P. and Agah A. *Task Allocation Methodologies for Multi-Robot Systems*, Technical Report Information and Telecommunication Technology Center, The University of Kansas, Lawrence, Kansas 66045, November 2002.

27. Dias, M.B. , Zlot, R.M. , Kalra N. , and Stentz A. "Market-Based Multirobot Coordination: A Survey and Analysis," Tech. report CMU-RI-TR-05-13, Robotics Institute, Carnegie Mellon University, 2005.

28. Bertsekas, D.P. "Auction Algorithms for Network Flow Problems: A Tutorial Introduction", Computational Optimization and Applications, 1992, pp. 7-66.

29. Oritz, C., Vincent, R. and Morriset, B. "Task Inference and Distributed Task Management in the Centibots Robotic System," Proceedings of the 4th Joint International Conference on Autonomous Agents and Multi-Agent Systems, Utrecht, The Netherlands, 2005, pp. 870-877.

30. Parker, L.E. "ALLIANCE: An Architecture for Fault Tolerant MultiRobot Cooperation", IEEE Transactions on Robotics and Automation, vol. 14, no. 2, 1998, pp. 220-240.

31. Gross, R., Mondada, F. and Dorigo, M. "Transport of an Object by Six Pre-attached Robots Interacting via Physical Links", Proceedings 2006 IEEE International Conference on Robotics and Automation, 15-19 May 2006, pp. 1317 – 1323.

32. Miyata N., Ota J., Aiyama, Y., Sasaki J. and Arai T. *Cooperative Transport System with Regrasping Car-like Mobile Robots*, Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. Sept. 1997.

33. Wang, Z.D., Hirata, Y. and Kosuge. K. *Control of a Rigid Caging Formation for Cooperative Object Transportation by Multiple Mobile Robots.* in *Proceedings of the 2004 IEEE International Conference on Robotics & Automation,* New Orleans, LA. April, 2004. pp. 1580-1585.

34. Kitts, C., McDonald, R., and Neumann, M. *Adaptive Navigation Control Primitives for Multirobot Clusters: Extrema Finding, Contour Following, Ridge/Trench Following, and Saddle Point Station Keeping.* 2018 IEEE Access. Volume 6, pp 17625-17642.

35. Tomer, S., Kitts, C., Neumann, M. , McDonald, R., Bertram, S., Cooper, R. , Demeter, M. , Guthrie, E. , Head, E. , Kashikar, A. , Kitts, J. , London, M. , Mahacek, A. , Rasay, R. , Rajabhandharaks, D. , and Yeh, T. *A low-cost Indoor Testbed for Multirobot Adaptive Navigation Research.* 2018 IEEE Aerospace Conference, pp 1-12.

36. Eu, K.S., and Yap, K.M. *Chemical plume tracing: A three-dimensional technique for quadrotors by considering the altitude control of the robot in the casting stage.* February 2018.

37. Cheng, Z. and Wang, G. *Real-time RGB-D SLAM with Points and Lines.* 2018 2nd IEEE Advanced Automation, Management, Communicates, Electronic and Automation Control Conference.

38. Li, W., Farrell, J.A., Pang, S. and Arrieta, R.M. *Moth-inspired chemical plume tracing on an autonomous underwater vehicle.* 2006 IEEE Transactions on Robotics, Vol 22, No 2, pp 292-307.

39. Kang, X. and Li, W. *Moth-inspired plume tracing via multiple autonomous vehicles under formation control.* Journal on Adaptive Behavior. Vol 20, no 2, pp 131-142, Apr 2012.

40. Marjovi, A. and Marques, L. *Swarm Robotic Plume Tracking for Intermittent and Time-Variant Odor Dispersion.* Institute of Systems and Robotics, University of Comibra, Portugal.

41. Khan, A., Mishra, V., and Zhang, F. *Bio Inspired Source Seeking: A Hybrid Speeding up and Slowing Down Algorithm.* 2016 IEEE 55th Conference on Decision and Control.

42. Wu, W. and Zhang, F. *A gradient-free 3-dimensional Source Seeking Strategy with Robustness Analysis.* 2018 IEEE Transactions on Automatic Control.

43. Pugh, J and Martinoli, A. *Inspiring and Modeling Multi Robot Search with Particle Swarm Optimization.* Proceedings of 2007 IEEE Swarm Intelligence Symposium. Pp 332-339.

44. Gainer J, et al. *Persistent Target Detection and Tracking by an Autonomous Swarm.* ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.

45. Koohifar, F., Guvenc, I., and Sichitiu, M.L. *Autonomous Tracking of Intermittent RF Source using a UAV Swarm.* IEEE Access, Special Section on Networks of Unmanned Aerial Vehicles. April 2018.

46. Fabbiano, R. and Garin, F. *Source Localization by Gradient Estimation Based Poisson Integral.* Elsevier automatica. 2014.

47. Xue, S. and Zeng, J. *Target Search using Swarm Robots with Kinematic Constraints.* 2009 Chinese Control and Decision Conference. pp 1-8.

48. Khatib, O. *Real-time Obstacle Avoidance for Manipulators and Mobile Robots.* In Proceedings of the IEEE Conference on Robotics and Automation, pages 500-505, 1985.

49. Kitts, Christopher, Hart, Shae, Reese, Maximilian, and Metzger, Nathan. *Robotics Simulator for Top-Down Development and Verification of Swarm Behaviors.* Submitted to Proceedings of ASME 2019 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.

Appendix A

Table A1. Simulation Parameters for Produced Figures in Chapter 1

Figure #	**Behaviors**	***N***	**T_{sim}**	**SV_{rnge}**	**Avoid**	**SV_{des}**	**SV_{buff}**	**I.C.**	**Speed**	**Field**
32	FMin	3	50	150	1	--	--	Rand	6	Sink
33	FMax	3	50	150	1	--	--	Rand	6	Source
36	FContour	3	40	150	1	5.5	0.2	Rand	5	Testbed

The figures referenced in Chapter 4 are included below:

