U.S.N.A. --- Trident Scholar project report; no. 328 (2004)

SYNTHESIS OF A CONTROLLER FOR SWARMING ROBOTS PERFORMING UNDERWATER MINE COUNTERMEASURES

by

Midshipman Yong Chye Tan, Class of 2004
United States Naval Academy
Annapolis, Maryland

____
(signature)

Certification of Adviser Approval

Associate Professor Bradley E. Bishop
Weapons and Systems Engineering Department

____
(signature)

____
(date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Deputy Director of Research & Scholarship

____
(signature)

____
(date)

REPORT DOCUMENTATION PAGE	**Form Approved** **OMB No. 074-0188**

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** 5 May 2004	**3. REPORT TYPE AND DATE COVERED**

4. TITLE AND SUBTITLE Synthesis of a controller for swarming robots performing underwater mine countermeasures	**5. FUNDING NUMBERS**
6. AUTHOR(S) Tan, Yong Chye, 1980-	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) US Naval Academy Annapolis, MD 21402	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** Trident Scholar project report no. 328 (2004)

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT This document has been approved for public release; its distribution is UNLIMITED.	**12b. DISTRIBUTION CODE**

13. ABSTRACT:

This Trident Scholar project involved the synthesis of a swarm controller that is suitable for controlling movements of a group of autonomous robots performing underwater mine countermeasures (UMCM). The main objective of this research project was to combine behavior-based robot control methods with systems-theoretic swarm control techniques to achieve a hybrid that has the best characteristics of both. The sub-goals were:
a) To simulate and study a simplified version of the UMCM problem, in 2D with basic robot dynamics and behaviors.
b) To investigate the performance of both behavior-based and systems-theoretic controllers for UMCM, and to determine their advantages and disadvantages.

Careful development of behavior-based methods using a non-traditional differential equations approach facilitated the hybridization of the two controllers under study, giving rise to a more functional controller capable of controlling swarm level functions while executing the appropriate behaviors at the same time.

14. SUBJECT TERMS: Swarm Control, Autonomous Mine Hunters	**15. NUMBER OF PAGES** 111
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**

Abstract

This Trident Scholar project involved the synthesis of a swarm controller that is suitable for controlling movements of a group of autonomous robots performing underwater mine countermeasures (UMCM).

The main objective of this research project was to combine behavior-based robot control methods with systems-theoretic swarm control techniques to achieve a hybrid that has the best characteristics of both.

The sub-goals were:

a) To simulate and study a simplified version of the UMCM problem, in 2D with basic robot dynamics and behaviors.

b) To investigate the performance of both behavior-based and systems-theoretic controllers for UMCM, and to determine their advantages and disadvantages.

Careful development of behavior-based methods using a non-traditional differential equations approach facilitated the hybridization of the two controllers under study, giving rise to a more functional controller capable of controlling swarm level functions while executing the appropriate behaviors at the same time.

Keywords: Swarm Control, Autonomous Mine Hunters

Acknowledgments

First and foremost, I would like to thank my advisor, Professor Bradley E. Bishop, for his guidance and advice. His wealth of knowledge and understanding of a myriad of topics and fields not only helped me in expanding my knowledge in Systems Engineering but also kept me grounded on what was achievable and what was not. Special thanks go out to Professor Shade and the Trident Scholar Committee for their dedication and hard work to ensure that the Trident Scholar Program is running smoothly. The countless long nights plugging through thick stacks of reports is much appreciated. The feedback from members of my Trident Visiting Committee, Prof. Andre, Prof. Zak and LT Hitchcock was especially helpful in making this research more cogent and coherent. Many thanks goes to Mr. Douglas McGee for teaching the finer workings of utilizing the search engines for professional papers, which effectively freed more time, for me, to work on the computers. Lastly, to my sponsors, John and Rose Cantrell, and Douglas Meister for their encouragement and support as well as shelter whenever I had to leave the hall to seek refuge from this project.

Preface

Modeled after colonies of ants or bees, autonomous robots working in cooperation have the potential of achieving complex functions with increased efficiency over single-unit methods. Cheaper, simpler robots that should comprise such a group are potentially suitable for a wide assortment of applications in the civil and military environment.

In the recent wars in Afghanistan and Iraq, robots were used to take over dangerous operations such as surveillance, reconnaissance, mine searching, and other repetitive missions. Multiple robots working together in a hostile environment may prove to be the new paradigm for a war in which they fight alongside soldiers in the air, on land and the sea.

This Trident Research Project focused on combining some of the work and theories conceived by other scientists and engineers to achieve a hybrid controller for underwater mine countermeasures (UMCM). The hybrid controller has the best characteristics of the original controllers with fewer drawbacks.

Underwater mines, which are cheap, easily fielded and capable of causing millions of dollars of damage, pose a major threat to navy ships. With recent developments in technology, the Navy has become interested in employing Unmanned Underwater Vehicles (UUVs) and Autonomous Underwater Vehicles (AUVs). The lack of human involvement is a key factor in employing robots for this application.

Eventually, robots will be programmed to conduct searches for mines autonomously and cooperatively, hence bringing greater automation as well as freedom of reign in water to this tricky problem. The results outlined in this report serve to demonstrate features of certain techniques as well as highlight a new hybrid controller that may eventually be implemented as a commercially viable controller.

Table of Contents

List of Figures

1. Background

A robotic swarm can be described as a group of simple robots working cooperatively. Well-designed swarms ideally perform the same tasks as complex robots, but with increased efficiency and robustness, to withstand damage and environmental fluctuations. These advantages indicate that swarms of robots may prove to be very useful in the harsh and taxing conditions associated with underwater mine countermeasures (UMCM).

Common traditional UMCM methods include using specialized ships, mine hunters, and mine sweepers to perform mine clearance operations, or relying on divers to defuse mines by hand. However, sailors are the most important asset to the Navy, and the objective of using modern technology is to ensure their safety. In the 1990s, Remote-Operated Vehicles (ROVs) were used extensively for mine hunting and only in recent years have advances in computing capability and understanding of automated craft allowed Autonomous Underwater Vehicles (AUVs) or Unmanned Underwater Vehicles (UUVs) to become viable alternatives to ROVs.

Currently, AUVs are employed on Navy ships, but most deployments use only a single unit or a pair of cooperating units to carry out simple tasks. To increase performance in UMCM, suitable control architectures must be developed for swarm AUVs cooperatively carrying out a deliberate search scheme. Such controllers will offer substantial increases in performance in the critical domain of UMCM, and may be ideal for solving the age-old problem of undersea mines. Applying a swarm of robots to mine hunting will improve efficiency, widen the search area as well as reduce the search time, and will be clearly observable (as with humans… more searchers are better), outweighing the performance of one or two robots. Adding a hybrid controller may open up further applications for autonomous deep-sea exploration or discovery of liquid environments in other planets. Lastly, the cost of replacing a robot is less should one simple robot be destroyed as compared to that of a sophisticated one.

Hardware will never work without the software that drives it and gives it intelligence. Several existing control strategies were considered for application in maneuvering a swarm. The two main groups of controls that were studied extensively are the systems-theoretic and behavior-based approaches. [1,2]

Systems-theoretic methods have provable performance and a well-understood design methodology, but are only viable when the environment is predictable and most, if not all, of the information about the environment is known. Rigid programming practices in some of these methods tend to generate pre-planned algorithms that try to control the smallest behaviors. By contrast, behavior-based approaches are flexible, simplistic and require neither large amounts of information nor specific details about the environment; they give rise to indeterministic yet interesting and useful 'emergent' outcomes as a result of their interactions with the environment.

A highly effective method of controlling multiple robots is the statistical approach, which is currently best classified as a systems-theoretic technique. The statistical method controls swarm-level functions (such as the mean and variance of the swarm) to direct the movement of the robots. During the course of this project, it has been demonstrated that there is a possibility of combining behavior-based and statistical approaches to give a better, more functional controller. The new controller generated is able to maneuver and direct a swarm of robots more effectively than either of the constituent methods. That hybridization became the primary focus of this research.

Several requirements were set to select viable and effective controllers for robotic UMCM. The first required that the robots be fairly simple in structure so that, while operating individually, they will not be bogged down by computational complexity. In this project, robots were assumed to be holonomic (able to move in any direction from any pose) and velocity controlled. The next requirement was that the controller had to create a system such that a fairly autonomous robot swarm is able to make decisions on its own, without depending on an external source of information. To achieve this autonomy, each robot was required to compromise some personal freedom to contribute to the overall independence of the swarm. Centralized (each unit given its orders by a central controller) as well as decentralized (each unit acts independently) approaches were considered to determine which form of control was more applicable to UMCM.

This paper is organized as follows. An overview of behavior-based robotics, its roots and developments, is given in Section 2, followed by two separate sections on the favored behavior-based structures: Motor Schema (Section 3) and Subsumption (Section 4). The relevance of each method to mine countermeasures is discussed in its respective section. Section 5 covers the basics of statistical control and explains some discoveries that allowed behavior-based methods to be merged with that architecture. Section 5 also details the components of the statistical controller and how they affect its actual operation. Section 6 touches on the hybridization of the controllers and how it affects the predicted outcomes of the system as well as its performance under simulated environmental effects. Section 7 summarizes the process of the research and the conclusions drawn from it.

2. Introduction to Behavior-Based Robotics

What exactly are robots? According to the definition given by the Robotics Institute of America (RIA), “a robot is a reprogrammable, multi-functional, manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks”[3] Robots involved in this project, however, belong to a more sophisticated breed. These Autonomous Underwater Vehicles (AUVs) or Unmanned Underwater Vehicles (UUVs) are categorized as *intelligent robots*. “An intelligent robot is a machine that is able to extract information from its environment and use knowledge about its world to move safely in a meaningful and purposive manner.”[4] In this project, simulated AUVs and UUVs were employed to detect mines in the highly variable oceanic environment, using information collected to avoid obstacles while at the same time relaying mine information to a mother ship.

Artificial intelligence plays an important role in enabling robots to behave in a manner similar to humans, conducting their own operational decision processes. Thus, the robots used in this project needed to be able to recognize and differentiate between pieces of information and make links to the particular tasks designated by the correct information.

Neural networks and fuzzy logic are several common means of recreating human-like decision structures in robots.[5] These systems are especially useful when mimicking human decision structures in performing a complex, hard-to-model task. Unfortunately, these methods typically require a great deal of training (generating enough runs to allow the system to learn and create a response) to be successful at even the most straightforward tasks.

Behavior-based robotics is a much simpler method of control, based on the decision-making capabilities of lower life forms. The initial phase of this project focused on the evaluation of behavior-based methods for the UMCM problem. This method was studied first because it is known to be a reactive controller that does not require exhaustive amounts of information. Behavior-based robotics has two main types of structures that make it a versatile controller for enabling several robots to move in cooperation.

Behavior-based robotic controllers were originally devised for robots that were to be used in environments that are difficult to model.[6] This class of controllers reacts readily to the changes in the environment, unlike many traditional systems-theoretic control approaches in which all the unknown information or details in an area have to be accounted for to ensure that the robotic agents are able to move.

Behavior-based systems, characterized by a reactive nature, generate robot responses based on current sensory information alone, typically through a very simple sensor-motor mapping.[7] By responding directly to a stimulus, the system efficiently discards all other environmental factors irrelevant to the problem. Simple low-level behaviors (such as ‘Avoid_Obstacle’ or ‘Seek_Light’) are combined to form a modular, often hierarchical, behavior system, which collectively reacts to a variety of stimuli in varying ways.[8] This combination of simple behaviors also leads to making the system completely indeterminate. Another interesting characteristic of behavior-based systems is that in event of a sensor breakdown, the robot will default to a lower-level behavior and perform its task as best as it can.

Behavior-based decision processes do not utilize the human cognitive process, but follow a simple sense-think-act sequence.[9] Each robot uses all sensory information it gathers to select actions guided by specific rules. If each member of a robot swarm were equipped with an independent behavior-based controller, the members would be able to move independently, a highly regarded characteristic, since it places little computational strain on the central controlling body. In other words, this type of control is promising for creating a decentralized autonomous underwater control system of multiple robots. Behavior-based control was therefore an excellent candidate for generating a fast and efficient means of reacting in a hostile and unknown environment for the UMCM problem.

Two specific behavior-based architectures were used for testing and implementation of robotic UMCM: motor schema and subsumption. Both of these approaches were originally designed and intended for single-robot (decentralized) control. Under these architectures, the robots each act independently, and the overall swarm behavior arises from their interactions with the environment and each other.

The study of schema-based systems was adapted from Dr. Arbib, who made a link between behavioral expressions in nature and behavior-based controls in robotics. [10,11] The schema-based theory combines behaviors for numerous tasks at the same time, and demonstrates how the system can react based on the calculation of simple equations. Dr. Ronald Arkin, who first addressed the implications of using motor schema for navigation in 1987 and later published several related papers, was the first researcher who focused on the method's applications on autonomous robotics.[12] He proposed that motor schema could be set up in such a way that there were direct relationships between sensors (perception agents) and the motors (end effectors) present in the system. The behavioral responses were represented in vector forms using an Artificial Potential Fields technique that will be discussed in the next section.

The second approach for behavior-based architectures used subsumption structures. Dr. Rodney Brooks of Massachusetts Institute of Technology first devised the concept of subsumption architecture in 1986.[13] Brooks' proposal was also a purely reactive system, which allowed the robot to decide what to do next, based entirely on current sensor data, without any structured planning.[14] Subsumption-based systems were built hierarchically with increasing complexity. Only one behavior was expressed at any point in time as a result of a suppressor function that restrained other lower behaviors within the hierarchy. The subsumption architecture was therefore a discrete rule-based process that allowed reactions to be triggered by pre-defined sensory information. Complex robot motions arose from the combination and switching among simple behaviors.

Brooks' idea was against the mainstream definition of artificial intelligence at that time, which required robotic decision structures to be organized with a complete appreciation of the surrounding environment. Brooks argued that the standard practice of using systems-theoretic methods was preventing the robot from making timely responses and forcing less flexibility in the control scheme. The simplicity and flexibility of the behavior-based approach proved that it was capable of performing well in a highly variable environment.

In conclusion, behavior-based robotics methods allow robots to perform tasks similar to humans by linking given information to a predetermined action. Reacting to particular sources of information and then performing a specific task reduces the need for an understanding of the entire environment. The control structure of behavior-based robots is simplistic and can be manipulated for practical applications. The focus of the next segment of the project was to generate working examples of existing behavior-based architectures for robots performing underwater mine countermeasures.

3. Motor Schema

Motor schema theory states that individual behaviors that act in response to different sensory inputs can be combined to give a resultant control vector that incorporates all of the simple behaviors. If the sensor-motor reactions can be written as simple velocity control vectors, this can be accomplished by a vector sum. The most common approach for motor schema implementations is the use of Artificial Potential Fields.

3.1. Artificial Potential Fields Theory

Artificial Potential Fields (APFs) arose in the early days of robot motion planning as a simple, computationally efficient planning routine. APF methods use simplified version of laws of nature, attractive and repulsive potential, to draw or repel an object (the robot) from one point to another.[15] In motor schema, an attractive or repulsive potential constitutes a behavior that arises as a result of a sensory stimulus. A wide variety of behavior commands can be derived, such as Avoid_Obstacle, Move_to_Light and Random_Search.

APF theory uses the positions of two points of interest to calculate attractive and repulsive vectors. The potentials are based on the vector connecting the two points of interest (in 2D for purposes of this work). The nature of the resulting potential will depend on the underlying behavior. The expressions for attractive potentials and repulsive potentials are slightly different since their desired reaction to a stimulus is inherently different.

Consider the example of an attractive potential for a robot moving towards the calculated centroid of a swarm (hence, an Aggregation behavior). The gradient of the potential field is determined by finding the difference between the position of the robot and that of the centroid. The angle between the two points is found by using the inverse tangent in the world coordinate frame. Lastly, the attractive vector in the x direction is found by multiplying the magnitude of the original vector by the cosine of the angle found, while the vector in the y direction is found the same way, with the only difference being multiplication with a sine. Example MATLAB code for developing the attractive vector is given below, where [*Xc*, *Yc*] is the center of the group of robots and [ROBOT(i,1), ROBOT(i,2)] are the (*x*, *y*) position of a robot of interest.

```
Mag_change=norm ([Xc-ROBOT (i, 1), Yc-ROBOT (i, 2)]); % finding the magnitude
theta_att=atan2 (Yc-ROBOT (i, 2), Xc-ROBOT (i, 1)); %angle between two points

att (i, 1)=Mag_change*cos (theta_att);    % find x attractive potential
att (i, 2)=Mag_change*sin (theta_att);     % find y attractive potential

attctrl (i,:)= Katt* att (i,:);       % attractive potential towards each other, Katt is the vector gain
```

The effect of this attractive potential draws the robot towards the centroid. An interesting observation is that the attractive potential results in rapid motion towards the destination if the distance is large but decreases gradually as the robot approaches the final position, finally stopping at the target.

The repulsive potential, on the other hand, does the complete opposite. The final calculation of the repulsive potential is highlighted by the multiplication of the x and y components of the vector with a (1/distance) factor. This factor creates the opposite behavior of that of the attractive behavior. As the robot moves towards an obstacle (some object in the environment or even another robot), the magnitude of the distance between the two points becomes smaller, approaching zero. As a result, the repulsive potential jumps to infinity when the robot is close to the object. This infinite repulsive vector guarantees that there can be no collision unless some other vector also increases unboundedly. Thus, it is common in motor schema systems to generate potentials that are all upper-bounded with the sole exception of the obstacle avoidance routine.

The code below demonstrates the computation of the repulsive vector. The [ROBOT(i,1), ROBOT(i,2)] are the (x, y) position of a robot of interest, [OBSTACLES(position, 1), OBSTACLES(position, 2)] are the position of the obstacles, and *Mag_L* is the magnitude of the distance from the robot to the obstacles. The radius of the circular obstacle, *r1*, will be required for finding the exact distance between the robot and the obstacle so that it mimics the maximum distance from the robot to the external radius of the obstacle. *Rho* is the radius of the obstacle plus the sensing radius of the robot to the external radius of the robot. The calculation with the radius and angles accounts for the distances between the robot and the edge of the obstacle. *Repobst* represents the repulsive vector, which gives the resultant motion of generating a strong repulsive vector away from the edge of the obstacle as the robot approaches it.

$$Repobst = \frac{1}{Mag\_L}\cos(theta\_obst) - \frac{1}{Rho - r1}\sin(theta\_obst)$$

```
theta_obst=atan2 (ROBOT(i, 2)-OBSTACLE(position, 2), ROBOT(i, 1)-OBSTACLE(position, 1));
% angle of repulsion form obstacle

Xdist=(ROBOT (i, 1)-OBSTACLE (position, 1)-rl*cos (theta_obst));
Ydist=(ROBOT (i, 2)-OBSTACLE (position, 2)-rl*sin (theta_obst));

% Vector to show length of repulsion vector to obstacle
Mag_L=norm ([Xdist; Ydist]); %normalize vector for magnitude of length
repobst (i, 1)= ((1/Mag_L)*cos (theta_obst))-((1/(rho-rl))*cos (theta_obst));
repobst (i, 2)= ((1/Mag_L)*sin (theta_obst))-((1/(rho-rl))*sin (theta_obst));
```

The example shows that the *theta_obst* and the magnitude, *Mag_L* is computed similarly to that of the attractive potential but the repulsive vector differs in the 1/*Mag_L*.

Figure 1 shows how the attractive and repulsive vectors are summed to produce a resultant vector that directs the path of the robot. As the robot moves towards the endpoint, the attractive vector becomes shorter, representative of the decreasing value. The value of the repulsive vector steadily increases as the robot gets close to the obstacle, and pushes the robot away from it, preventing a collision.

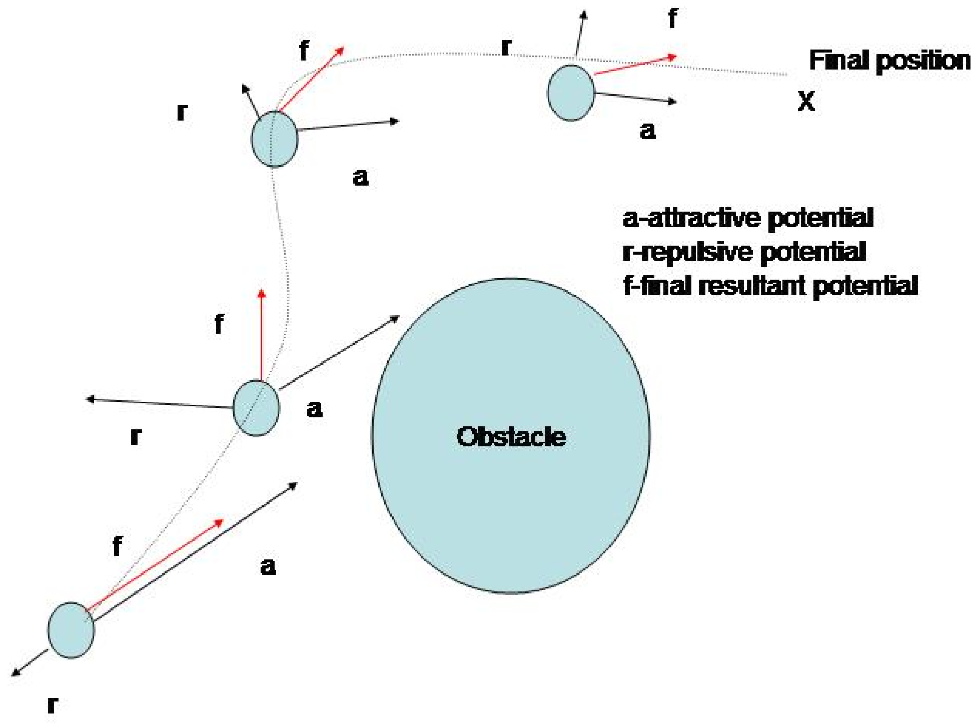


Figure 1: Conceptual diagram of motion of a robot influenced by Artificial Potential Fields

3.2. Motor Schema Controllers for UMCM

The potential fields methods described above are used in a behavior-based control method called 'motor schema'. The motor schema architecture is shown in Figure 2. The first characteristic of motor schema is similar to most other behavior-based approaches: it is dependent on the environment to provide stimulus to the system to create some reaction. The next characteristic is the presence of numerous environmental sensors designed to pick up specific changes in the environment so that the appropriate response can be triggered. The information picked up by the environmental sensors is transferred to the perception schema, which triggers the specific response programmed. A number of these perception schemas derived from different sensors play a part in creating an overall composite motor schema to tell the robot how to move. Several motor schemas contribute to create a single summation behavior. Figure 3 illustrates how the specific behaviors can be summed to generate a resultant motion in accordance to the motor schema structure.

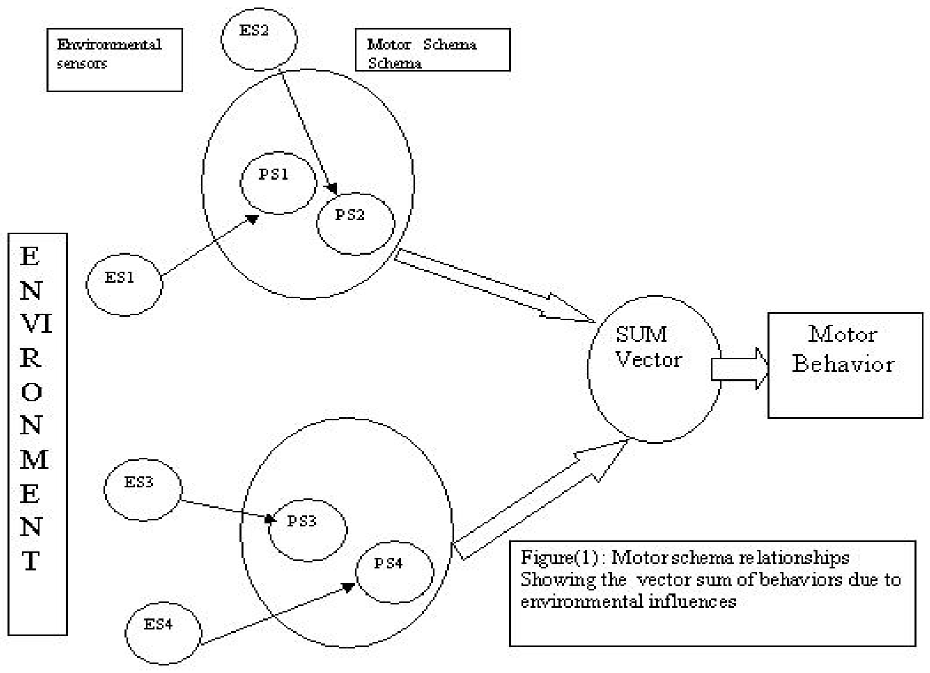


Figure 2: Motor schema diagram[16]

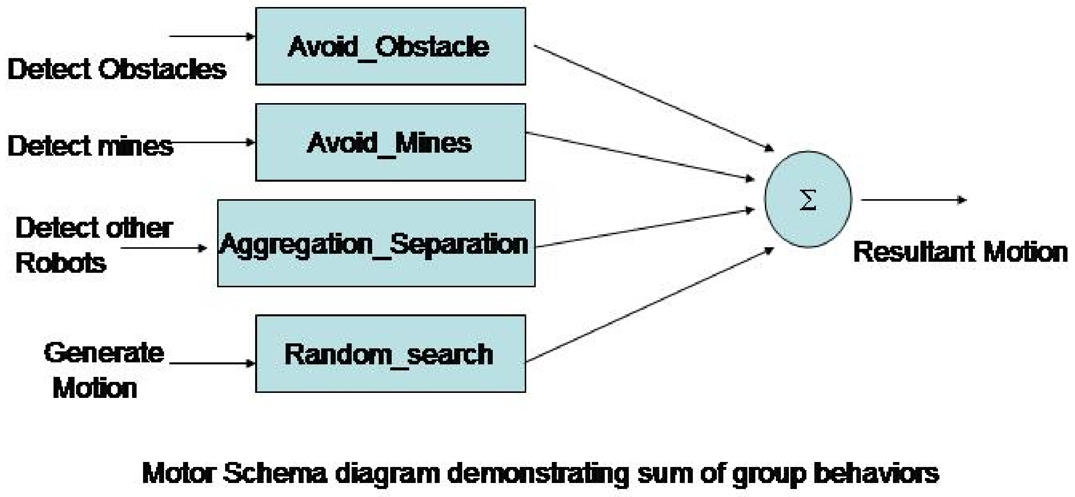


Figure 3: Motor schema architecture for mine hunting

3.2.1 Motivation for Using Motor Schema in Robotics in General

There are several reasons why motor schemas are created and applied in behavior-based architectures. First of all, motor schemas essentially consist of simple systems that are fundamentally easy to execute and debug, which is helpful for generating a robust system of control. This is achieved by using simple individual behaviors as the building blocks of a complex system. Complexity can thus be built from the summation of these simple behaviors. Additionally, a system using motor schema is very capable of working effectively in an unknown dynamic environment. By having the environmental sensors tuned only to pick up specific information, other uncertainties and contributing errors are removed. Furthermore, this allows relevant information to be used only at a specific point in time once the environmental sensors have been fired. Lastly, the lack of complex, pre-determined planning structure simplifies the system. A motor schema system makes use of pre-determined direct relationships between sensors and receptors, and allows the system to run freely. Therefore the system is indeterministic in nature and results are an expression of the behaviors due to the environment.

3.2.2 Previous and Current Work

Dr Ronald Arkin of the Georgia Institute of Technology has done most of the current work on applying APF to the motor schema behavior-based architecture. His book, *Behavior-Based Robotics*, has become a popular text with college professors who are teaching about behavior-based robotics. His book grew from his 1988 paper on Artificial Potential Fields theory, and his belief in the lack of a suitable text that addresses the theories and applications of behavior-based artificial intelligence. Much additional research has stemmed from his discoveries. [17,18]

3.2.3 Application of Motor Schema to UMCM

The motor schema approach is extremely well suited for mine hunting, since the undersea environment is relatively unknown and filled with numerous uncertainties. Hence, engineers using motor schema in creating robots with behavior-based architectures may be more successful in such dynamic environments. Furthermore, implementation in real life robots will be relatively straightforward. The sensors used on a typical prototype would be situated around the robot to increase sensory detection in the environment as the robot conducts a search in a delineated area. Furthermore, motor schema, when programmed into individual robots, can easily provide the basic instructions for moving and finding mines. These programmed basic functions, combined with functions that prevent collisions, will give the robot some autonomy. This form of control is classified as decentralized control since the robot does not take orders from a central robot or receive information updates from a mother robot.

3.2.4 Motor Schema UMCM Implementation

Several assumptions and operations are discussed before the experimental results are explained. The simulated robots are assumed to be *holonomic*. Holonomic robots are those that can move in all directions freely regardless of pose, which is unrealistic due to physical limitations (think of parallel parking). However, in the simulations, holonomic robots prove to be extremely useful to demonstrate the effects of different behaviors without worrying about robot morphology and kinematics. Some holonomic systems do exist, and certain mathematical tricks can be employed to make non-holonomic systems appear holonomic.[19]

The robots are all assumed to be velocity controlled. That is, the behavior architecture is designed to provide a desired velocity for the robots, based on sensor input. The robots are assumed at this point to be able to follow the commanded velocity exactly. In more advanced simulations, environmental stimuli such as drift currents can be included to make the situations more realistic. Although acceleration-based controllers are generally preferred to the velocity based form that was employed in this project, the respective simulations do not differ significantly for a well-designed controller.[20] Therefore the velocity-based simulations sufficiently represented the objectives of the project. Expressing the controller in acceleration terms is a viable objective for future work.

The initial simulations presented did not include drift or other environmental effects. It was assumed that the robots had full control of their sensors and that these sensors were able to detect their target stimulus in the water perfectly, although with limited range. Furthermore, the robots were assumed to be fully communicative between each other, indicating a decentralized control.

3.2.4.1 Attractive and Repulsive Vectors

Code was developed in MATLAB (see Appendix A) to simulate a simple attractive-repulsive combination for a single robot. The attractive behavior was 'Begin_Search,' while the repulsive was 'Avoid_Obstacle.' This set of basic functions formed the basis for all future APF-based techniques. The basic concepts are illustrated in Figure 1, while a sample run is shown in Figure 4.

The objective of the sample run shown in Figure 4 was to demonstrate the effects of the repulsive and attractive vector acting on a holonomic robot. The plot shows a holonomic robot starting at point (0,0) and moving towards (3,5) as a result of the attractive vector. As expected from the definition of 'Begin_Search,' the robot initially moved at high speeds and slowed down when it neared the final point. The target point was reached with no collision.

The deviation of the robot from a straight-line path was a result of the repulsive vector created by 'Avoid_Obstacle' as the robot neared the object. The robot had a sensing radius of 0.3 units that kept it away from the object. This result stemmed from the calculation of the repulsion factor where it was multiplied by inverse of the magnitude of the distance between the robot and the obstacle, 1/Mag_L. As the robot approached the obstacle, a huge repulsive vector in the opposite direction was created. Moreover, summed with the attractive vector, the robot moved towards the final point even as it repelled away from the object.

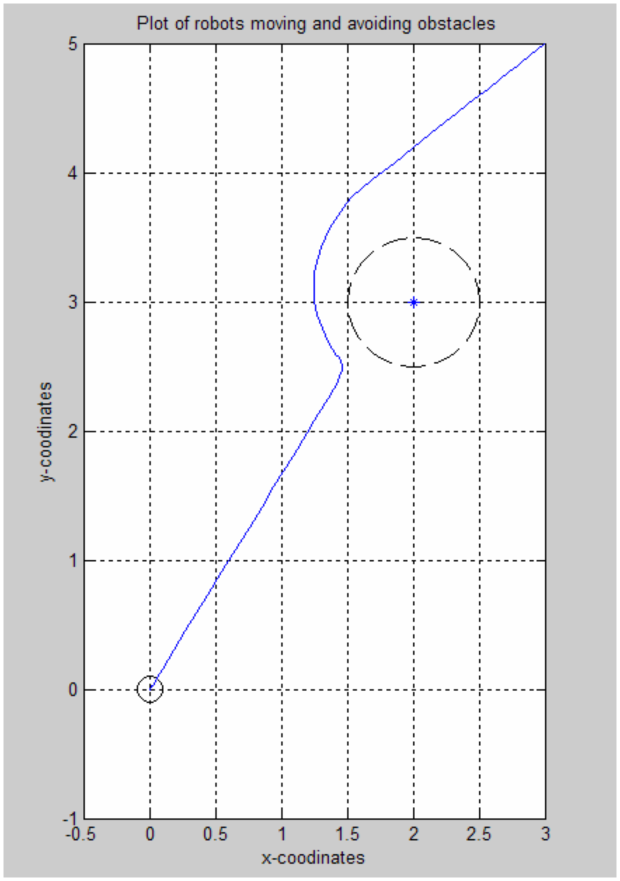


Figure 4: Simulation of a robot avoiding an obstacle combining 'Begin_Search' and Avoid_Obstacle'

3.2.4.2 Attractive and Repulsive Vectors between Robots

The next step in developing a motor-schema based swarm controller was to build behaviors for some interaction between cooperating robots. A simulation was written (see Appendix B) that included three robots, each using an attractive APF toward the center of the three combined with a repulsive APF from nearby robots.

The sample run shown in Figure 5 demonstrates how this simple two-behavior motor schema resulted in three robots staying within a certain distance from each other in an emergent formation (a triangle, in this case) without collision. It is important to note that, since the robots were not specified in a fixed formation, they move around the calculated center of mass, repositioning themselves at the optimum location, where the attractive and repulsive vectors are maximized. This behavior was labeled as 'Aggregation_Separation,' as it combines an attractive potential to the center of mass with inter-robot repulsive potentials. The repulsive and attractive vectors are each controlled by a respective gain, which determines the spread of the robot positions around the center of mass. The behaviors demonstrated were very similar to the first sample run since the repulsive and attractive vectors were utilized to move the robots to a pre-determined point in space, but now multiple robots are cooperating.

Several major observations arose from this simulation. Primary among these is that the robots were kept in a simple formation around the centroid. The three robots kept their distance from the centroid at an equal distance, which created a triangular formation. If there were four robots, the mean formation was a square. As the numbers increased, the mean formation tends to be concentric, with no specific structure. It was concluded that the robots generated the attractive and repulsive vectors correctly, and that this behavior could be combined with the first sample run shown in Figure 4.

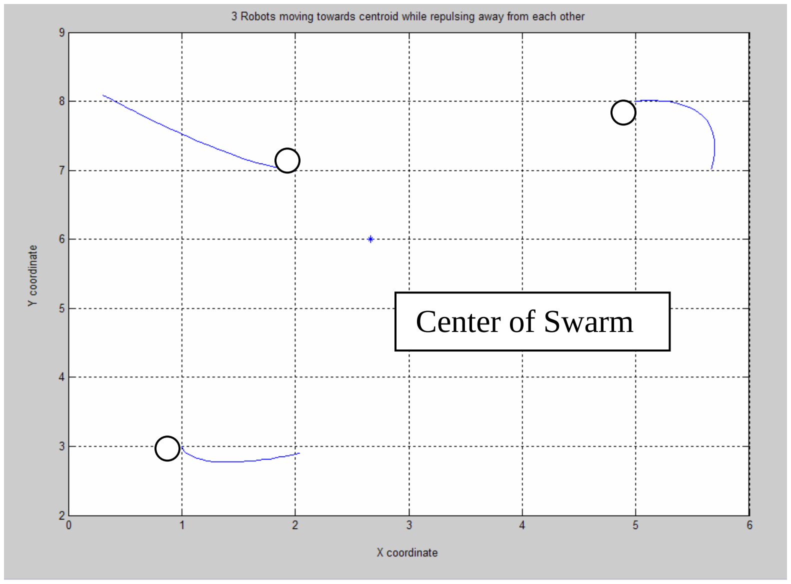


Figure 5: Plot of 3 robots performing 'Aggregation_Separation.' Initial positions are marked with an 'O'.

3.2.4.3 A Robot Swarm Moving Towards an Endpoint

The objective of this next sample run was to combine the two controllers generated in earlier programs. The new controller was designed to move a swarm to a target centroid while maintaining each robot’s position with respect to the others using attractive and repulsive potentials. The constituent motor schema of this sample run included the schema to force the robots towards an endpoint (‘Begin_Search’), to cause the robots to avoid an obstacle while in motion, (‘Avoid_Obstacle’), and to keep a particular distance away from the other robots without collision, (‘Aggregation_Separation’). The results of the simulation can be seen in Figure 6. This run demonstrated the process of building complexity using simple behaviors. Code can be found in Appendix C.

This particular combination of different behaviors lacked a structured approach to organize the priority of the behaviors. The resultant behavior was based on the sum of the different behaviors, so it did not differentiate the order in which the behaviors were determined in the individual decision process of the robot. Moreover, each robot had a sensing radius of 0.3 units, which represented a sensory buffer that would prevent the robots from getting too close to an obstacle or another robot. If a robot detected any obstacles inside this buffer zone, it would create the repulsive vector and repel away from it.

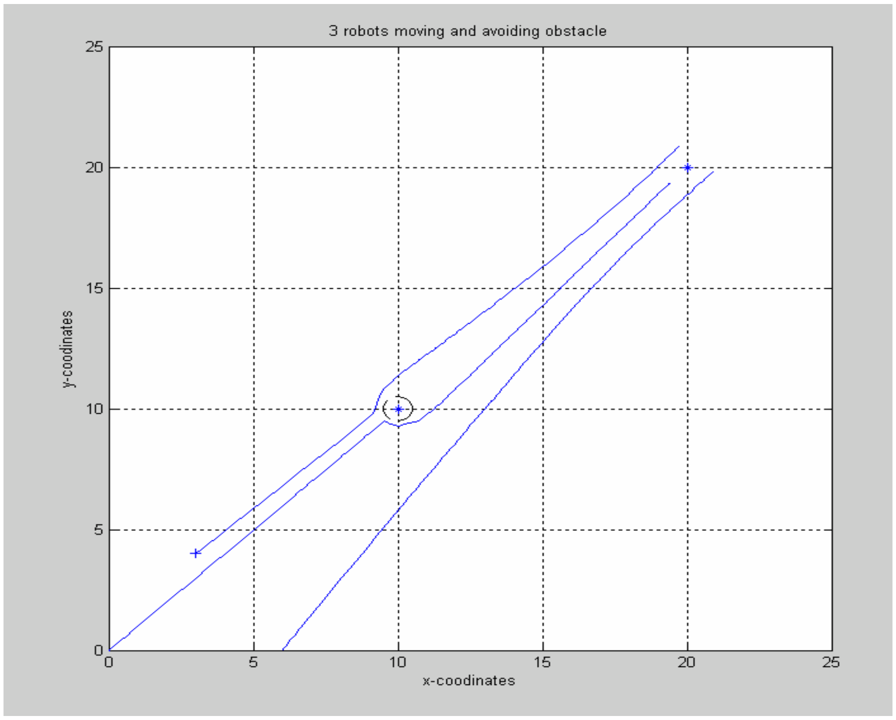


Figure 6: Plot of 3 robots avoiding an obstacle

The plot in Figure 7, from the same sample run, shows the path of the robots with respect to time for this simulation. From the three-variable plot, an observer can easily determine if the robots were moving and colliding at any given time. If two paths of motion of the robots cross, it means that the robots collided. Since the speed of the robots was not limited, the swarm moved rapidly towards the endpoint, completing the motion in less than 5 seconds. The speeds reached in this case were not representative of the speeds that could be reproduced in water.