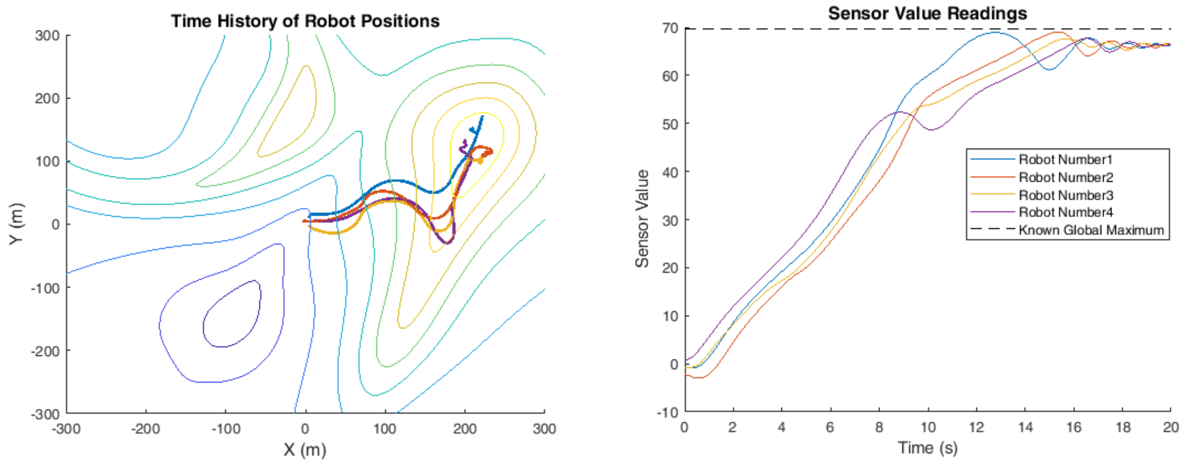


Figure A1. Max Finding, *N=4* Robots

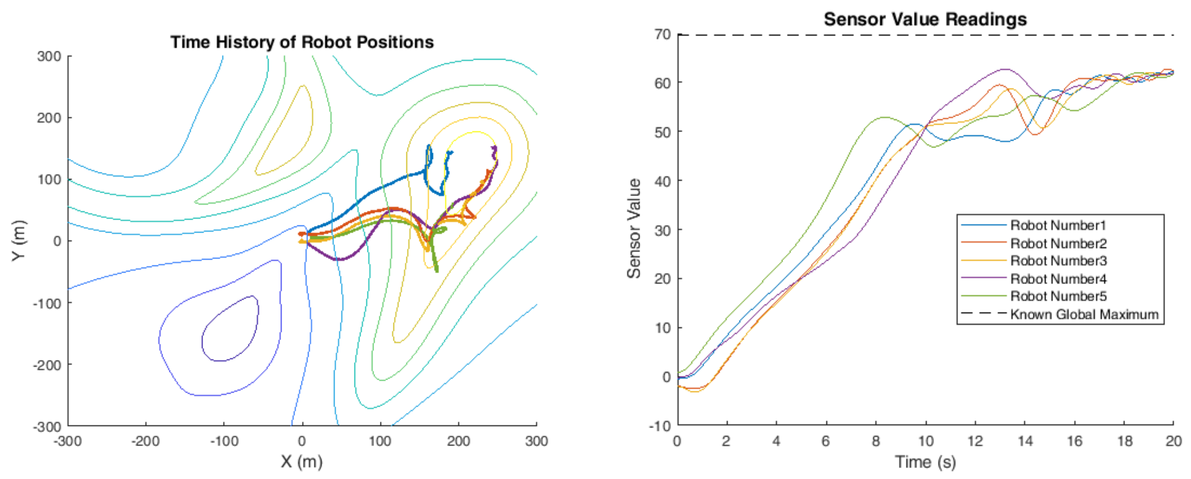


Figure A2. Max Finding, *N=5* Robots

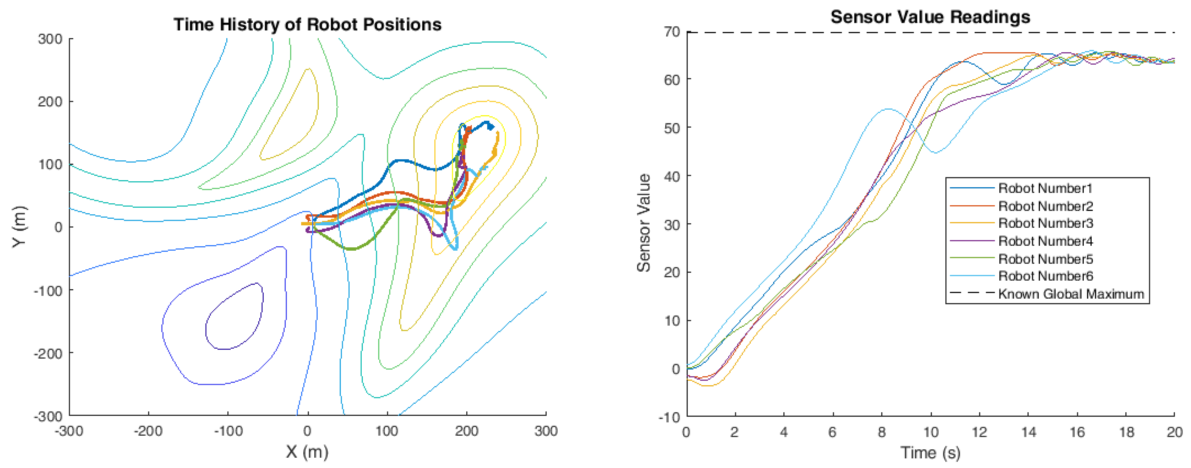


Figure A3. Max Finding, *N=6* Robots

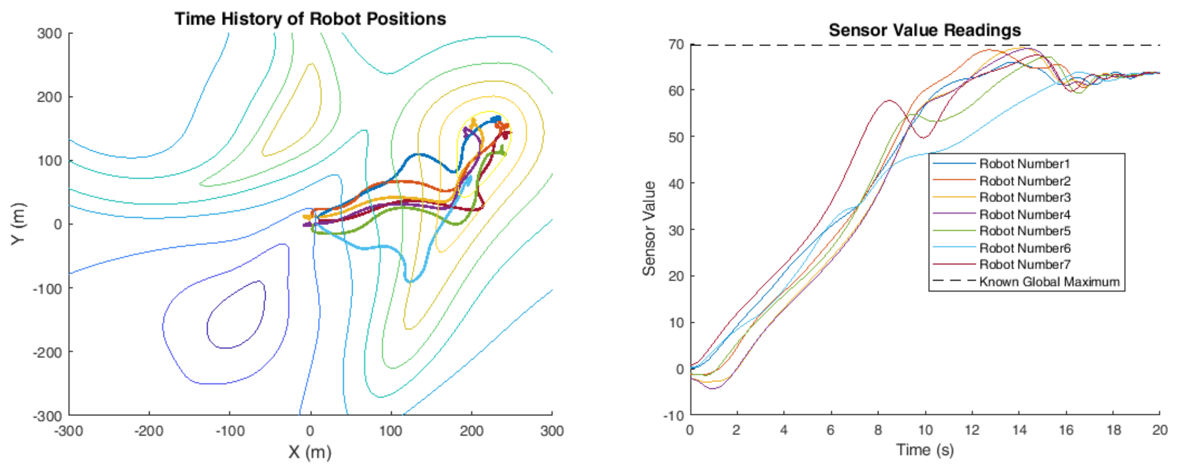


Figure A4. Max Finding, *N=7* Robots

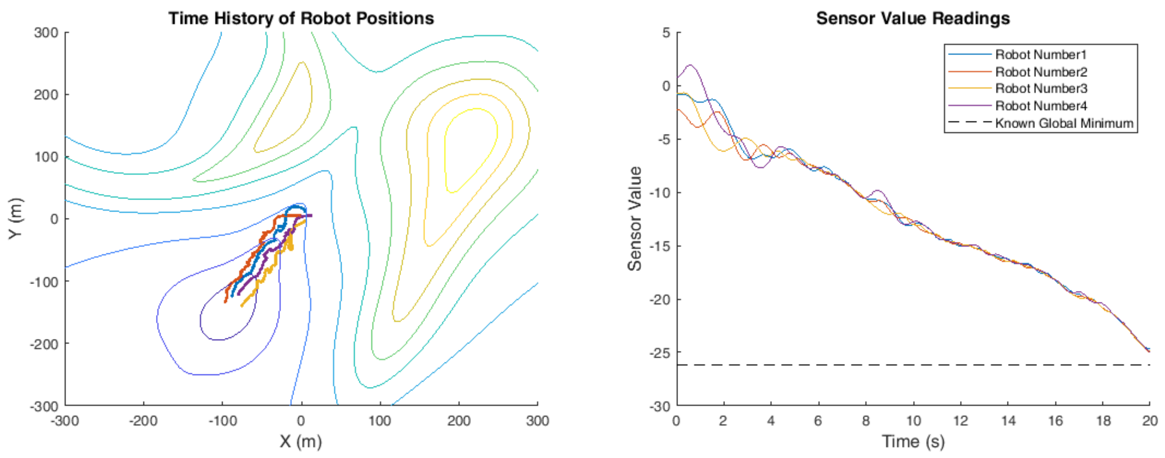