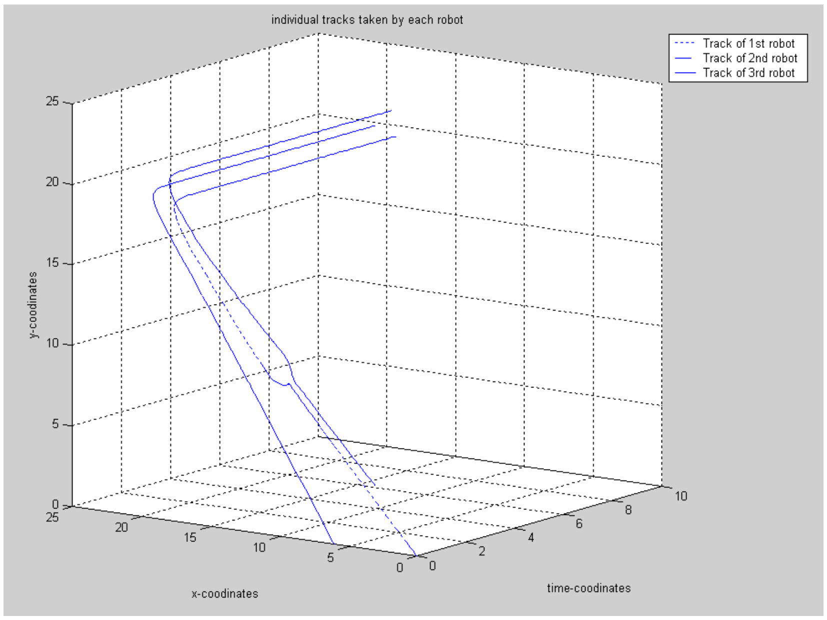


Figure 7: Plot of individual robot tracks without normalized (limited) speeds.

The following two plots, (Figures 8 and 9), show the same structure and behavior, but with speed limited to 1 unit/sec. The reduction of the magnitude has specific real life application to make the speeds executable on real robots.[21] The robots took 44 seconds to reach the final point. This approach allowed the user to change the gain to a suitable value to simulate the speed of an AUV or UUV moving through the water.

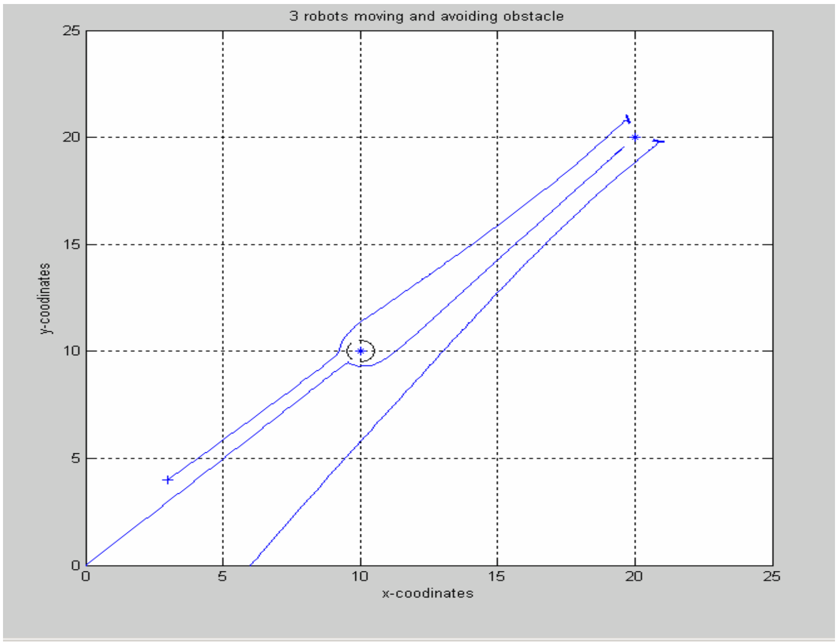


Figure 8: 3 swarming robots avoiding an obstacle with normalized tracks.

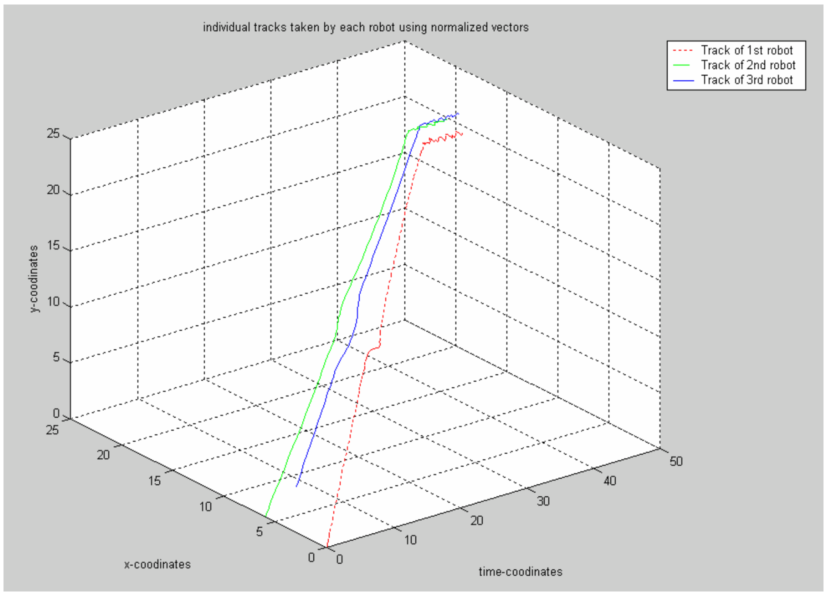


Figure 9: Individual robot tracks with normalized (limited) speed.

3.2.4.4 Sample Run with Multiple Obstacles and Multiple Robots

The next simulation was developed to demonstrate the capabilities of the designed controller in more complex environments and with a larger number of robots. This simulation included numerous obstacles that were made up of multiple circular sub-obstacles. From the sample run in Figure 10, it was observed that the robots were able to avoid the obstacles while moving towards the final point. The plot shows that the attractive and repulsive vectors worked effectively together to generate the desired motion towards the endpoint.

Note that, as the number of robots increases (six in this case), the formation of the robots becomes less apparent. This could be attributed to the change in mean positions, as the robots were moving; hence the robots were not able to form a consistent shape around the center of mass due to broad path changes resulting from obstacle avoidance. The plot of individual tracks in Figure 11 shows that the robots were moving towards the endpoint without colliding with each other. Code for these simulations can be found in Appendix D.

A problem encountered during this particular run was the creation of multiple ‘meta’-obstacles using smaller, circular obstacles. There were only certain shapes that could be created due to the limitation of adding individual obstacle points. The benefit of this approach was that computation of the nearest obstacle point was straightforward. However, as these shapes were not realistic, this problem was addressed in later simulations by creating a more complex obstacle field.

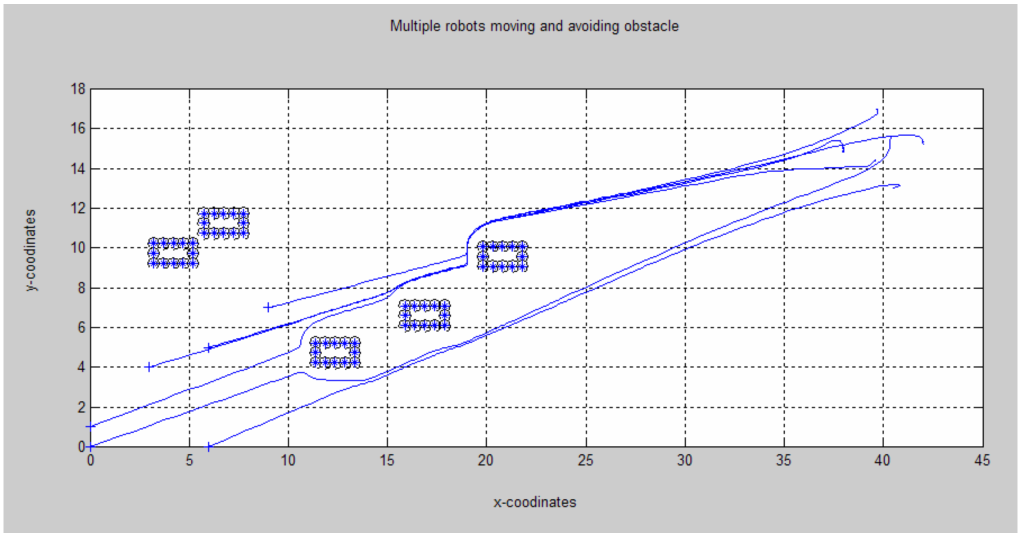


Figure 10: Multiple robots avoiding multiple obstacles

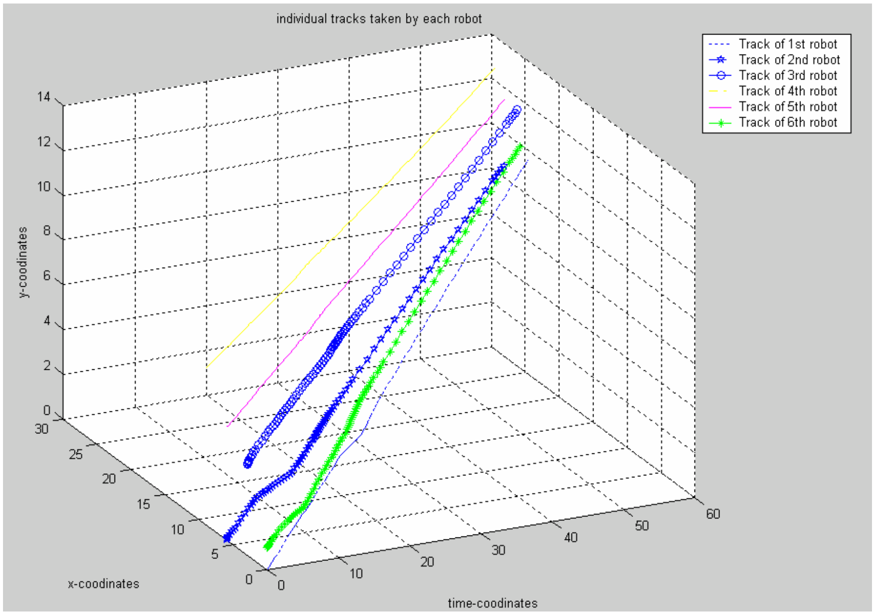


Figure 11: Individual robot tracks avoiding multiple obstacles.

3.2.4.5 Sample Run with Polygonal Obstacles

The next sample run was generated using more sophisticated obstacle modeling, with shapes more representative of the objects the robots might encounter in the real undersea environment. As seen in Figure 12 below, the shapes of the objects resembled those of ships. An assumption made was that the robots would not be searching at great depths but would mainly focus on littoral areas where the mines are more likely to be placed. Hence, in shallow waters, the robots need to account for the hulls of the ships for obstacle avoidance.

Furthermore, with the generated change in obstacles, a new program for creating the repulsive vector was modified and recreated from an existing program.[22] The repulsive vector function was called to determine the closest obstacle point from each robot and to generate a repulsive vector away from the particular edge of the object. From the results of the runs, it was seen that the programs work fairly well. The plot of the individual tracks showed no signs of collision in Figure 13 and demonstrated the effectiveness of the controller in obstacle avoidance. Code for the new obstacle avoidance routine can be found in Appendix E.

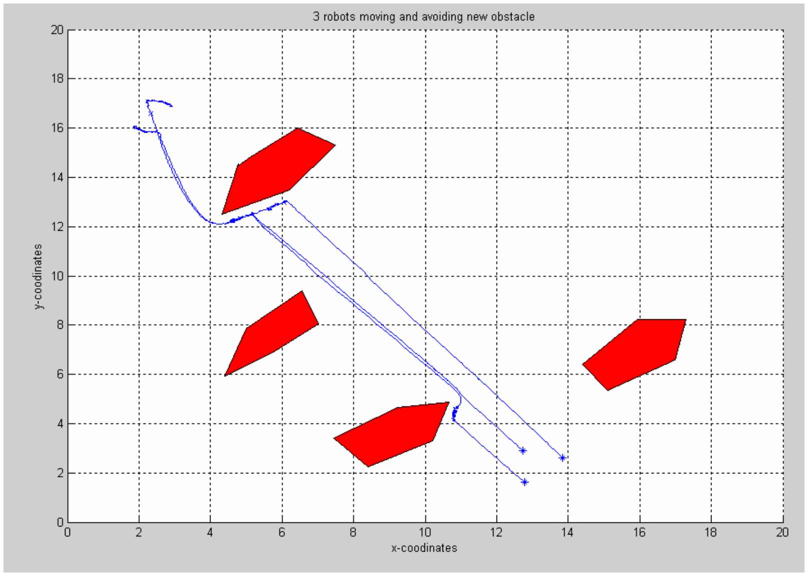


Figure 12: Plot of motor schema with new obstacles.

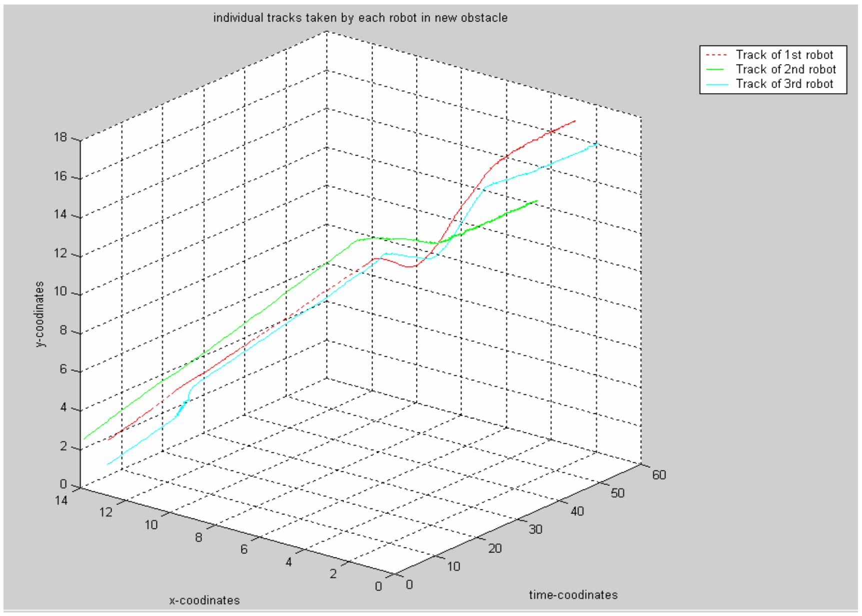


Figure 13: Individual tracks of the robots avoiding new obstacles.

3.2.4.6 Additional Improvements to Motor Schema UMCM

One major limitation of motor schema can be seen in the simulation shown in Figure 14, where the robots collide with an obstacle. The results come from a phenomenon that is known as local minima. [23,24] A robot using APF methods experiences local minima when the attractive and repulsive vectors exactly cancel at some point. This occurred in the simulation when the desired attractive potential was exactly perpendicular to an obstacle edge. One method to remedy this problem is to add a random motion behavior, ‘Noise’, so as to increase the chances of the robot moving out of the constrained region as shown in Figure 15. The new schema, ‘Random_Search’ involves a simple additive random velocity held over a small sample period, and proves effective for small regions of attraction to local minima. More sophisticated methods, involving recorded time histories, are sometimes used in complex problems.[25]

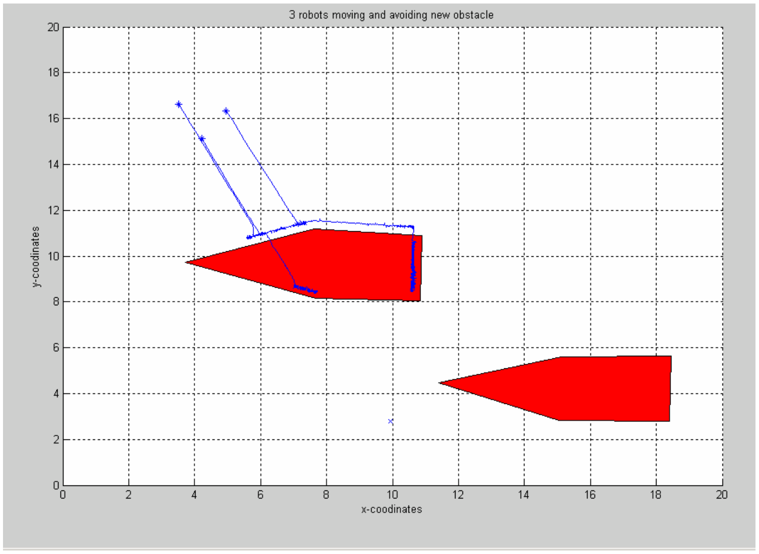


Figure 14: Local minima of robots with associated collision.

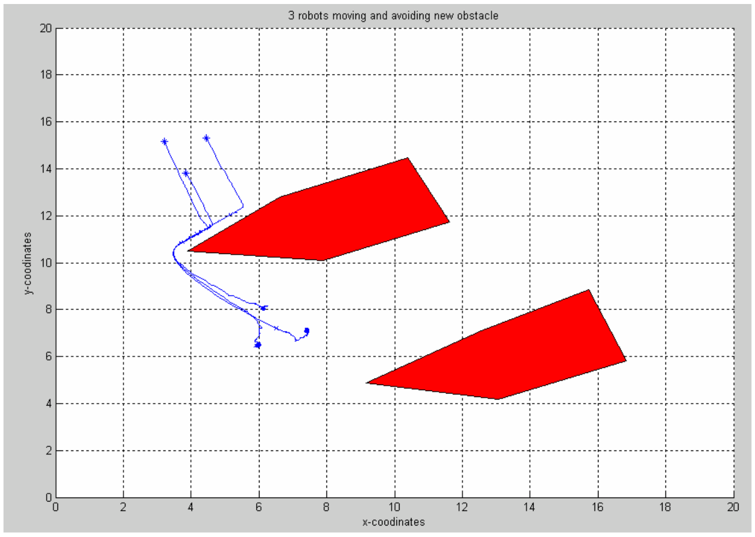


Figure 15: System with added 'Random_search' behavior.

3.2.4.7 Robot Swarms Conducting Underwater Mine Countermeasures

The next simulation that was conducted combined some of the desired effects that have been discussed in the runs mentioned previously. Specifically, this simulation combined ‘Begin_Search’ with ‘Avoid_Obstacle,’ ‘Aggregate_Separate’ and ‘Random_Search.’ The resultant attractive vector pointed towards a weighted combination of the final point and the center of mass while that of the repulsive vector pointed away from the other robots as well as the obstacles, with some random additive values. The obstacles, created to resembled ships, were laid out to loosely represent ships steaming in a narrow stretch of water in an enclosed channel. The robots were also able to repel away from those limits. Furthermore, the robots also tagged the mines as they came close to them, while performing the new behavior 'Avoid_mines'. 'Avoid_Mines' is effectively identical to 'Avoid_Obstacle' with the exception of the additional tagging action. These basic simulations served the purpose of verifying the effectiveness of the program by allowing a qualitative analysis of the performance of the system.