Figure A5. Min Finding, *N=4* Robots

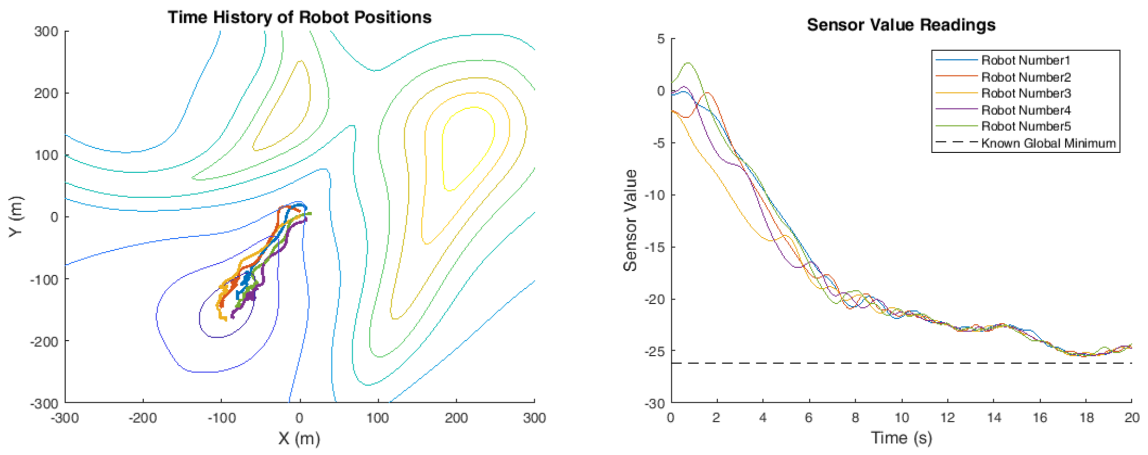


Figure A6. Min Finding, *N=5* Robots

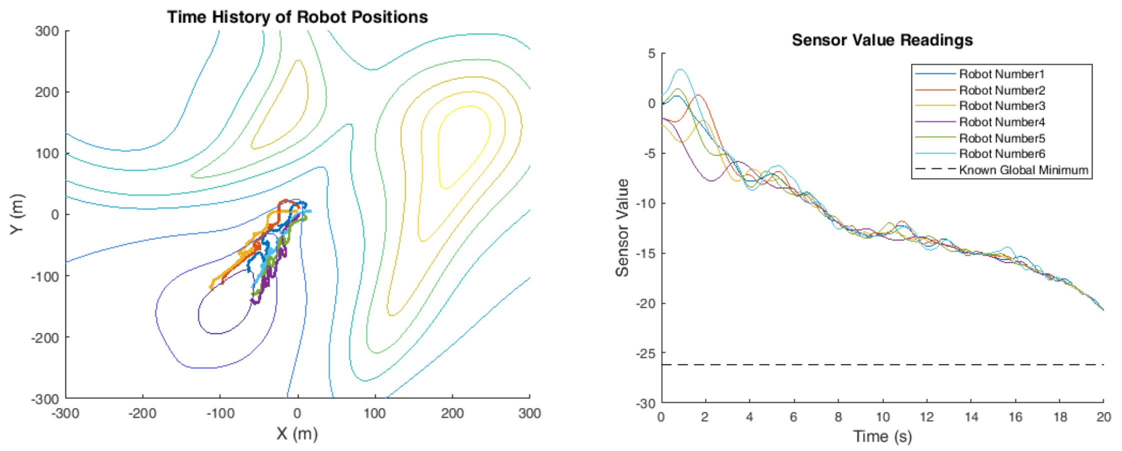


Figure A7. Min Finding, *N=6* Robots

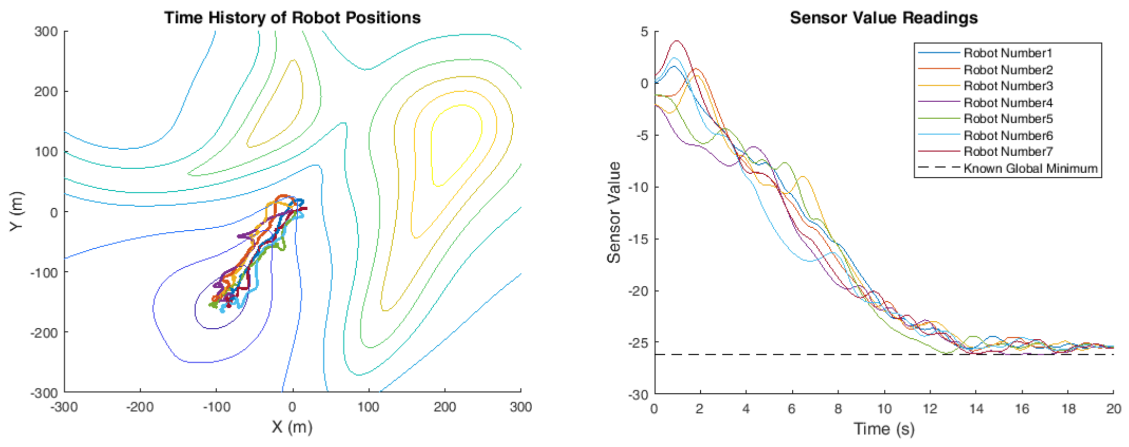


Figure A8. Min Finding, *N=7* Robots

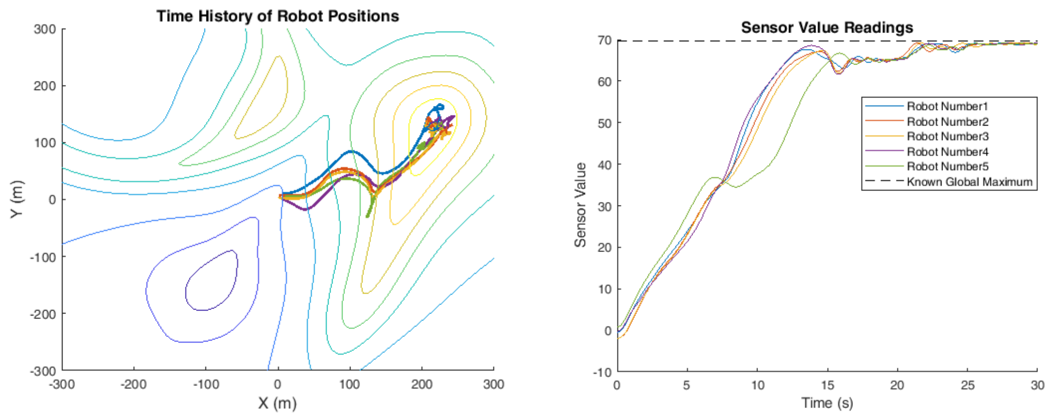


Figure A9. Max Finding, Obst. Avoid = 5m