From the results, shown in Figure 16 on the next page, it was observed that the initial positions of the robots formed a straight line. When the robots began to move toward the final position initiated by 'Begin_Search,' individual robots moved to their respective relative positions through the action of the behavioral vector component 'Aggregation_Separation.' The second behavior allowed the robots to position themselves at a reasonable location from the mean position of the respective robots. As the robots moved toward the final position, the distance from between each robot was gradually reduced. 'Avoid_Obstacle,' was also at work, evident from the obstacle avoidance around an object. The 'Avoid_Mine' behavior allowed the robots to avoid the obstacles at the same time tagging the positions of the mines. MATLAB code can be found in Appendix F.

The swarm motion patterns discussed thus far were *ad hoc.* The Parallel Search And Rescue (SAR) pattern shown in Figure 17 was modeled after patterns adopted by the U.S. Coast Guard for finding downed pilots or missing personnel at sea.[26] The axis of search is parallel to the major axis. It is a useful pattern to use when the search target has the possibility of being anywhere along the major axis. Furthermore, fixed wing craft uses this pattern to cover a large area in a short time. The objective of underwater mine clearance by AUVs and UUVs involves clearing a path for ships to pass through. This pattern satisfies the objective by clearing a path or channel quickly while finding all mines and obstacles in the way.

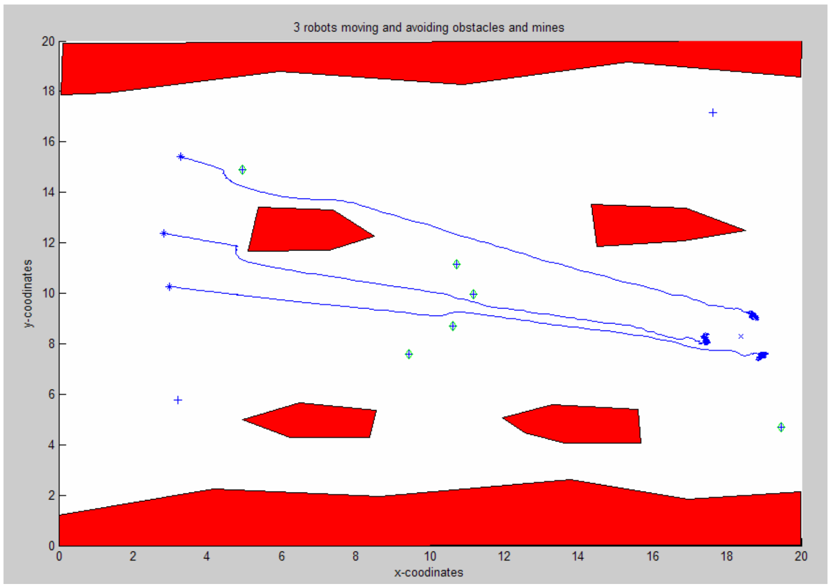


Figure 16: Robot swarm conducting UMCM

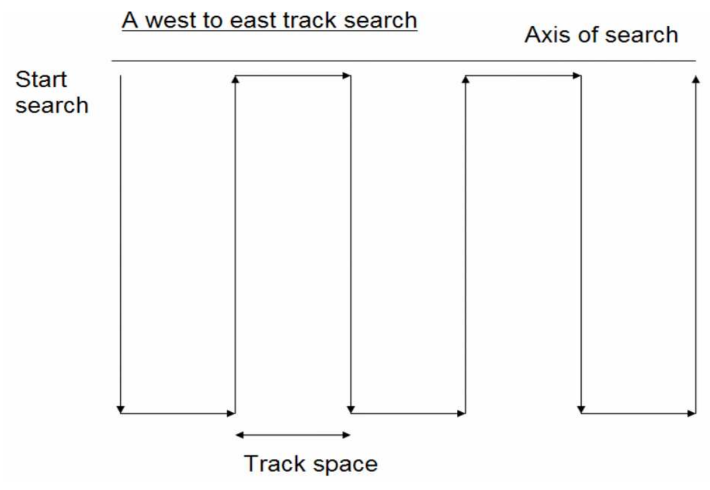


Figure 17: Parallel SAR search pattern

Figure 18 shows the robot swarm executing a search scheme to mark all the mines in the sample minefield. The robots follow specific waypoints that delineate a path of search mimicking the Parallel SAR pattern. The swarm exhibited good swarm maneuver capabilities, avoiding obstacles and mines. However, the robots were not able to knowingly perform tasks according to changes in the environment. The resultant behaviors expressed were generated by a vector sum of numerous behaviors, but the robots had no decision-making capability. Every behavior was considered equally important at all times.

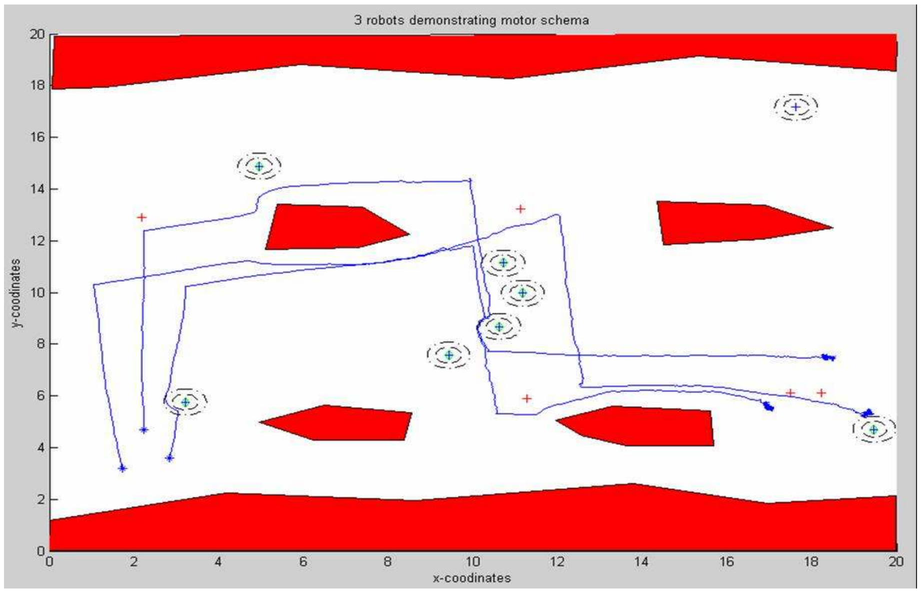


Figure 18: Robot swarm performing UMCM with motor schema

3.3 Conclusions Drawn from Motor Schema

Some advantages of the motor schema were observed from the research conducted. First, the behaviors are executed in real time, and this capability allows the robot to generate behaviors in accordance to specific stimulus in the environment at a particular point in time without a full knowledge of the environment. Second, the potential fields cause the robots to automatically decelerate as they approach a target point, or accelerate away from an obstacle, which reduces the dependence on velocity compensation in real life. Third, the schemas for each individual behavior are very modular in nature. Throughout the process of developing the code, cutting and splicing individual sequences of behaviors was very straightforward. This makes development and addition of future behaviors a rather easy task. Lastly, it is worth noting that the control is still decentralized. Under this control, the robots behave autonomously using available information to make the right decisions. It is important to note, however, that the scope of this research did not deal with how each robot determined the location of obstacles, the actual centroid, or its own position using available sensors.

Several disadvantages of a pure motor schema structure were also observed. The lack of a clear decision structure made the system truly reactive. Placing different behaviors in a hierarchical order would improve the functionality as well as allow certain behaviors to be triggered only under specific circumstances.

This phase of the project resulted in several major accomplishments with regards to the motor schema approaches shown in this section. While the motor schema architecture is well known, no specific methods have been described to generate behaviors that can be matched precisely to such a desired objective. That is, there is no known closed-form design methodology. Nonetheless, the groundwork done to generate appropriate behaviors for the UMCM problem demonstrated that motor schema were viable for UMCM, and representative of certain desirable traits of a mine-hunting robot. The behaviors were created using Artificial Potential Fields since manipulating differential equations could generate the different behaviors. The reduction of behaviors into differential equations made it possible for further development and addition into other viable structures, which will be discussed in later sections.

4. The Subsumption Architecture

Another form of behavior-based controls, the subsumption architecture, was also studied for this research project. The strengths of subsumption architecture make up for the weaknesses present in the motor schema control. It creates a decision hierarchy, whereby a robot is able to make decisions based on the information currently present. However, subsumption is also similar to other behavior-based controllers in that its structure is extremely reactive in nature.

Motor schema, as discussed in Section 3, has no decision hierarchy… each behavior is a component of a composite behavior. The only structure motor schema possesses is the summation of different potential fields contributing to the resultant force on a robot. In a subsumption approach, only one behavior is active at any given time. There are no ‘expected’ behaviors in a subsumption system, since the robots are made to move at run time and the robot has to figure out what to do next, based on the current information. [27]

A possible application of subsumption includes multi-agent controls, which were (in part) investigated in this research. The significant characteristic employed for that purpose was the decentralized nature of the robot control. Some have argued that even in service industries, smaller modular robotic systems have clear advantages over traditional sophisticated ones in terms of costs and susceptibility to damage or loss.[28] To interconnect and relate information between these robots is to use artificial intelligence structures such as subsumption. Several of the direct comparisons between a conventional AI and behavior-based subsumption control can be found in the paper written by Dr Van Dyke Parunak.[29] He addresses the problems of conventional AI methods, including slow response, need for a central computing body, fragile responses to change, and difficulty in reconfiguration. The behavior-based approach, however, represents what happens in reality: fast response due to calculation at run time, robust response to changes, and no need for a central computing agent.

The subsumption architecture builds complexity based on simple behaviors. This form of decision structure overcomes traditional AI problems of integrating numerous sensors and having to respond to multiple task goals. Such AI systems give specific instruction to actuators to respond to the perceived sensory information. An advantage of the subsumption method is not overwhelming the system by taxing the computer performing the computations, but by directing specific reactions for different sensory information.

Another highlight of the subsumption architecture is the use of a behavior hierarchy that coordinates decisions.[30] Two methods of coordination are inhibition and suppression. Inhibition is the prevention of the transmission of a signal, while suppression is the prevention of the transmission of a signal and, at the same time, replacement of that signal with the suppressing signal's information. Figure 19 shows a simple subsumption structure that has been created for the specific purpose of autonomous robots performing UMCM.

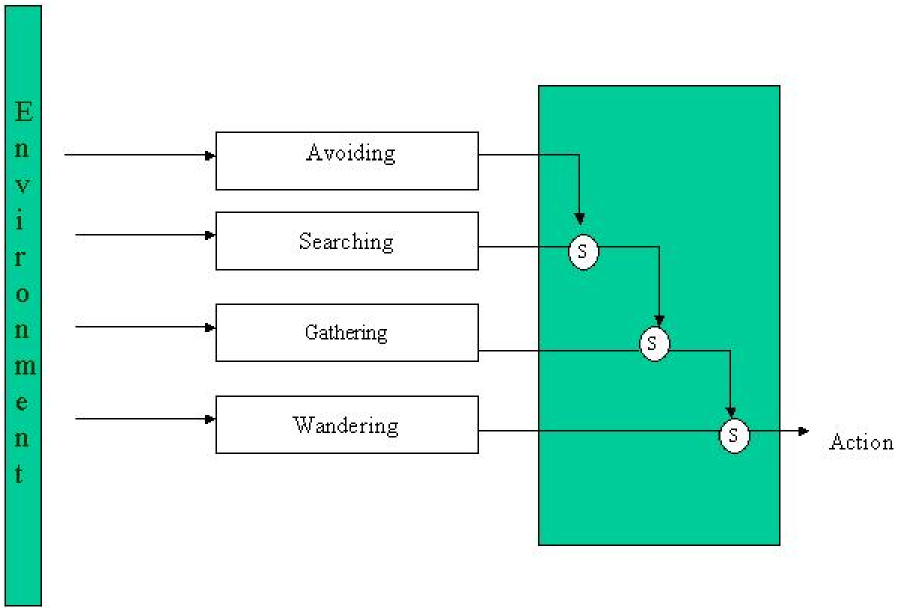


Figure 19: Subsumption architecture of a typical robot.

From Figure 19, the decision structure of a robot is set up such that there is a hierarchy that the robot uses to decide which behavior to implement at any given time. For instance, the highest priority is to avoid an obstacle, followed by searching for objects, gathering with other robots, and finally wandering around. These behaviors are dictated by the environmental stimulus. The circles with a letter, s, represent the suppression mechanism at work. If the sensors for a higher ranked behavior are fired, the suppressor will prevent all behaviors from the lower ranks from being expressed. This allows the current behavior to be the only one expressed at that point in time. Once those sensors stop detecting the stimulus, the robot reverts back to the next lower behavior whose sensors pick up the next most important stimulus. Again, the result of the subsumption architecture allows only one behavior to be expressed at a time; all other lower behaviors are masked.

4.1 Previous Work

Rodney Brooks from Massachusetts Institute of Technology (MIT) originally developed the subsumption architecture. He proposed that different layers of behaviors could be set in a hierarchy based on rules for performing specific tasks. This form of decision structure was inspired by biology, as he realized that the actions of a robot could be separated and artificial intelligence could emerge to build complex interactions with the environment. Several important aspects that Dr Brooks focused on were the facts that the robots did not need an internal model as was common in conventional methods, since the new architecture was able to deal with imperfections from the physical world and did not rely on predicting or expecting information that might not be available or easily modeled.[31]

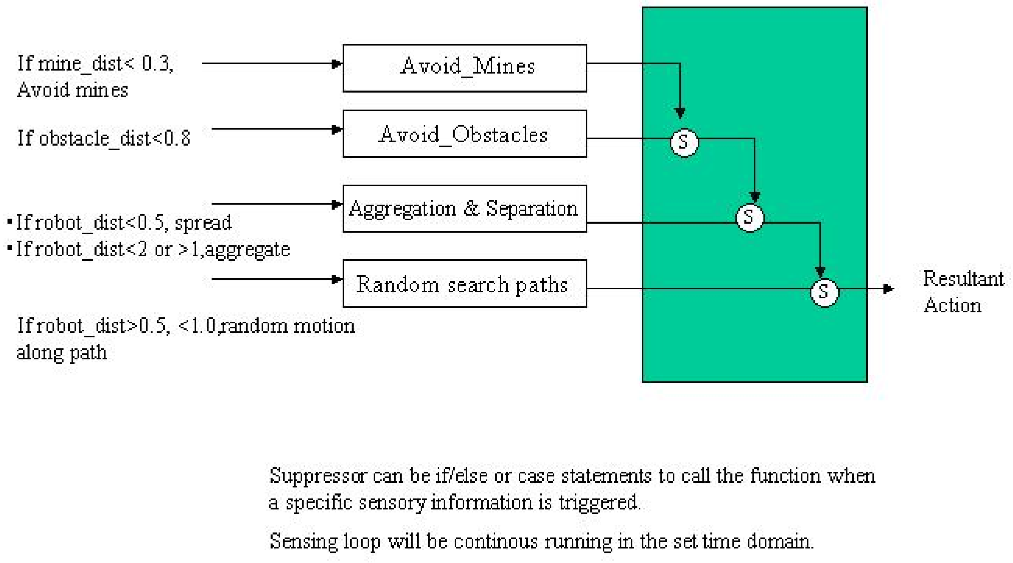
4.2 Relevance of Subsumption to UMCM

The study of the subsumption architecture was critical in generating a controller that can be easily embedded into hardware controls, because of the presence of a rigid decision structure. However, as the architecture does not specify how to generate individual behaviors, the development of subsumption controls relied in the initial stages on generation of suitable behaviors using a basis of APFs and motor schema (see Section 3). Even though the basic building blocks are motor schema, the subsumption structure proved more flexible and robust as a controller (due to the decision capabilities). Furthermore, embedding the motor schema behaviors into the subsumption architecture created a more sophisticated behavior-based system that duplicated the best characteristics of motor schema (when only one behavior was active) while adding new capabilities. Finally, the controller still relied on behaviors that could be modeled with differential equations, which was an important characteristic that was desired for purposes of further hybridization, as will be discussed in Section 6.

The types of controls dramatically change from continuous in motor schema to a switched controller in subsumption. Nevertheless, the robots were able to generate highly reactive behaviors and performances that were not possible without these forms of control. Therefore the robots were able to perform their tasks in an organized manner yet with total unpredictability. These mine hunting robots are meant to be autonomous, hence the decision structures were designed to be self-governing to allow the robots free reign based on the environmental factors at that present moment.

4.3 Subsumption Architecture for UMCM

Figure 20 shows the developed UMCM subsumption structure, which consists of the various behaviors as created from the programs in motor schema, 'Avoid_Mines,' 'Avoid_Obstacles,' and 'Aggregation_Separation' as well as the 'Random_Search' behavior (used in a different context here). As shown above, behaviors were activated if the individual distance calculated exceeded the requirements, suppressing lower behaviors.



If mine_dist< 0.3, Avoid mines
If obstacle_dist<0.8
▪If robot_dist<0.5, spread
▪If robot_dist<2 or >1,aggregate
If robot_dist>0.5, <1.0,random motion along path
Avoid_Mines
Avoid_Obstacles
Aggregation & Separation
Random search paths
S
S
S
Resultant Action
Suppressor can be if/else or case statements to call the function when a specific sensory information is triggered.
Sensing loop will be continous running in the set time domain.

Figure 20: Subsumption architecture of a mine-hunting robot.

Pseudocode below shows the basic algorithmic flow of the subsumption architecture.

```
For (number of robots)
    (1 robot at a time)
    If (distance of mine < d_Mine)                %Avoid_mine
            •  Tag closest mine
            •  Repulse away from the mine
    Else
        If (dist to obstacles < d_Obst)
                •  Repel from obstacles
        Else
            If (distance to robots> d_x)          %Avoid_Obstacles
                •  Do nothing
            Elseif (distance to robots>d_y & distance to robots< d_x)      % Aggregation
                •  Find centroid of robots
                •  Attractive vector towards common center
            Elseif (distance to robots<d_y & distance to robots> 0)
                •  Repel away from other robots
            Else        %Random Search
                •  Random search for mine
                •  Follow pre-plotted waypoints
```

From this pseudocode, the MATLAB simulation code was developed. It used simple if-else statements to act as suppressive agents to create the hierarchy of the behaviors. Furthermore, the presence of limits to activate certain behaviors was an accurate way to represent the information being detected by sensors. If the level of the sensory information exceeded that of the threshold value, the behavior would fire, at the same time suppressing all other behaviors. The various codes that expressed behaviors used a form very similar to those from the motor schema discussion (Section 3). MATLAB code can be found in Appendix F.

Several problems were encountered in coding the subsumption structure. The first was finding a suitable suppressor element. After much contemplation, an if-else statement was the simplest, most direct manner in which the suppressor could be implemented in code. Furthermore, it proved a difficult task to combine various lines of codes from different behaviors and link them to produce a combined, autonomous effect in an organized structure. However, the use of velocity-based control vectors helped facilitate the application of the motor schema behaviors to the creation of the subsumption architecture.

In fact, it is worth mentioning at this point that traditional behavior-based systems do not lend themselves to simulation. Primary difficulties include the fact that most such architectures do not involve equations of motion, but simply connect sensors to motors and allow the behavior to ‘emerge.’ As such, it was necessary to develop velocity controllers based on APFs and motor schema methods as a first step to building a subsumption architecture, as these methods can be written to provide direct motion commands. Even so, simulation of switched systems, such as subsumption-based methods, was nontrivial and required some care. As will be seen in the following section, it is this careful, systematic simulation development that led to the major breakthrough of this work.

4.3.1 Subsumption Simulations

The subsumption code for running a swarm of robots was modeled after the pseudocode from the previous section (see Appendix G). In the background, at points where the robots appear to change direction, lie the waypoints. The generation of such waypoints allowed the user to generate a certain pattern of search for the mines using the robots (see Section 3.2.4.8). As the robots approached the waypoints, the reduction in distance between the individual robots and the waypoint was calculated and compared to a set limit. If the distance was less than the given limit, then the waypoint was switched to the next one, allowing the robots to change waypoints as they moved along the designated path.

The results of the simulation are given in Figure 21. As the robots moved along the intended path, they tagged the mines (originally marked with +) with a diamond to indicate that the position of the mine had been located (sensing radius for mine labeling was 1 unit). Ideally, this information would be reported to a central unit to assess the threat of mines. The plot in Figure 22 shows that the robots did not collide during the run.

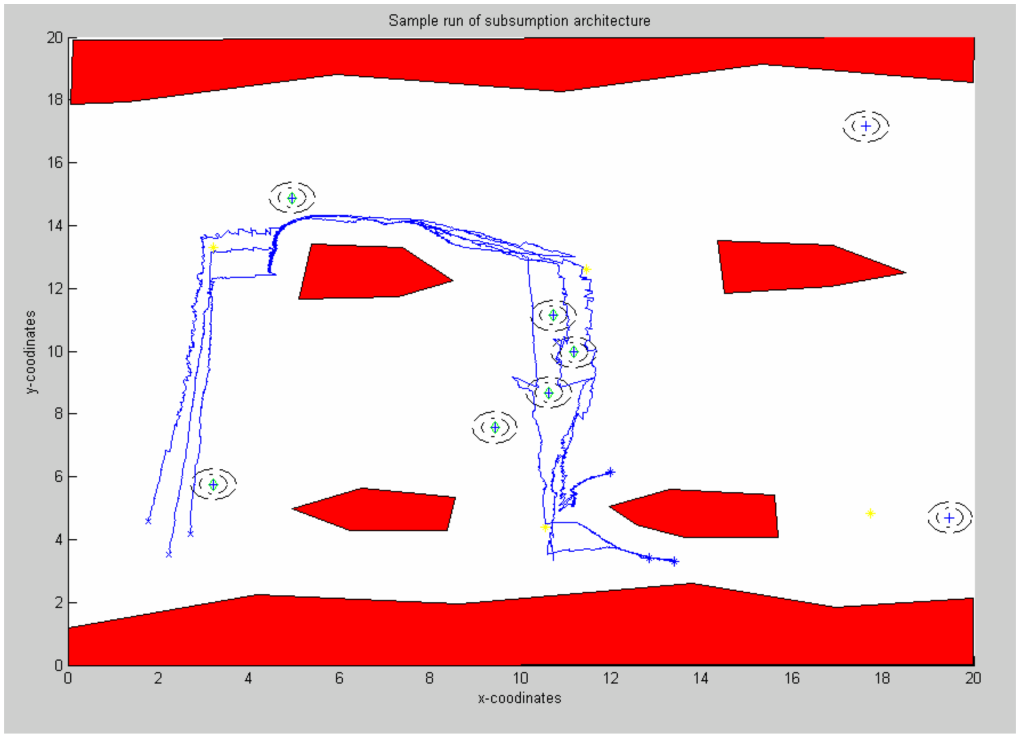


Figure 21: 3 robots performing UMCM under subsumption architecture

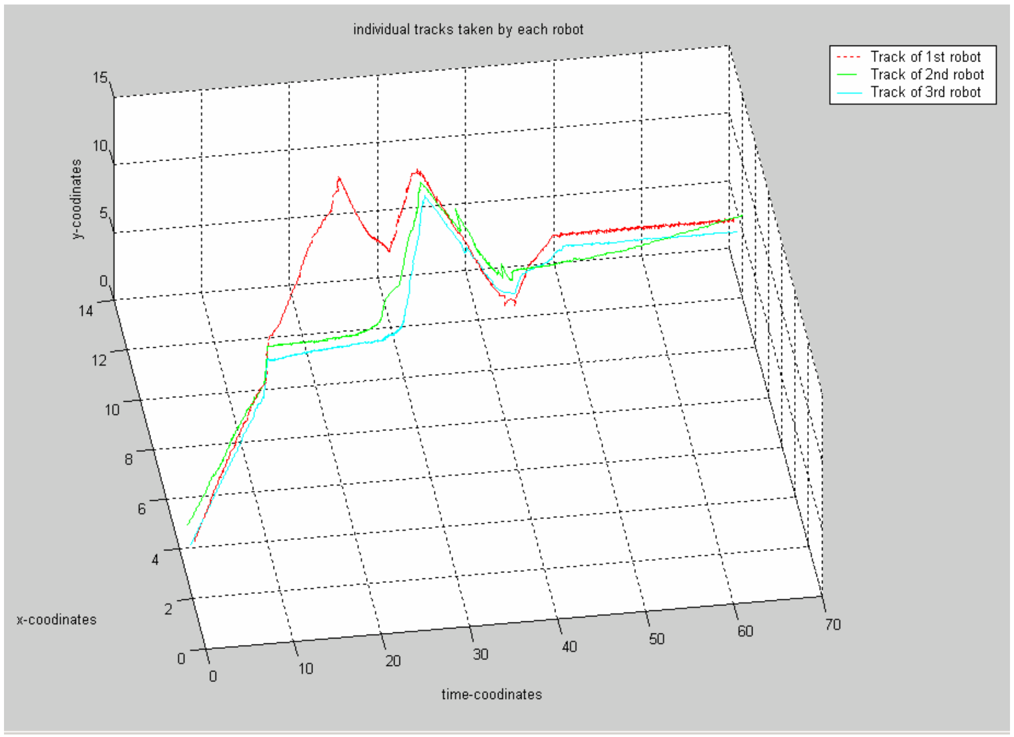


Figure 22: Plot of individual tracks for subsumption architecture

In these simulations, mine avoidance was of utmost priority, since a robot could be destroyed if it ventured too close to a suspected mine. This behavior was adopted from motor schema behavior, ‘Avoid_Mines.’ To limit the types of behaviors expressed when the robots moved close to the mines, different sizes of sensing circles were created. A circle of radius 0.3 units around each mine simulated the region of sensitivity of the robots to the mines. A significant characteristic of this behavior was embedded hysteresis. If the robot was inside the inner radius, ‘Avoid_Mines’ was triggered. That behavior remained active until the robot moved outside a larger, outer radius of 0.5 units. The zigzag and jagged response seen in Figure 21 exemplifies hysteresis. Once the robot moved out of outer circle, the robot would activate the next higher behavior and continue with its movement.

The second most important behavior was to make the robots avoid the obstacles. This was achieved by adopting the behavior, ‘Avoid_Obstacle.’ The robots had a separation distance of 0.8 units to allow the robots to clear the obstacles. The observations from the code showed that some randomization was important because it prevented the occurrence of local minima of the potential fields theory found when a robot moved around an obstacle. Even so, the randomization was limited because the robot had to follow certain waypoints and their individual paths were generally moving towards the endpoints. Since only one behavior was expressed at any one time, random motion for avoiding local minima was taken into consideration within each individual behavior. Therefore, the robots were able to generate random motion under the ‘Random_Motion’ behavior as well as the ‘Avoid_Obstacle’ and ‘Avoid_Mine’ behaviors. This was easily accomplished due to the underlying motor schema structure.

The third significant behavior was the need to aggregate toward a common center while avoiding each other robot. This behavior was adopted from the motor schema

‘Aggregate_Separate.’ When the distance between robots was between 1 to 2 units, the robots were too far away from each other. Hence the robots would aggregate, or move towards each other to reduce the separation. If the separation was greater than 3 units, the behavior was disregarded. This allowed the robots to peel off from the main group while performing ‘Avoid_Obstacle’. Thus, if a robot moved in the opposite direction from the rest of the swarm around an obstacle, then the robot would be allowed to move randomly and perform ‘Random_Search’ in a new area. Furthermore, this behavior allowed the robots to have some freedom to move around an obstacle individually without having to sacrifice the mobility of the swarm to avoid the obstacle.

The aggregation behavior was supplemented with a separation component. If the distance between a given robot and its nearest neighbor was less than 0.5 units, the robots would repel from each other to increase the separation distance.

Due to the limits on triggering of the ‘Aggregate_Separate’ behavior, only when specific values of the separation distance were met would the behaviors be triggered. As expected, this resulted in inconsistent group aggregate shapes and formations.

The last behavior was ‘Random_Search.’ This behavior occurred when the robot was not performing any of the other behaviors. The robot would move, in general, towards a pre-assigned waypoint. As it approached the waypoint, the movement was interjected with a lot of noise, to enable the robot to move in a more random manner. When the distance to the current waypoint became less than 0.1 units, the next waypoint was initiated and the robot swarm proceeded to the next waypoint.

4.4 Conclusions Drawn from Subsumption Architectures

Several conclusions can be drawn about the performance of the subsumption architecture. The subsumption architecture is a reactive system that allows the individual robots to behave autonomously, using their own decision structure to pick the correct behavior to suit their movement and the environment at a particular time. Furthermore, the subsumption approach may reduce the computational strain on a microprocessor as a result of the lack of computational complexity rising from a single point behavior at every instance in time. The suppressor agent, in this case, the if-else statement, prevents any lower behavior from interacting or interfering with the behaviors higher up in the hierarchy. Such controllers are easy to implement in actual hardware, since there is a specific structure that the robot can base its action on instead of relying on a diffused schema, where multiple behaviors are added together with a ‘hope for the best’ attitude. The subsumption architecture implicitly creates Artificial Intelligence (AI), since the architecture makes decisions based on environmental feedback without a need for planning. The presence of the suppressor adds to that effect with the fact that it generates a yes/no response on different types of input, implicitly avoiding the sensor fusion problem (wherein difficulties commonly arise when combining data from disparate sensors for a unified controller).

Some problems encountered with the subsumption approach included difficulty in finding a particular structure suitable for UMCM as well as having indeterminate overall systems behaviors. There were no set architectures or design methodologies for how robots performing UMCM should be created. The process of creating the architecture required more art than science, since there was no one mathematical standard to generate that decision structure. Hence, it took creative energy to generate a functional architecture.

The indeterminate systems behavior arose from the fact that the robots' responses to the environment were highly reactive based on the information present at a specific point in time. This unpredictability could be a disadvantage in real operations, since the operators would not have complete understanding or control of the robot swarm. Additionally, it had been seen that similar architectures used for modeling birds in flight and schools of fish suffer from chaotic instability that is impossible to predict but simple to achieve.[32] As such, the next facet of the research was to investigate more systematic swarm control methods that admit some closed-form analysis.

5. Statistical Control

As mentioned, behavior-based and system-theoretic approaches to swarm control have been studied extensively[33]. Behavior-based systems are flexible, easy to implement and do not require a complete model or knowledge of the environment. System-theoretic methods are commonly based on differential equations. Such systems have provable results with well-understood analysis tools, and are sometimes controllable through the application of compensators to give results that match desired specifications.[34] The next phase of this project focused on a systems-theoretic controller that was very promising for swarm control. The controller selected was the so-called ‘statistical controller.’ Statistical controllers are designed to manipulate the individual robots in a swarm to provably and robustly generate a desired swarm profile and yet still allow the units some degree of autonomy.[35]

5.1 Introduction to Statistical Control

Statistical swarm control was investigated to determine its suitability for UMCM. Under this systems-theoretic control methodology, swarm-level functions (such as overall mean position and variance) can be controlled in a provable manner. Such swarm-level functions dictate how a group of robots coordinates individual unit motion as the platoon transitions from point A to point B.

Statistical control methods, a form of systems-theoretic control, have desirable characteristics for the UMCM problem. Again, systems-theoretic control systems such as this are typically based on differential equations, which make them well suited for simulation and real implementation. Statistical controls can be run in real time, giving rise to fairly reactive control by limiting the spread of the robots while at the same time allowing the robots to interact and move with some autonomy. Weaknesses of this method include rigidity in structure, necessity for pre-planning of swarm paths and tasks, and an overall predictability (which is an advantage for stability but a disadvantage for reactive searching). The statistical control architecture is well suited to simulation study, since the mean and variance of the spread of the positions of the swarm of robots can be observed and calculated directly from the simulation with ease.

5.2 Applications to UMCM

Statistical methods have the capability to coordinate the motions of each robot in a swarm without having to plan their individual paths and motions in advance (which is a computationally complex problem for large swarms). These controllers also allow certain secondary tasks to be encoded in such a way as to be carried out only when not interfering with the primary (swarm-level) tasks. These characteristics are truly beneficial to UMCM. The robots are free to roam or move in a designated area while maintaining a particular distribution limited by restrictions placed on the swarm. In addition, these controllers admit simple reconfiguration of primary and secondary tasks, which enhances the multi-tasking capability of the robot swarm.

5.3 Previous Work

Professor Bishop from the United States Naval Academy pioneered the theory of statistical controller. He has since worked with Professor Stilwell of Virginia Tech. They have developed a comprehensive statistical controller for swarming robots, but have not developed any task-specific implementations, such as for UMCM. In the next few sections, the basics of this control are discussed, followed by a design for the UMCM problem.

5.4 Fundamentals of Statistical Control

Statistical methods control swarm-level functions, such as mean and variance of the positions of the robots, which in turn restrict the movements of a swarm as a whole.[36] The individual task objective of each robot remains unchanged… determining the location of the mines and avoiding obstacles. The swarm, as a collective entity, obeys some desired function profile by coordinating the motion of the robots.

This approach involves using a special matrix, known as a Jacobian, that allows the calculation of the best velocity profile for the robots according to the set limits of the performance of the swarm. Jacobian matrices are typically applied in traditional robotics to relate joint velocity to end-effector (tool) velocity.[37] A Jacobian matrix is defined by the state q of the system and some task function $f(q)$. Denoted $J(q)$, the Jacobian is given by:

$$J(q) = \begin{bmatrix} \frac{\partial f_1(q)}{\partial q_1} & \cdot & \cdot & \frac{\partial f_1(q)}{\partial q_n} \\ & & \cdot & \\ & & \cdot & \\ \frac{\partial f_m(q)}{\partial q_1} & \cdot & \cdot & \frac{\partial f_m(q)}{\partial q_n} \end{bmatrix} \qquad (1)$$

where the state q is n dimensional, the task function $f(q)$ is m dimensional, and $\dot{f}(q) = J(q)\dot{q}$.[38]

In a statistical controller, $f(q)$ represents the swarm task function, which can be any closed-form, differentiable function of the swarm state, which is typically given by the concatenation of the individual units' positions. In this work, the state vector for an r unit swarm in 2D was taken to be $q = [x_1, x_2, \dots, x_r, y_1, y_2, \dots, y_r]^T$. An example of a task function $f(q)$, using swarm mean and variance, is given by[39]:

$$f(q) = \begin{bmatrix} \mu_x \\ \sigma_x^2 \\ \mu_y \\ \sigma_y^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{r}\sum_{i=1}^{r} q_i \\ \frac{1}{r-2}\sum_{i=1}^{r}(q_i - \mu_x)^2 \\ \frac{1}{r}\sum_{i=r+1}^{2r} q_i \\ \frac{1}{r-2}\sum_{i=r+1}^{2r}(q_i - \mu_y)^2 \end{bmatrix} \qquad (2)$$

In this simple example, the mean (μ_x, μ_y) effectively determines the overall swarm position while the variance (σ_x^2 , σ_y^2) dictates the spread of the elements. The Jacobian for the task vector given by (2) is shown in (3), (4) and (5)[40]:

$$J = [\begin{matrix} J_x & 0 \\ 0 & J_y \end{matrix}] \qquad (3)$$

$$J_x = \begin{bmatrix} \frac{1}{r} & . \ . \ . & \frac{1}{r} \\ \frac{2}{r-1}(q_1 - \mu_x) & . \ . \ . & \frac{2}{r-1}(q_r - \mu_x) \end{bmatrix} \qquad (4)$$

$$J_y = \begin{bmatrix} \frac{1}{r} & . \ . \ . & \frac{1}{r} \\ \frac{2}{r-1}(q_{r+1} - \mu_x) & . \ . \ . & \frac{2}{r-1}(q_{2r} - \mu_x) \end{bmatrix} \qquad (5)$$

5.5. Components of the Statistical Controller

The statistical controller as seen in Figure 23 is fundamentally a basic closed-loop feedback control system. Using standard methods from feedback control allows the system to reach the desired values of the swarm functions *f*(*q*) at equilibrium. The proportional (gain) controller brings the swarm to the desired *f*(*q*) while the null space controller (discussed below) moves the robots in such a way that there are no collisions.

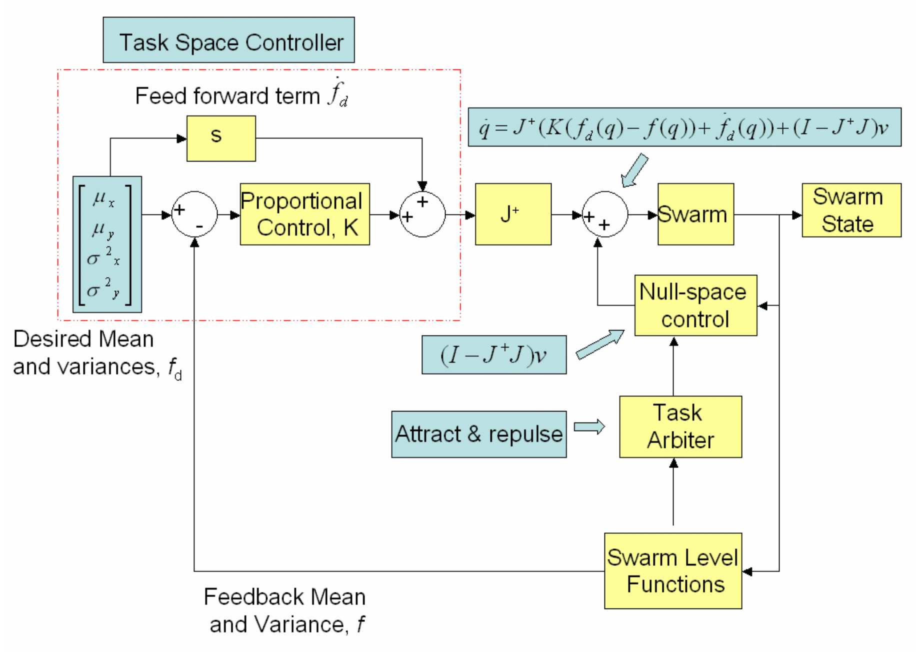

Task Space Controller
Feed forward term $\dot{f}_d$
s
$\begin{bmatrix}\mu_x \\ \mu_y \\ \sigma^2_x \\ \sigma^2_y\end{bmatrix}$
+
-
Proportional Control, K
+
+
J+
$\dot{q} = J^+(K(f_d(q) - f(q)) + \dot{f}_d(q)) + (I - J^+J)v$
+
+
Swarm
Swarm State
Null-space control
$(I - J^+J)v$
Desired Mean and variances, f_d
Attract & repulse
Task Arbiter
Swarm Level Functions
Feedback Mean and Variance, f

Figure 23: Diagram of a statistical controller

The *f*(*q*) that was used in the simulations performed for this part of the research included the mean and variance of the swarm in the *x* and *y* direction, as indicated in (2). The task-space controller and the Jacobian pseudoinverse continually computed the error and compensated the system so that the swarm followed a preferred path, moving along different waypoints defined by the user. The swarm was ordered to move from point to point while each robot performed its mission of searching for mines and avoiding both mines and obstacles. Each facet of the control will be described in the subsequent section.

5.5.1 Task-Space Control

Task-space control is a common method for systems that rely on Jacobians to convert from one coordinate space to another. A controller is designed in the space defined by the task (here, the mean and variance of the platoon). This control generates desired velocities in the task space (here, $\dot{\mu}_x, \dot{\mu}_y$, etc.) The Jacobian is then used to convert those desired task-space velocities to the state space of the system (here, the velocities of each swarm unit).