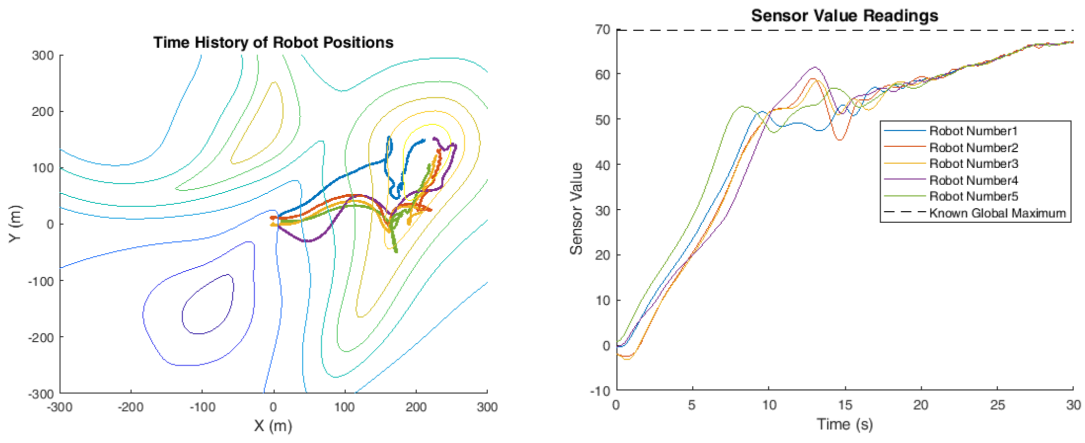


Figure A10. Max Finding, Obst. Avoid = 10m

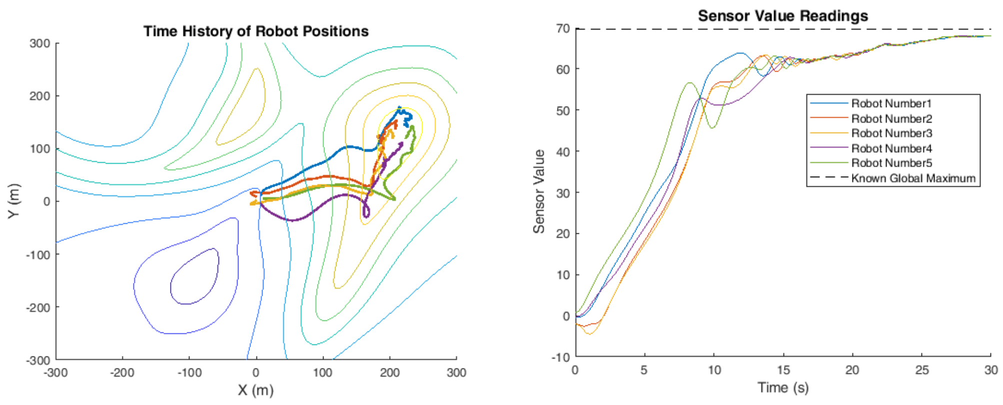


Figure A11. Max Finding, Obst. Avoid = 15m

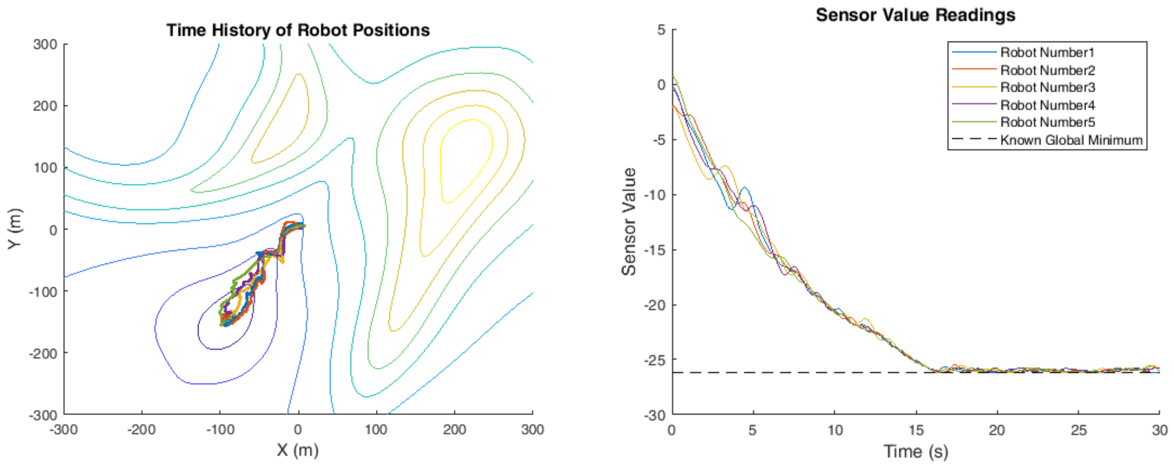


Figure A12. Min Finding, Obst. Avoid = 5m

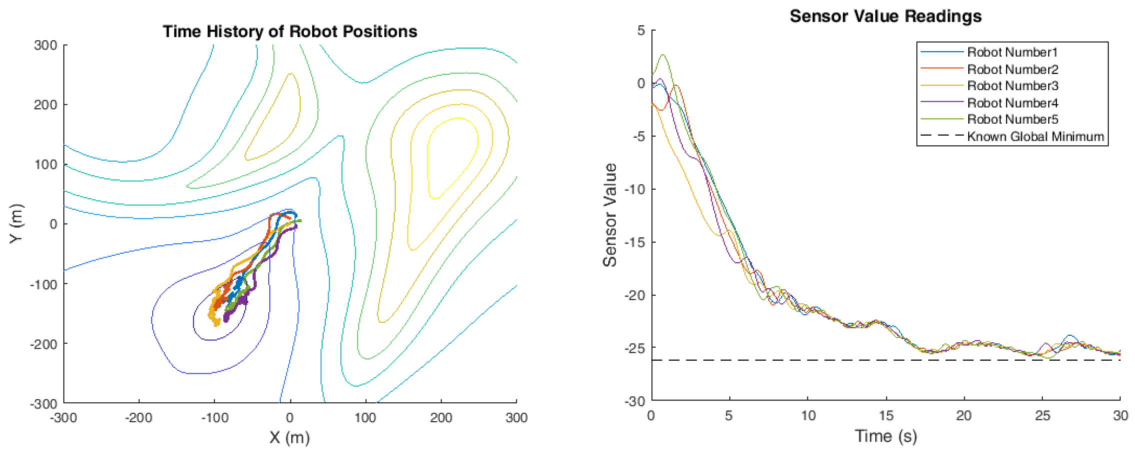


Figure A13. Min Finding, Obst. Avoid = 10m

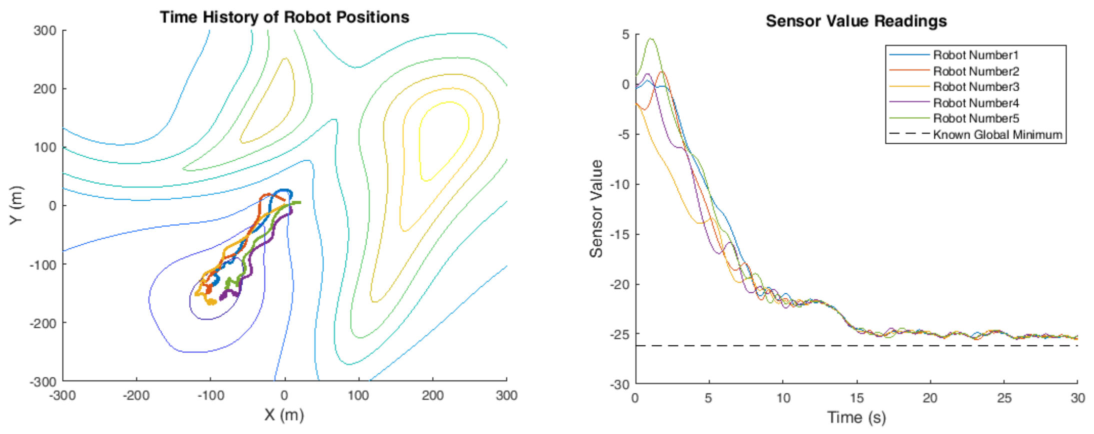