A proportional controller was used for the task-space component of this system because it is a basic compensator intended to track a desired differentiable trajectory, and because it was easy to implement in the system. The controller uses a gain value, K, to magnify the error between the desired and actual swarm state and to compensate for that error. The compensated system shows improved stability over behavior-based approaches, as it measures error and forces the response to the desired equilibrium. This basic gain controller is described by (6). Figure 24 depicts the controller in a standard closed feedback loop.

$$C=K^*\ e\ (k) \qquad (6)$$

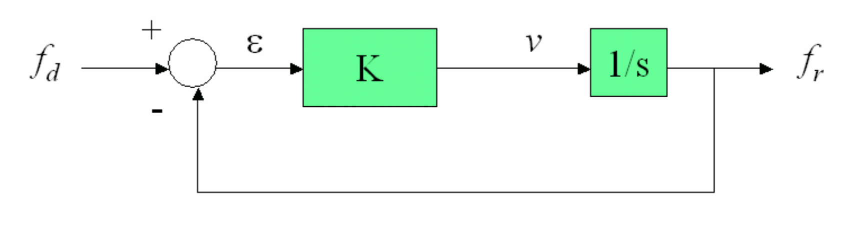


Figure 24: Proportional gain control

In the simulations performed, e(k) was the error calculated between the desired and the actual values of the swarm level functions after every iteration of the control code. K was the control gain that was used to adjust the transient performance of the controller. The error feedback and the feedforward velocity $\dot{f}^d(q)$ (based on planned swarm trajectories) guaranteed perfect tracking of the desired swarm profile in the absence of disturbances (which are addressed in section 6.2.1).

5.5.2 Jacobian Pseudoinverse

The task space controller computes desired velocities for the platoon, but the actual control occurs at the unit level, in state space. The swarm state-space velocity $\dot{q}$ represents the motion of the swarm at every instant in time. For a large swarm of robots, the number of state variables (*x* and *y* for each robot) is typically much greater than the number of task variables (swarm mean position and variance in each dimension, for example). This redundancy creates an infinite number of possible configurations for the swarm while still achieving the desired profile.[41] The basic control that achieves this task takes the following form:

$$\dot{q} = J^{+}(K(f^{d}(q) - f(q)) + \dot{f}^{d}(q)) \qquad (7)$$

$$J^{+} = J^{T}(JJ^{T})^{-1} \qquad (8)$$

where the parenthetical component of (7) is the task-space controller and J^{+} is the so-called Moore-Penrose pseudoinverse of the Jacobian, which is defined by (1) and (2) for this problem. The pseudoinverse is used because the Jacobian for a large swarm is non-square and thus not directly invertible. The application of the pseudoinverse of the Jacobian allows the task-space controller to be converted to a state-space controller, generating motion commands for the units based on the task-space error.

5.5.3 Null Space Control

Since the proposed solution for the problem of UMCM is to use numerous robots to maximize capability, the swarm will inherit redundancy qualities. The swarm has more members than objective functions that need to be met. Therefore, there are more capabilities present than would be utilized by the pseudoinverse controller of (7)-(8). Effectively, there are an infinite number of configurations for the units to satisfy the desired task-space functions.

Redundancy resolution, taken from redundant manipulator control, is employed to manage swarm resources and functions by moving the swarm along the *self-motion manifold* (the set of all configurations that match the desired $f(q)$) toward a locally desirable configuration. In this implementation, additional tasks, beyond the coded functions of $f(q)$, are projected onto the null space of the Jacobian $J(q)$, meaning that those tasks are carried out in such a manner as to leave the primary objectives encoded in $f(q)$ unchanged.

For a UMCM problem, primary tasks are maintaining the mean and variance while the secondary objectives are to avoid obstacles and mines. A Jacobian null space projection term $(I-J^{+}J)v$ basically allows such a swarm to compute appropriate velocities of the robots for achieving secondary objectives given by velocity vectors v while maintaining the primary tasks defined by $f(q)$. Although this does not make each robot fully autonomous, individual units may react to the environment in a constrained manner. In simple terms, when one robot must react to an environmental stimulus and therefore be deflected from the nominal velocity provided by (7)-(8), the entire swarm reacts as a whole to guarantee that the primary task $f(q)$ is still achieved. In certain cases, the response of the units is limited by the primary tasks. On the other hand, the swarm as a whole acts as an autonomous system.[42] It is possible to decentralize certain tasks under this form of control, making the units fully autonomous, although the distinction can be misleading.[43]

Gradient projection methods are employed to achieve the null space control. A redundancy-resolution controller is shown in (9), where $(I - J^{+}J)$ is the projection term. This projection operator can be thought of as a driver that forces the robot swarm to coordinate motion to accommodate secondary objectives.[44] The secondary objectives are encoded in the vector v, which is often a gradient of an objective function c(q) given by $\partial c(q)/\partial q$. The secondary objectives as encoded in *v* are always state-space velocities, although they may arise from the gradient of a task vector.

$$\dot{q} = J^{+}(K(f_d(q) - f(q)) + \dot{f}_d(q)) + (I - J^{+}J)v \qquad (9)$$

5.5.4 Task Arbiter

Specific secondary tasks for the null-space control are embedded in the functional block labeled *Task Arbiter* so that the robots are able to perform some secondary tasks while moving as a swarm. The design of the *Task Arbiter*, which represents *v*, required that the tasks be written in differential equation form for interface with the null-space projection (which requires velocity vectors). An important note to make at this point is that the velocity-based secondary task can be anything that is in the form of differential equations that describe tasks that are also velocity based.

The only secondary task defined for this controller was obstacle avoidance. The obstacle avoidance task was encoded in exactly the same form as the 'Avoid_Obstacle' behavior from Sections 3 and 4, with the exception that the repulsive vectors for every robot were concatenated to form *v*.

In conclusion, the combination of the different components gave rise to a statistical controller that was capable of regulating swarm-level functions while simultaneously accomplishing secondary tasks defined as state-space velocity terms. These secondary tasks proved to be the key to combining systems-theoretic and behavior-based controllers.

5.6 Statistical Simulations

The previous section on the fundamentals of statistical control discussed the theory on which the swarm controller was based. This section demonstrates the implementation of control characteristics for UMCM.

The robot swarm in these simulations used waypoints to guide its movement, moving in a specified path designated by the user. Given desired points and time intervals, the planned path trajectories were projected based on the cubic polynomial of the time in each task coordinate, μ_x, μ_y, σ_x^2, σ_y^2. The initial starting point and the next waypoint were used to derive the initial trajectories. The mean position of the robot swarm was required to fall along the planned trajectory to prove that the task space controller was able to keep the mean and variance of the system.

As the swarm moved along the designated path, singularities in the Jacobian, (3), did not occur. A singularity for this controller could only occur when the units form a straight line along the *x* or *y* direction, which cannot occur unless the commanded variance in one direction is set to zero.

The primary task of the statistical controller performing UMCM was to control the task variables, which were the mean and variance of the swarm. The secondary task in this simulation of the system was purely obstacle avoidance. These robots were meant to mimic the motions of actual AUVs and UUVs, and the most fundamental action was avoiding obstacles at all costs.

The simulations for the statistical method were written without showing extensive obstacles unlike the simulations in the behavior-based simulations because of the computational complexity of the calculations and the difficulty in simulating switched controllers of this sort. For simplicity, the obstacles were shown as circular objects.

5.6.1 Simulation Results of Statistical Controller

One of the first considerations for running the simulation was to determine the number of robots simulated. The swarm of robots was required to satisfy 4 task variables, the mean and variance of the positions in the *x* and *y*-plane. The minimum number of units for redundancy was therefore three (each unit possessing, in our simplified environment, two DOF). The two redundant degrees of freedom enabled the robots to move with some flexibility, but it did not allow them to avoid obstacles or mines easily. Obstacle or mine avoidance demanded more degrees of freedom from the system to allow the robot swarm to react and adapt to the changing environment. Therefore, the number of robots was raised to six, providing eight degrees of redundancy and very good swarm flexibility.

Three waypoints were set in this simulation to allow the robots to move from an initial point to a final point. The robot swarm also displayed its ability to avoid obstacles and move towards targets as instructed in the task arbiter. The null-space controller compensated and counterbalanced the movements of the robots to allow them to adhere to the set mean and variances.

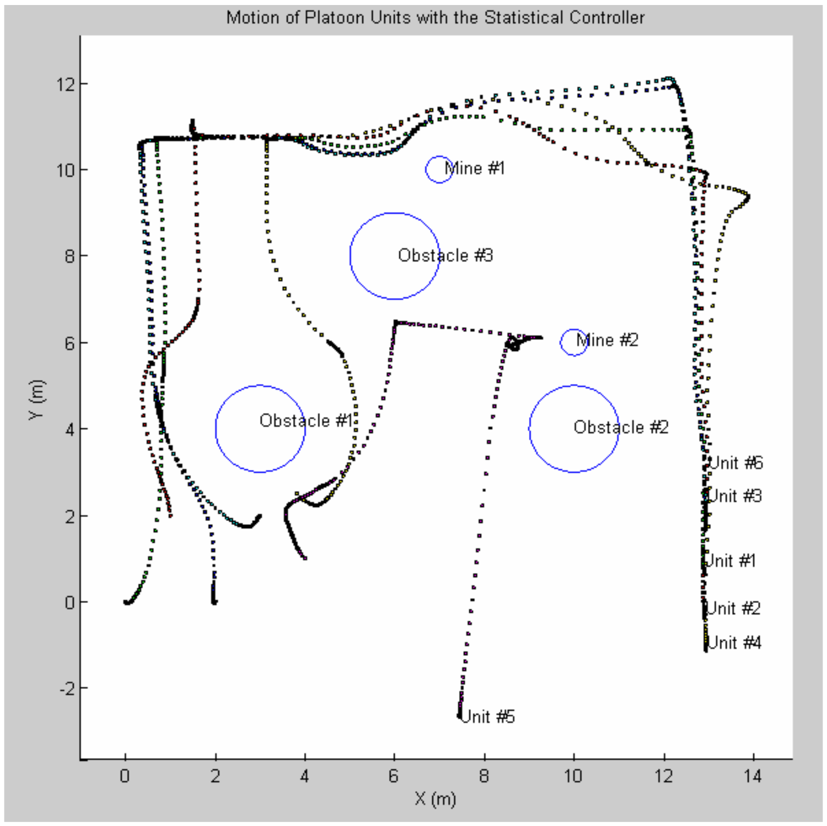


Figure 25: Plot of the motion of a robot swarm

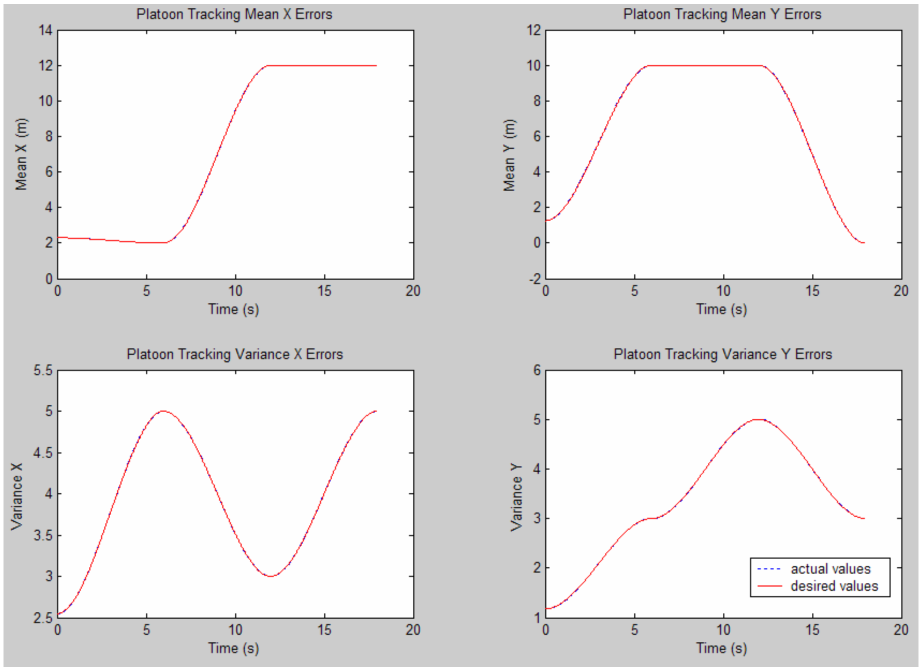


Figure 26: Plot of actual and desired mean and variance of statistical control

The swarm of 6 robots moving through the sequence of waypoints is shown in Figure 25. The squares represent the robots' position at each moment in time. The robots were observed to perform simple obstacle and mine avoidance tasks. The majority of the robots in the swarm clumped together and moved in a similar linear formation while a single robot moved away to ensure that the swarm maintained the desired mean and variance. This result shows that the statistical controller did not perform other tasks, such as inter-robot repulsion or random motion, which are highly regarded in autonomous robotic motion.

Figure 26 shows the desired (solid) and actual (dashed) mean and variance of the platoon over the simulation. The actual and desired mean and variance of the robot swarm remained the same with minor disparities of value 0.01. These disparities represent simple numerical simulation effects. Code can be found in Appendix H and I.

5.6.2 Conclusions

The results of the simulation show that the robot swarm had poor coverage of the area it was searching, and also had poor secondary task capabilities. The main focus of the robots was to maintain the mean and variance while performing obstacle avoidance. Although the robots kept to the projected trajectory of the swarm, they showed little individuality or autonomy when moving independently. Five out of the six robots present displayed similar path characteristics that did not contribute to maximize the search area. Thus, each robot in the swarm needed to have more autonomy to move around based on its surroundings, while at the same time staying within the area designated by the swarm task space control.

5.7 Important Discovery

An important observation was made regarding this controller. The behaviors designed in Sections 3 and 4 return velocity commands to the swarm elements, and were written in terms of differential equations. The statistical controller relies on secondary tasks encoded as velocity commands. Thus, the possibility of combining the controllers together became obvious. The velocity-based behaviors can be included in the v that defines secondary objectives, while the motion and spread of the platoon are controlled in a provable manner. Such a combination of the systems-theoretic and behavior-based approaches ultimately gave rise to a new hybrid controller with much greater functionality than either of the original controllers, as the individual characteristics of the two schemes were combined to provide some of the best qualities of each without the associated drawbacks.

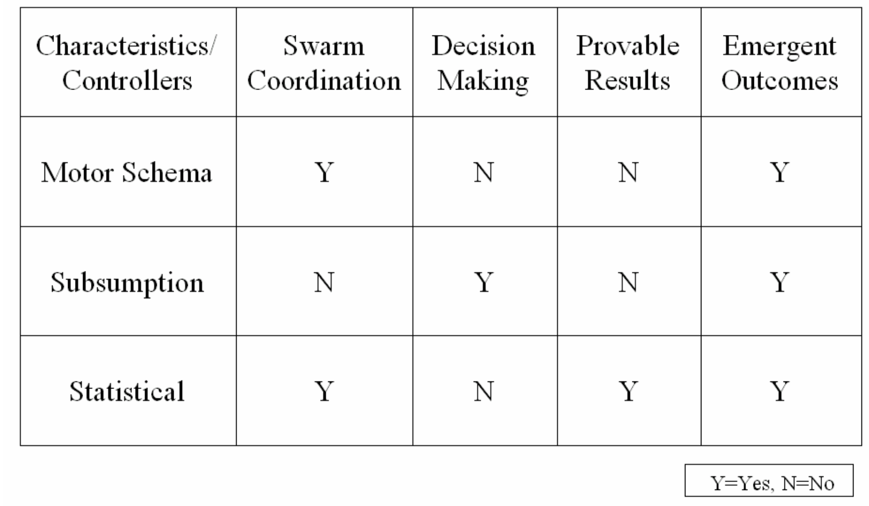
6. Hybrid Controllers

6.1 Reasons for Hybrid Control

Based on the results from the behavior-based and statistical controllers, some conclusions were drawn. Behavior-based systems were seen to be highly reactive while requiring little computation, even when controlling a large number of robots (since such systems tend to be decentralized). Unfortunately, as exemplified by the behavior-based approaches shown in Sections 3 and 4, coordination and control of a group of robots using only decentralized APFs was not practical, nor an especially effective use of the swarm capabilities. Although using the centroid as a center of mass for the swarm to calculate aggregation potentials makes sense, the resultant formation for a large number of robots tended to be concentric, with no specific structure. In fact, for small platoons in open water, the configuration resembled the vertices of a regular polygon. These behaviors were characteristic of the use of inter-robot repulsion as well as attraction to the centroid. Unfortunately, this result served no real purpose for UMCM; it effectively constrains the platoon to a loose conglomeration formation, influenced strongly by the environment and the parameters of the controller.

In the tested behavior-based systems, there was neither feedback control to ensure that the swarm was capable of reaching desired swarm specifications, nor even any guarantee of stability. While simple combinations of repulsive and attractive forces may work for a small swarms of robots such as those simulated in Sections 3 and 4, such controllers may not prove effective if the number of units employed is in the hundreds. This is because there is no real coordination between robots except through simple local reaction. Due to this, behavior-based controllers for large swarms of simulated animals have been seen to suffer from stability problems.[45]

System-theoretic methods, on the other hand, have provable performance, being designed from the standpoint of stability analysis. An analysis of the statistical controller indicated that even though it had great swarm level control capabilities, the controller lacked capacity to introduce intelligence to the system. The robots under statistical control revealed that they lacked the ability to make decisions based on their surroundings. Subsumption, on the other hand, showed remarkable performance in directing the robots to avoid obstacles, making decisions instantaneously. Therefore it made sense to try to combine these two methods into a hybrid that would not only grant autonomy for each robot to move around but also constrain the motion of the swarm as a whole using swarm-level functions. The discussion in Section 5 has shown that the statistical controllers are capable of being combined with behavior-based controllers. The task arbiter, used in statistical swarm control was determined to be favorable for inserting velocity-based behaviors under the motor schema and subsumption architectures. Figure 27 summarizes the comparisons of the controllers.



Characteristics/ Controllers	Swarm Coordination	Decision Making	Provable Results	Emergent Outcomes
Motor Schema	Y	N	N	Y
Subsumption	N	Y	N	Y
Statistical	Y	N	Y	Y

Y=Yes, N=No

Figure 27: Table of comparison of the controllers

The combination of two structurally different controllers resulted in a hybrid that is incredibly robust, with much better performance than the original controllers. The remainder of this section is devoted to development of this new controller and its capabilities.

6.2 Structure of Hybrid Controller

6.2.1 Hybrid Controller Combining Subsumption with Statistical Method

The simulations in this section show how the statistical and subsumption controllers were combined to result in a hybrid controller. The change made to the statistical controller was minimal; the task arbiter block was converted to a behavior arbiter. The behavior arbiter block was where the subsumption architecture was embedded. Figure 28 shows the structure of the hybrid controller while Figure 29 shows the internal organization of the behavior arbiter.