Figure A14. Min Finding, Obst. Avoid = 15m

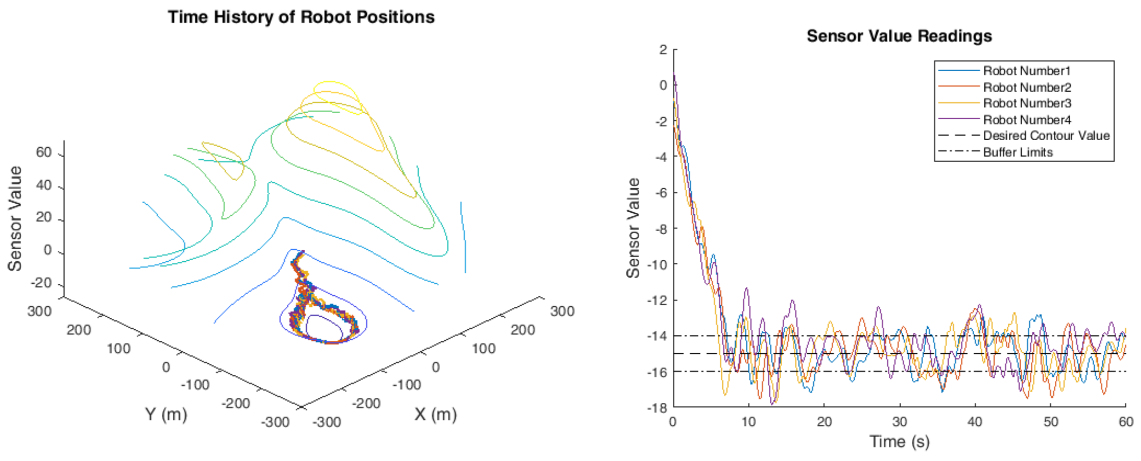


Figure A15. Contour Following, *N=4* Robots

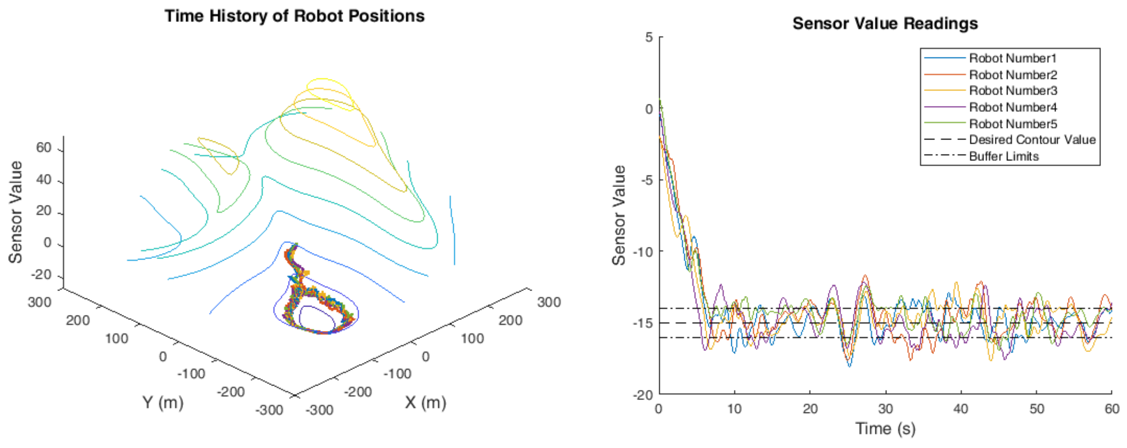
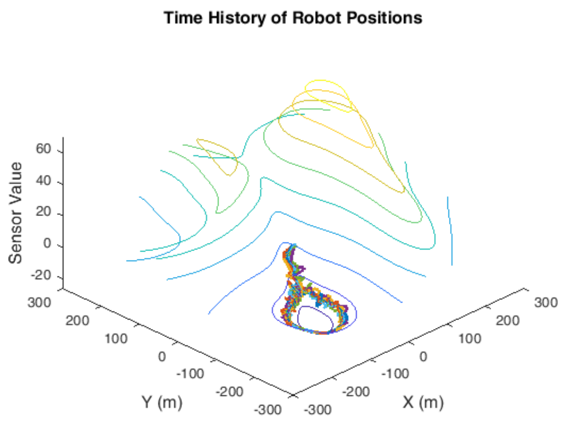


Figure A16. Contour Following, *N=5* Robots



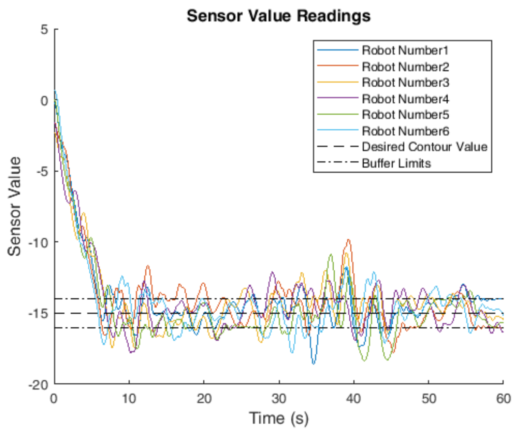
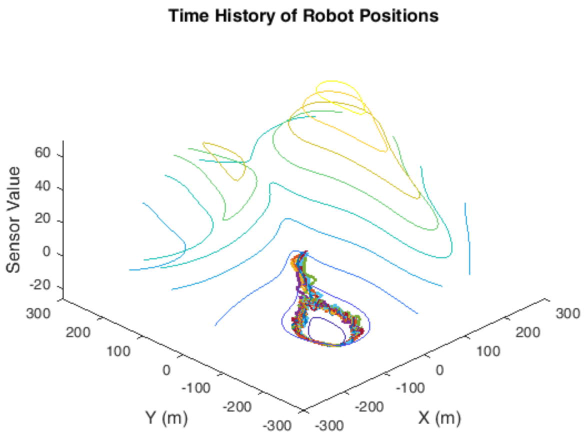


Figure A17. Contour Following, *N=6* Robots



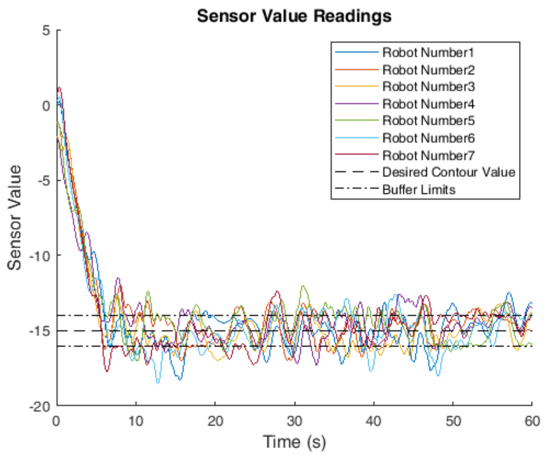
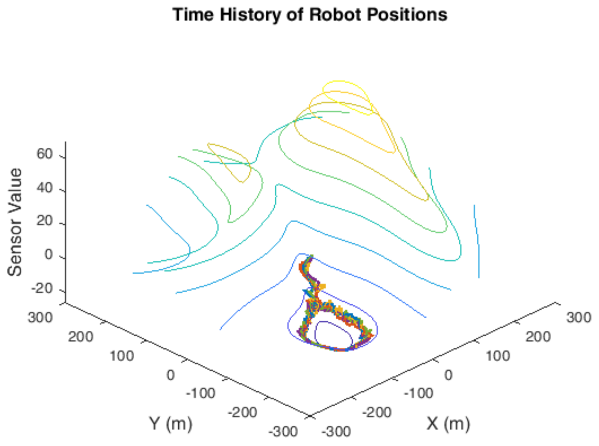