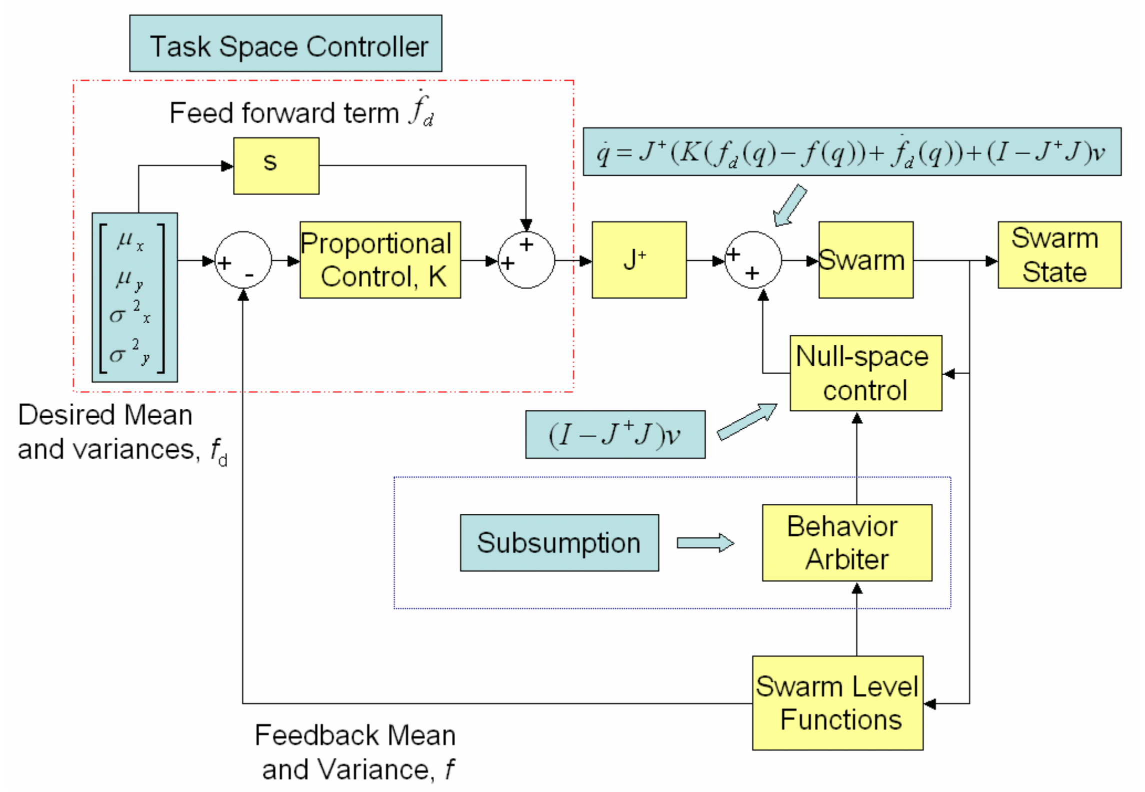

Task Space Controller
Feed forward term $\dot{f}_d$
s
$\dot{q} = J^+(K(f_d(q) - f(q)) + \dot{f}_d(q)) + (I - J^+J)v$
$\begin{bmatrix} \mu_x \\ \mu_y \\ \sigma^2_x \\ \sigma^2_y \end{bmatrix}$
+
-
Proportional Control, K
+
+
J^+
+
+
Swarm
Swarm State
Null-space control
$(I - J^+J)v$
Desired Mean and variances, f_d
Subsumption
Behavior Arbiter
Swarm Level Functions
Feedback Mean and Variance, f

Figure 28: Hybrid controller block diagram

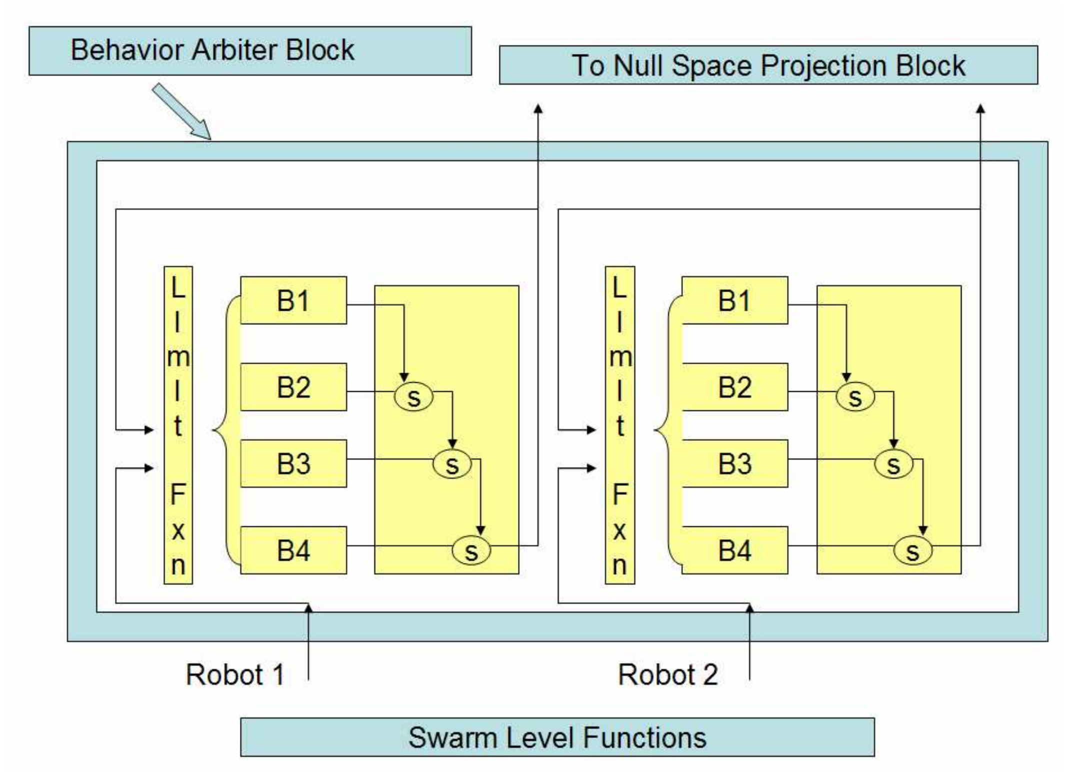


Figure 29: Behavior arbiter block embedded with subsumption

The behavior arbiter in Figure 29 shows that, within the block, there was a subsumption architecture for each robot. That architecture controlled the secondary tasks for the robot through a decision making process. The subsumption architecture was similar to the one developed in the behavior-based section in section 4.3 and had the same behaviors programmed for UMCM as seen in Figure 20. It was ordered as such: the highest behavior in the hierarchy is mine avoidance followed by obstacle avoidance, aggregation, separation and then random motion. The stimuli for activating specific behavior in the hierarchy were still based on calculations of ranges from the robot to another robot, obstacle or mine. The robots in the simulation were assumed to be able to receive range information to an obstacle similar to real implementation using sensors, which would give relevant ranging information when an obstacle is discovered.

Additionally, the limit function in the behavior arbiter block effectively prevented all the robots from reacting to the environment at the same time, thereby overtaxing the system. It limited the number of robots attempting ‘Random_Search’, the lowest-level behavior in the subsumption architecture. If there were more than three robots with active behaviors, the remaining robots only implemented behaviors of higher priority than ‘Random_Search.’ This was a simple method of resource planning intended to minimize the impact of random motion on obstacle avoidance of other units by not competing for the ‘redundant’ degrees of freedom. The behavior arbiter block sent the expressed behavior for each robot to the null-space projection block where calculations were done to move each robot to the best position to satisfy the null-space and task space variable conditions.

6.2.2 Simulation Results

The simulation result for the hybrid controller is shown in Figure 30. The robots performed obstacle, mine and robot avoidance as they moved along the desired trajectory. The coverage of the swarm dramatically improved compared to the simulation results of the statistical controller alone. The improved dispersion of the trajectories taken by each robot is evident from the plots. The randomness was indicative of the inherent decision making processes in the controller. The decisions made by each robot were different based on its position and environmental factors present. The paths around the obstacle were also distinctive of the decision process. When a robot was close to an obstacle, it repelled away from it in response. At the next iteration, once again, each robot had to decide on what environmental factors were important enough to be considered. If the robot was not repelled far enough to be out of the obstacle or mine range, it continued with obstacle or mine avoidance. However, if the robot escaped the repulsion range of the obstacle or mine, it then decided what other behavior to express. This zigzag motion, known as hysteresis, around the obstacle demonstrated the decision process that the robots actively engaged in on an individual level.

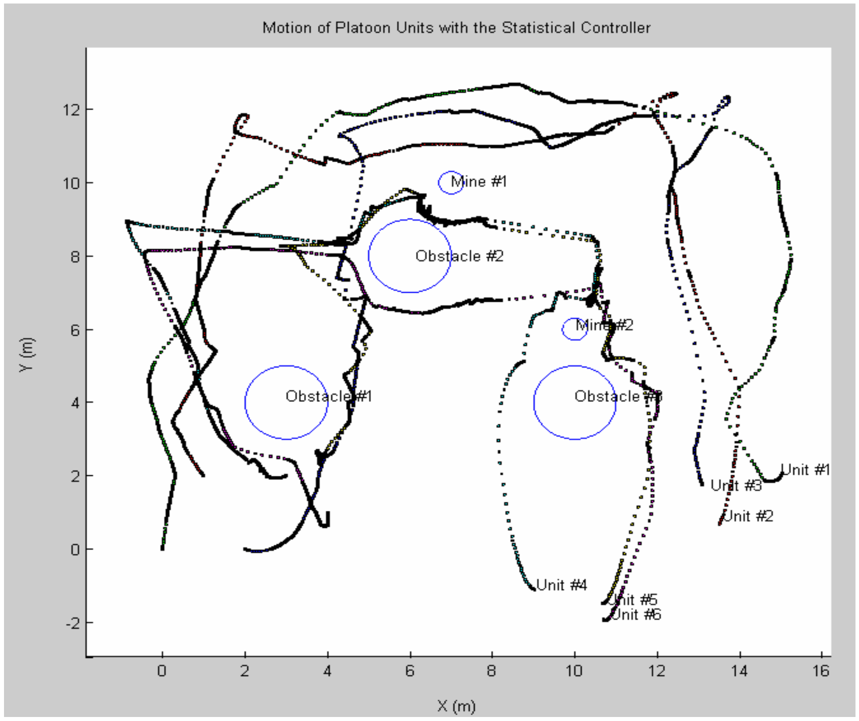


Figure 30: Plot of motion of robot swarm with hybrid control

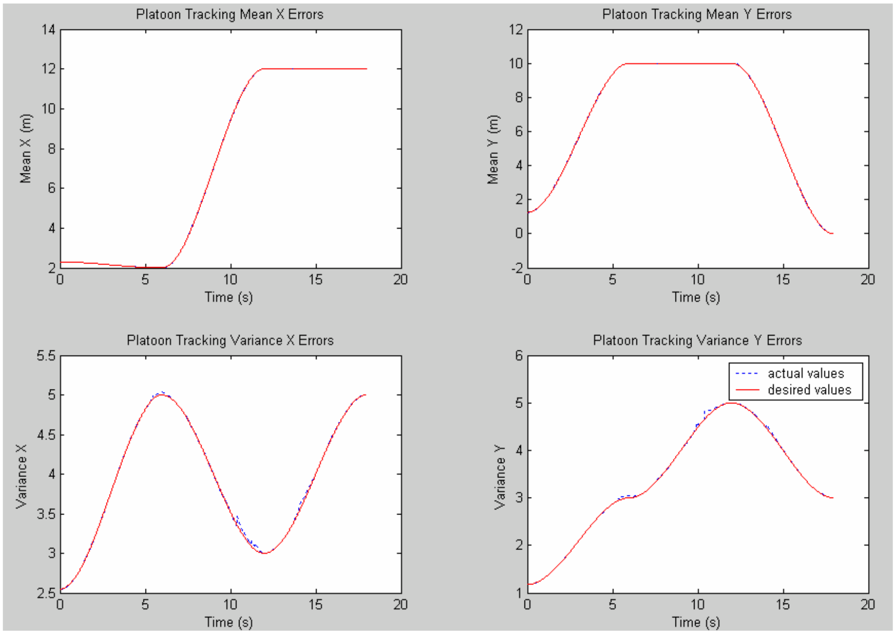


Figure 31: Plot of mean and variance in X and Y with hybrid control

Even though each robot moved in accordance with the environmental changes, the robots were still able to maintain the mean and variance of the swarm. The plots in Figure 31 show the plot of the mean and variance in the X and Y directions. The desired and actual mean and variance of the swarm were very similar with minute errors, again due to computation limits. This result demonstrated the flexibility of the hybrid controller to have different robot units behave in ways to suit the individual's situation as well as satisfying the swarm's requirement. MATLAB codes can be found in Appendix J and K.

6.2.3 Conclusions

From Figures 30 and 31, it was concluded that the hybrid controller performed as expected, tracking the desired swarm mean and variance. While the task-space controller satisfied the swarm-level functions, the null-space controller clearly took into account the decisions made by individual robots, providing substantially increased unit autonomy and better overall area coverage than the statistical controller alone.

6.3 Hybrid Controller with Disturbances

Although the hybrid controller worked well for the initial simulation, the performance of the controller in more complex situations was still unknown. The next simulation carried out attempted to investigate the behavior of the hybrid controller when the environment became more complex. This was accomplished by the addition of a simple drift term intended to model a current field. The drift in a real underwater environment is highly variable and unpredictable, hence any controller to be used in such an environment would require a large degree of robustness and flexibility so that it could react and compensate for the external disturbance.

Figure 32 shows the addition of an environmental disturbance block to the block diagram for the hybrid controller while Figure 33 illustrates the simulated drift currents.

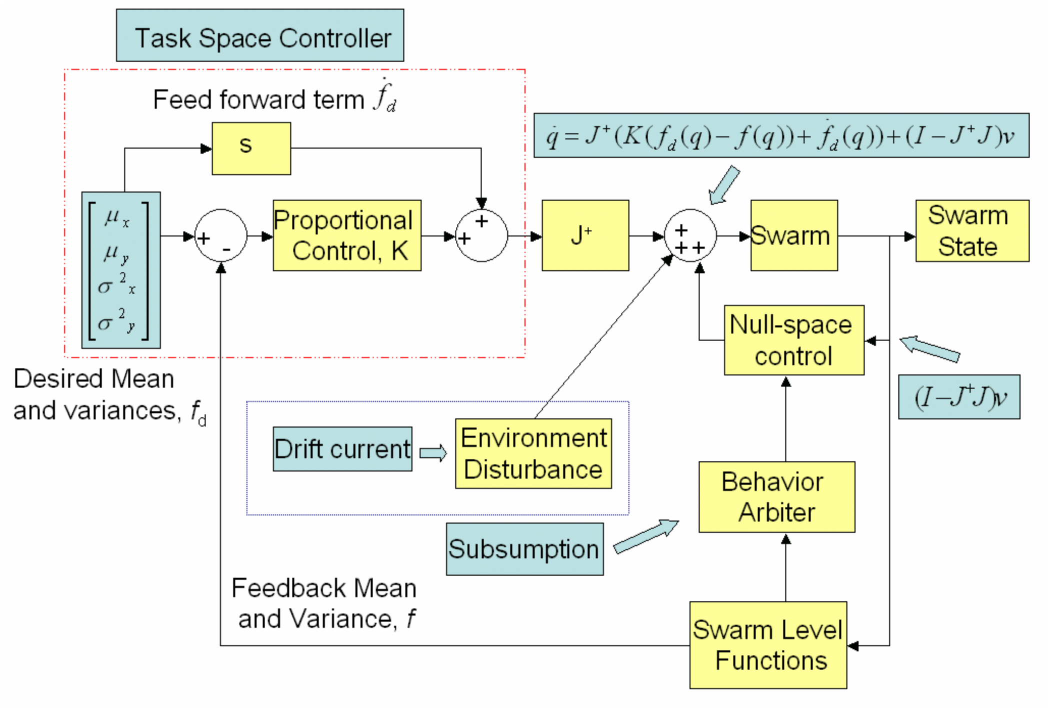


Figure 32: Hybrid controller block diagram with environmental disturbance

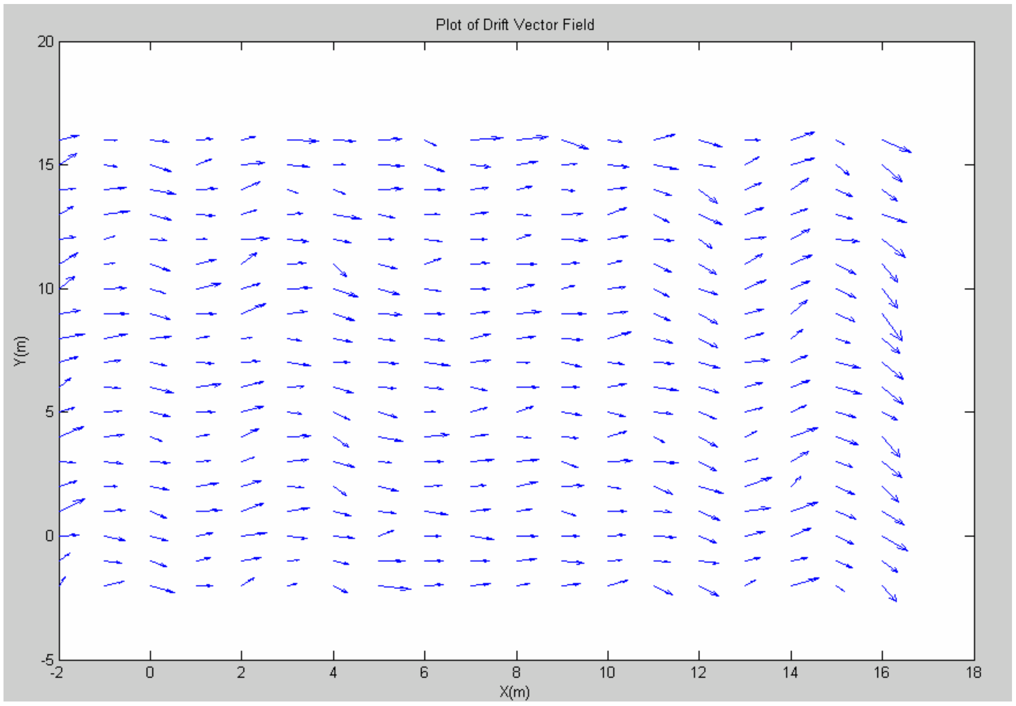


Figure 33: Vector field depicting a drift current

The non-uniform vector arrows in the above field plot represented the drift current that would be expected in an undersea environment. The individual vector field arrows in the simulation denote the direction of the current flowing at a given point. Unlike a real drift current with laminar flow, the direction of flow at different points in the simulated current field varied discretely across the space. This flow field represented an extremely challenging sample so that it was possible to test the limits of the capabilities of the swarm. The simulated current (environmental disturbance) was added to the commanded velocity of each robot in the robot swarm, based on its location, at each iteration of the simulation.