Figure A18. Contour Following, *N=7* Robots



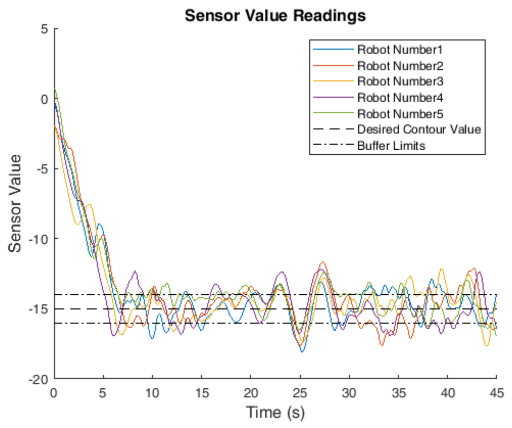


Figure A19. Contour Following, Obst. Avoid = 5m

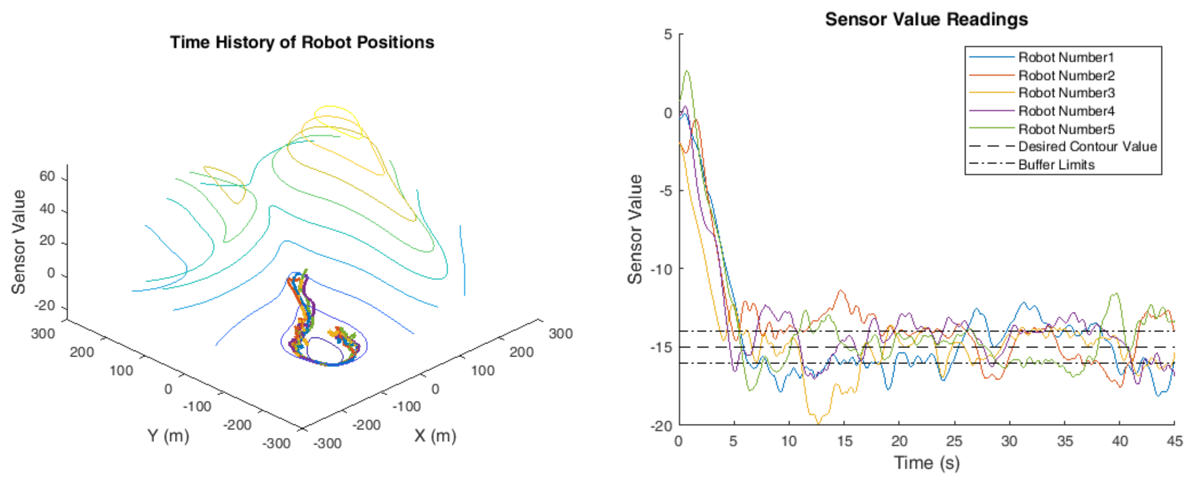


Figure A20. Contour Following, Obst. Avoid = 10 m

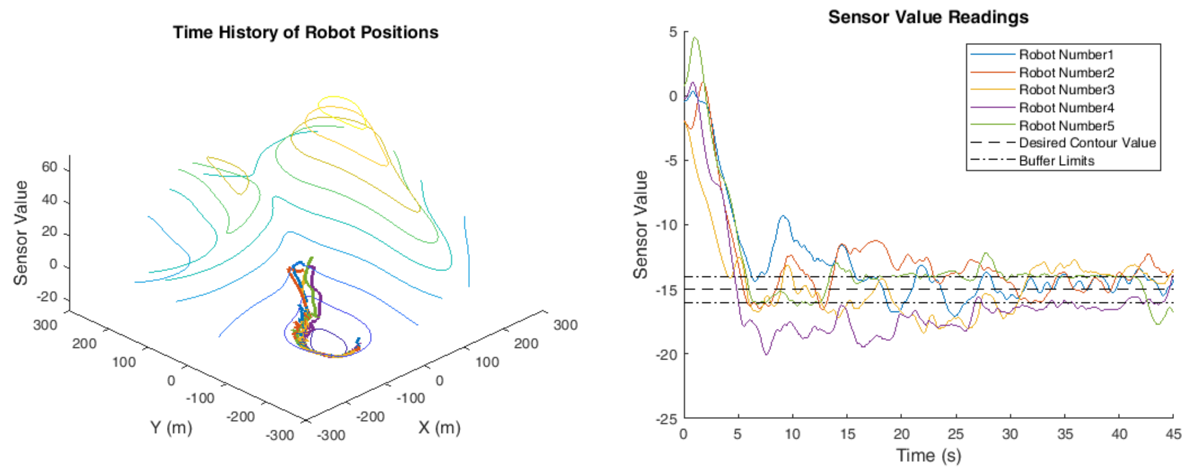


Figure A21. Contour Following, Obst. Avoid = 15m

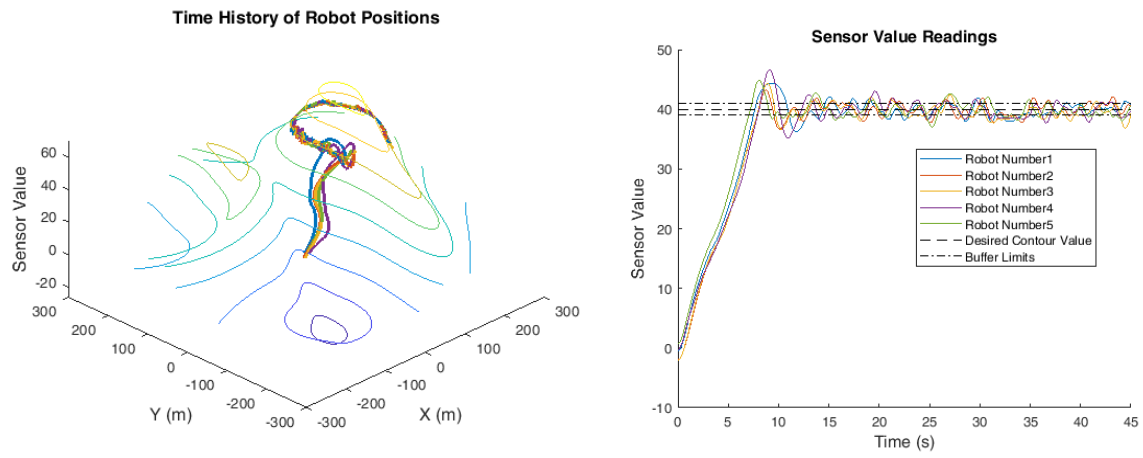


Figure A22. Contour Following, Buffer=1

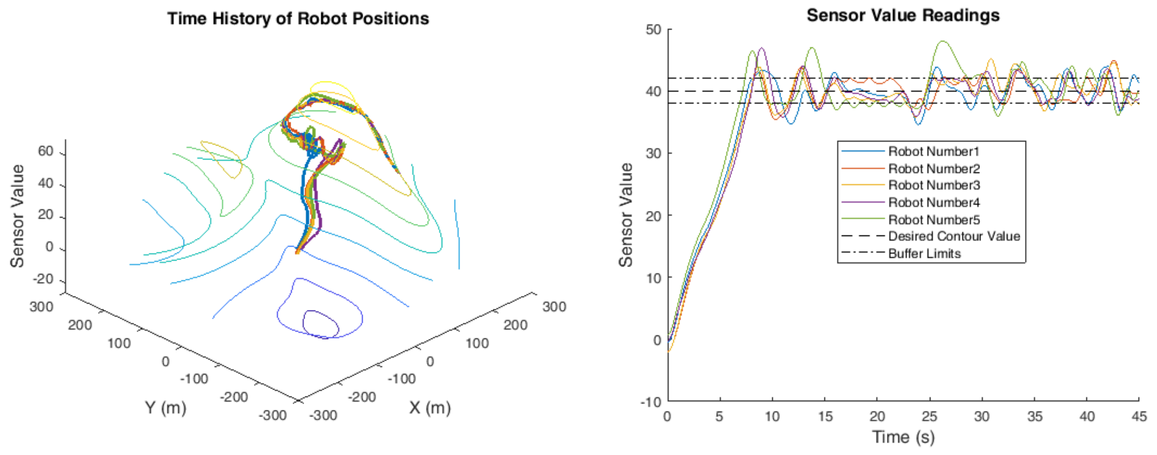


Figure A23. Contour Following, Buffer = 2

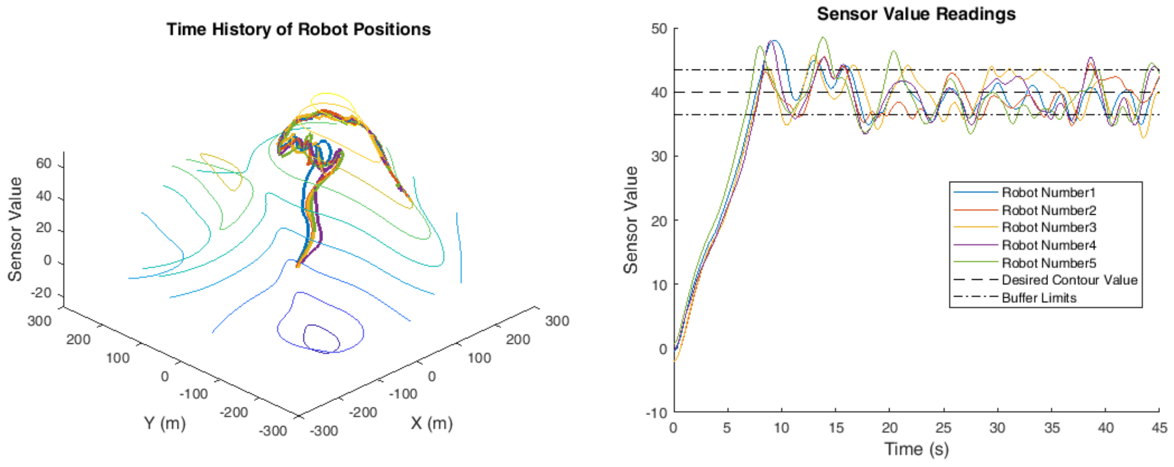


Figure A24. Contour Following, Buffer = 3.5

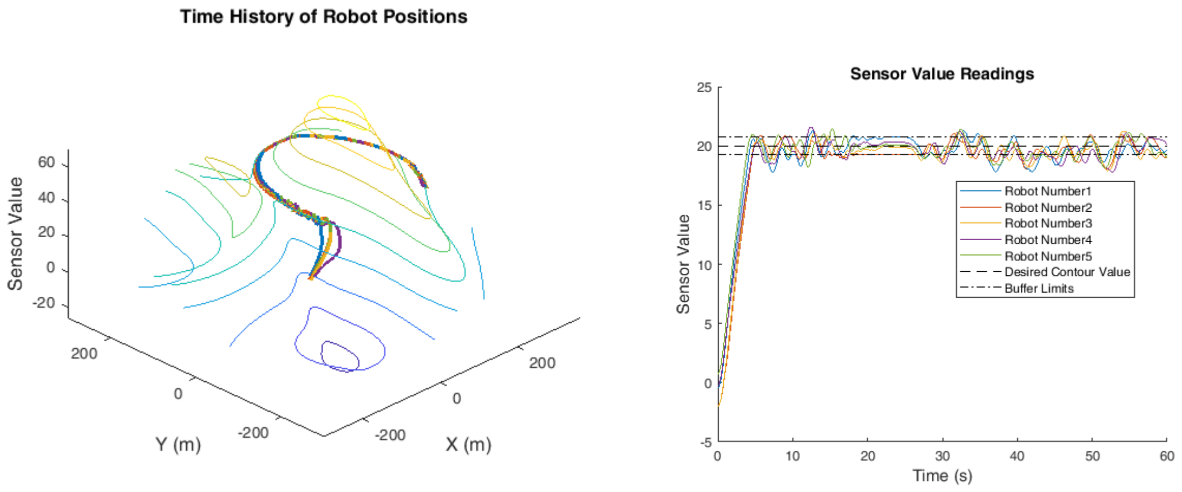


Figure A25. Contour Following, Speed = 30

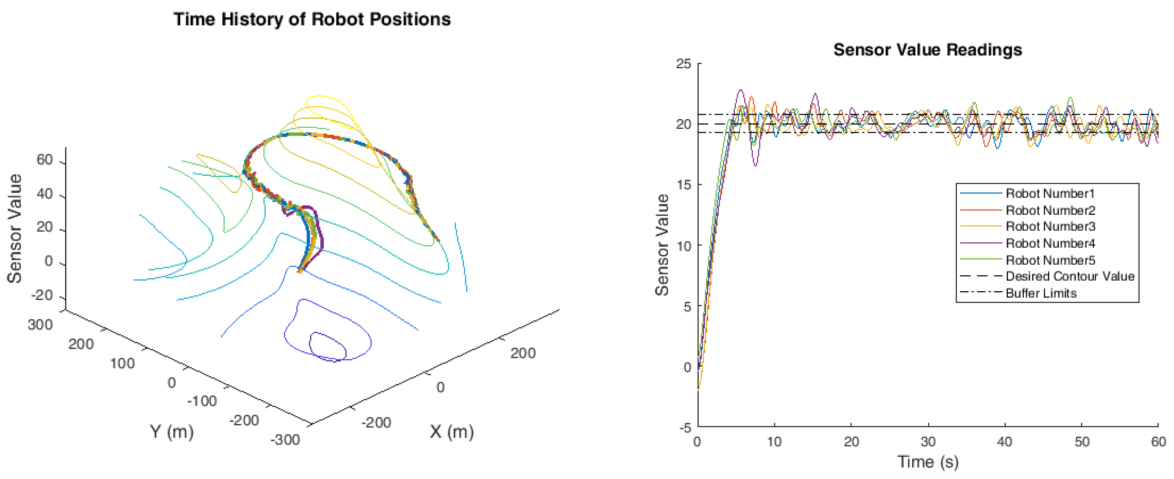


Figure A26. Contour Following, Speed = 45

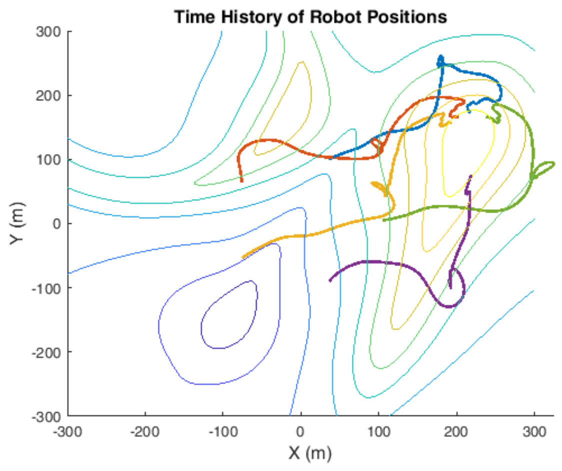


Figure A27. Position History for Extrema Finding for Initial Conditions (100, 5, 5)

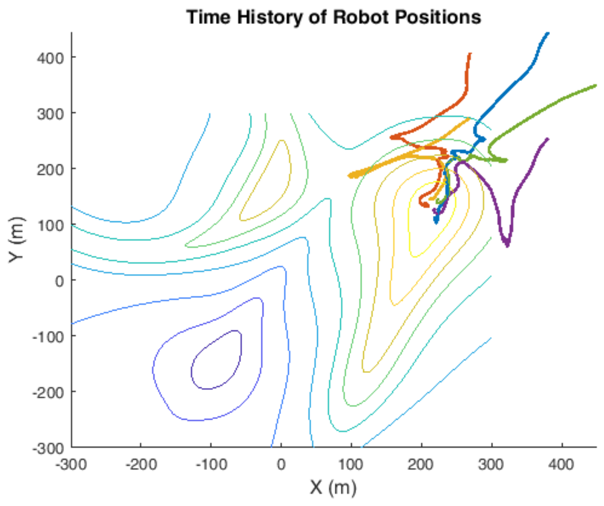


Figure A28. Position History for Extrema Finding for Initial Conditions (100, 350, 350)

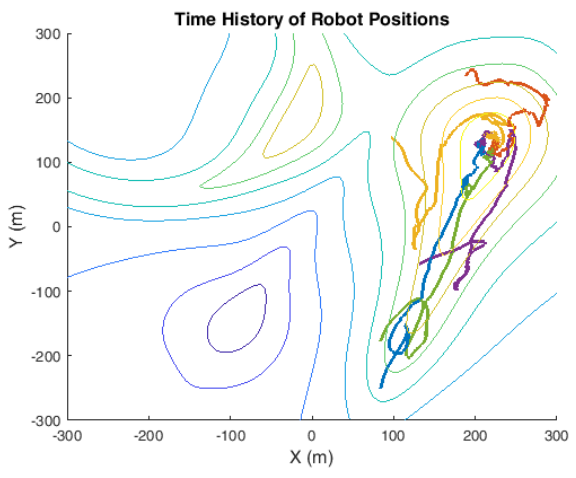


Figure A29. Position History for Extrema Finding for Random I.C.'s, Trial 2

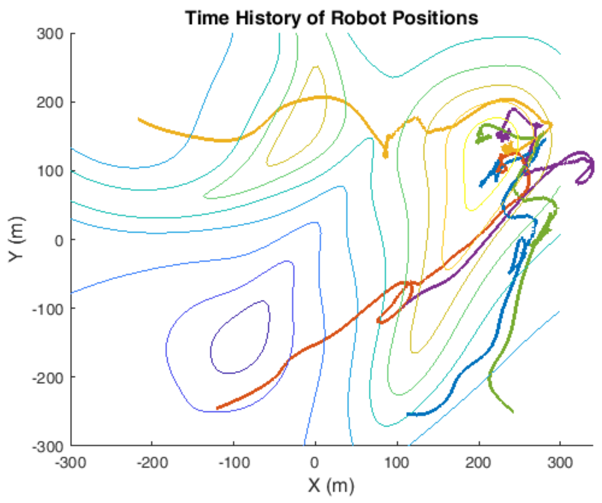


Figure A30. Position History for Extrema Finding for Random I.C.'s, Trial 3

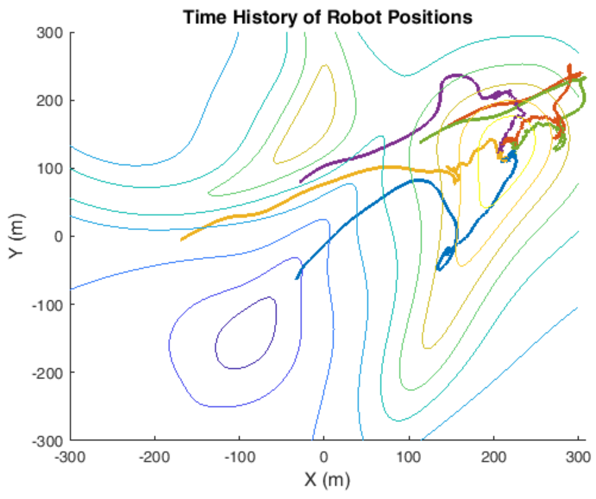


Figure A31. Position History for Extrema Finding for Random I.C.'s, Trial 4

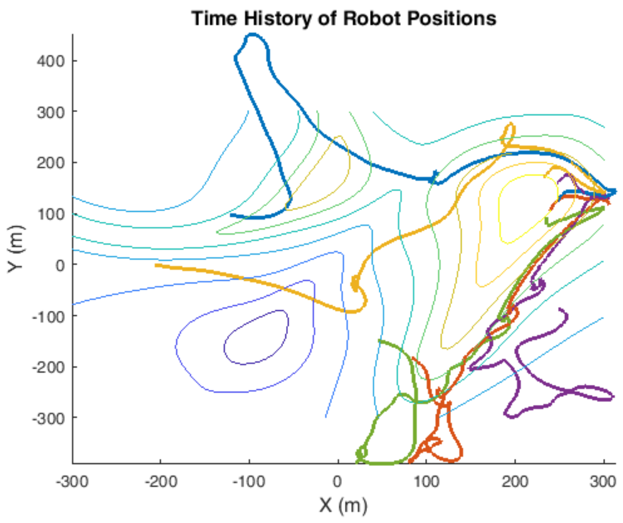


Figure A32. Position History for Extrema Finding for Random I.C.'s, Trial 5