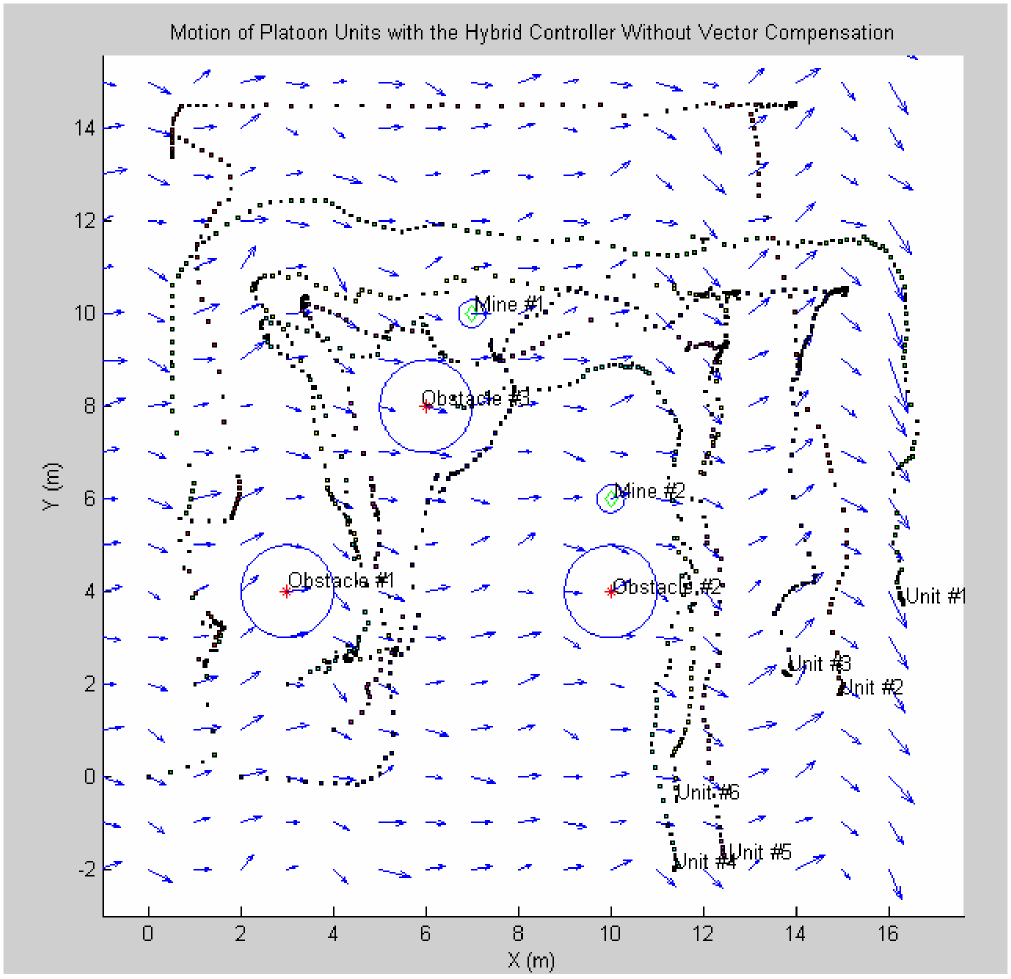


Figure 34: Plot of swarming robots with hybrid control in a drift current

Figure 34 illustrates the motion of the robots as they move along the same designated path as in the simulations from Section 5.6.1. Different from the initial simulation, it was observed that the robots were pushed away in the direction of the current and were not able to counteract the effects of the additional drift currents. Since the flow field moved from left to right, the error in the mean and variance in the x direction was higher. Figure 35 shows how the addition of the environmental disturbance affected the mean and variance control. It is worth noting that the error seen in these plots could be somewhat reduced by increasing the proportional gain. Even though the robot swarm did not match the desired trajectory exactly, it still followed the general form of the desired motion and fulfilled individual task requirements, avoiding collisions with surrounding objects and fellow robots. MATLAB codes can be found in Appendix L and M.

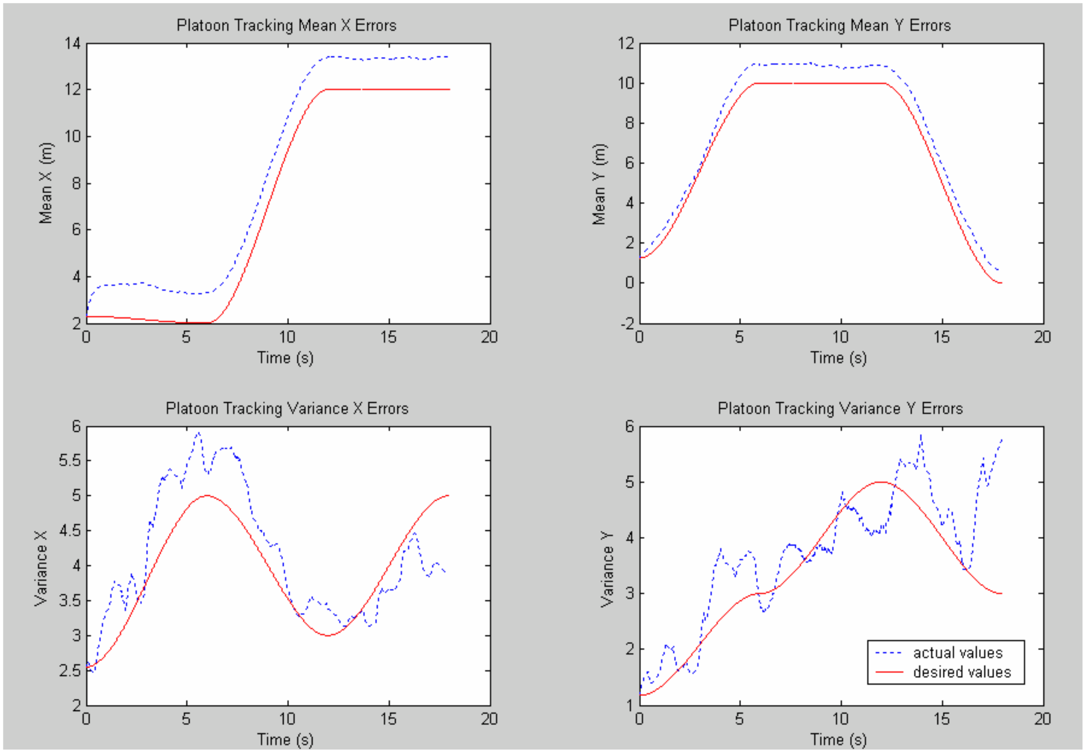


Figure 35: Plot of mean and variance of a hybrid controller in a drift current

6.3.1 Conclusions

The swarm was not successful in achieving the desired task function $f(q)$ when faced with simulated drift current. The robots were dispersed outwards, away from the desired mean with the actual mean being consistently higher, which was expected based on the drift current. The variance, on the other hand, was very poorly controlled. The swarm task space controller failed to satisfactorily limit the position of the robots based on the variance. The addition of the external stimulus demonstrated a weakness of the controller. Fortunately, the systems-theoretic nature of the primary task controller led to an improved implementation using standard tools of feedback control design.

6.4 Hybrid Controller with Vector Compensation

Based on the results of Section 6.2, the hybrid controller was altered to take into account the environmental errors present. A task-space error integration block was added to the hybrid controller to compensate for the error produced between the desired and actual position of the mean and variance. The addition of the integrated error enables the hybrid controller to slowly, over time, compensate for the influence of the vector field. Figure 36 shows where the error integration block was added into the hybrid controller.

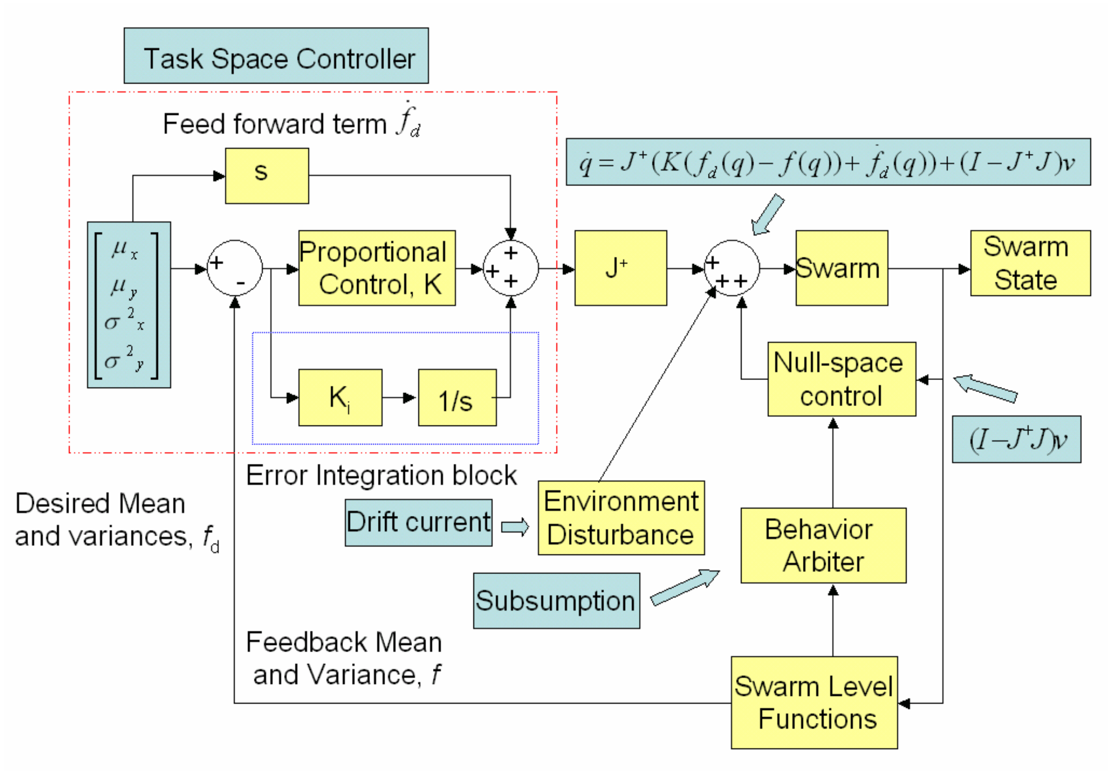

Task Space Controller
Feed forward term $\dot{f}_d$
s
$\dot{q} = J^{+}(K(f_d(q) - f(q)) + \dot{f}_d(q)) + (I - J^{+}J)v$
$\begin{bmatrix} \mu_x \\ \mu_y \\ \sigma^2_x \\ \sigma^2_y \end{bmatrix}$
Proportional Control, K
J^{+}
Swarm
Swarm State
K_i
1/s
Null-space control
$(I - J^{+}J)v$
Error Integration block
Desired Mean and variances, f_d
Drift current
Environment Disturbance
Behavior Arbiter
Subsumption
Feedback Mean and Variance, f
Swarm Level Functions

Figure 36: Block diagram of a hybrid controller with vector compensation

Figure 37 demonstrates the effect of including the task-space error integration block, which effectively removed the error as seen in Figure 38. Although the paths of the swarm elements were not clearly ‘better’ than those of the simulation without the integral term, the calculation of the desired and actual mean and variance of the robot swarm in Figure 37 showed significant improvement in tracking the desired task function. The integral control dramatically improved the mean tracking characteristics. The variance control, while not as clean as the mean, still effectively improved performance by decreasing the maximum error from the desired variances. A direct comparison of mean-squared error over the trajectory would be misleading, as the paths of the units passed through different vector fields throughout the two simulations.

The difficulty in maintaining the variance of the swarm was due to the effect of the varying drift factors. The vector field was set up such that it changes spatially as the robots reach different positions. Nevertheless, the simulations showed that a hybrid controller with vector compensation works much better since the controllers adapted the motion of each robot to suit the objectives of the swarm. In this case, the swarm level objectives were not compromised while the task objectives were being achieved. MATLAB codes can be found in Appendix N and O.

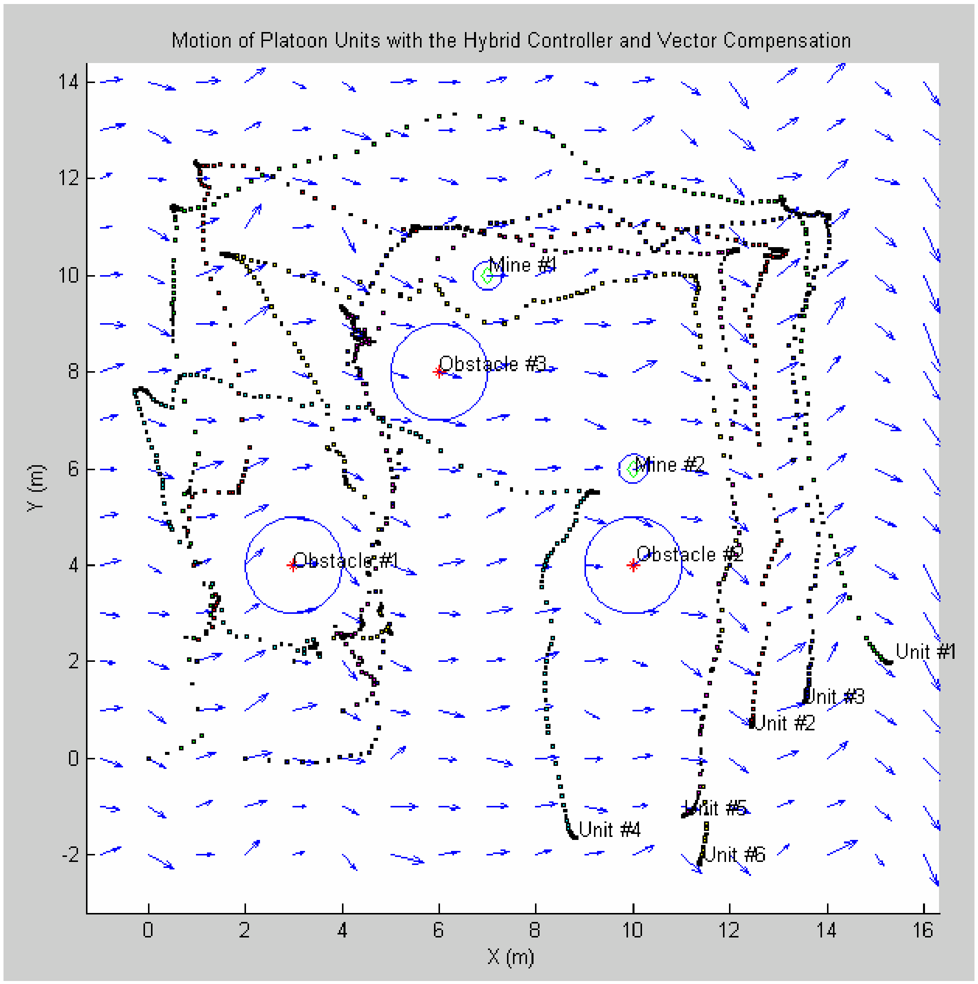


Figure 37: Plot of swarming robots with hybrid control and vector compensation

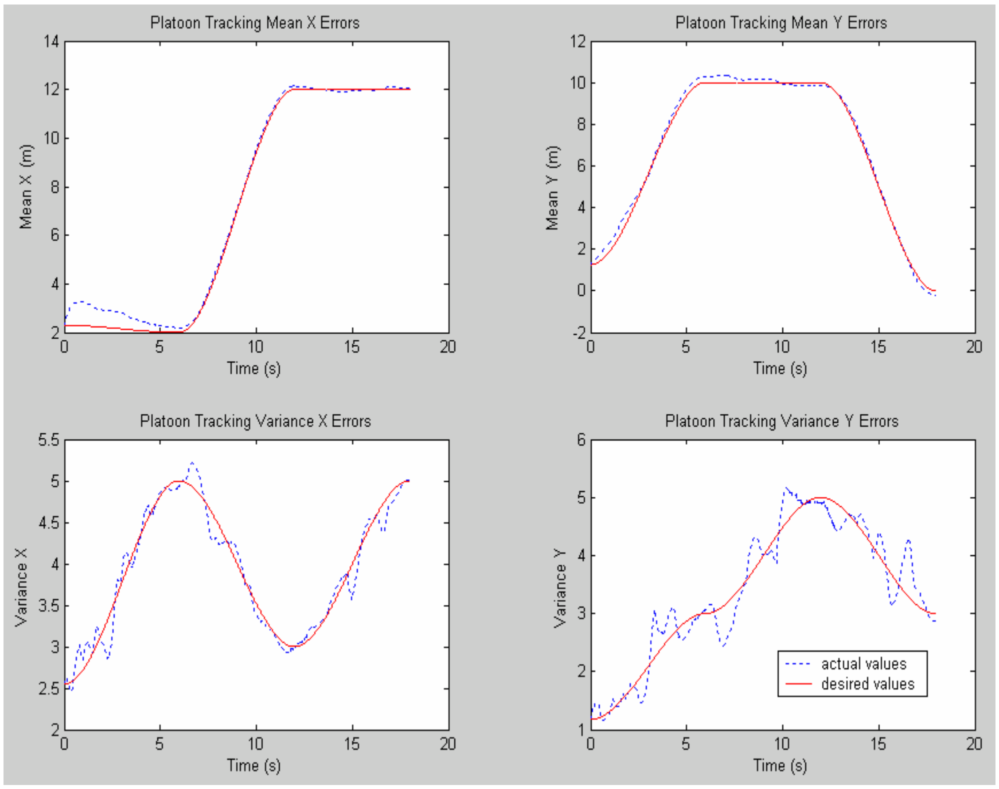


Figure 38: Tracking errors for a hybrid controller with vector compensation

6.4.1 Conclusions

The new hybrid controller showed greater versatility and robustness when it was modified to include the task-space error integration block that corrected for the addition of the vector fields in the system. The new hybrid controller therefore has both quantitative and qualitative improvements over its parent controllers; it has error rejection for improved robustness, decision-making capabilities to give the robots enhanced artificial intelligence for potentially better swarm usage, and the ability to regulate swarm-level functions of the platoon state. No other controller has all of these capabilities.

7. Summary of findings

This project has effectively and thoroughly studied two specific types of control methodologies for a swarm of robots performing underwater UMCM. The behavior-based methods showed their strengths and limitations in the earlier sections in the research. Behavior-based systems are highly flexible and robust systems, as they need not have a lot of specific information about the surroundings they are in. They only require real-time sensor data to trigger individual behaviors. The motor schema approach was effective for controlling the motion of the individual robots within a swarm, but lacked flexibility or any provable stability due to its lack of a feedback controller to regulate the performance of the system. Subsumption, on the other hand, showed poor aptitude for swarm control. On an individual member level, the subsumption structure gave each robot a nominal amount of artificial intelligence for it to carry out its tasks. Each robot in the swarm was able to decide what the next behavior or task it should perform was, based on environmental information alone. As a result, each individual robot moved in a distinct path, different from its neighboring robots. Thus, the swarm lacked coordination except on the most basic level (collision avoidance).

Statistical control was the systems-theoretic controller that was investigated. Statistical control uses classical control methods combined with techniques from redundant manipulator control to determine motion commands for each member in a robot swarm. The original statistical controller was able to move the swarm effectively; following a desired task-space trajectory defined by swarm mean and swarm variance. Although the robot swarm avoided obstacles while it moved from waypoint to waypoint, it did not show intelligence in deciding what path it was taking. Effectively, the redundancy of the swarm was not well used.

A qualitative comparison of the 3 methods indicates that motor schema showed the least potential because it neither produced good provable performance nor demonstrated good robot intelligence or a decision-making structure. Subsumption proved its versatility in handling the individual robots through its decision-making capabilities, but showed a weakness in swarm coordination. The statistical controller demonstrated good performance, but lacked the artificial intelligence required for the autonomous agents to work well in an underwater environment. Therefore, it made sense that a hybrid between the statistical controller and subsumption architecture could result in a more robust system that could handle the demands of UMCM.

In the process of developing the analysis simulations, an important link was made between two methods. The statistical controller used differential equations to represent secondary tasks to be carried out. The possibility of hybridizing subsumption and statistical methods was immediately recognized since the subsumption codes, already written in differential equations, could be embedded into the statistical controller codes to generate a hybrid. Creating a hybrid controller meant that the product would be highly versatile in an unknown environment while still being able to meet provable performance demands. Furthermore, the ability to make decisions in an unknown environment was seen to be critical for improved performance in real-world situations.

The creation of the hybrid controller proved to be successful. The resultant controller demonstrated the best characteristics of its parent controllers, satisfying the individual behavior-based tasks as well as regulating swarm level functions while still requiring only minimal planning. The subsumption architecture allowed the swarm to make decisions based on the simulated environment and as a result, gave rise to emergent, indeterministic paths. The statistical controller framework enabled the swarm to track the desired mean and variance trajectories.

A weakness of the hybrid system was observed in the testing phase, when drift factors were added into the simulation. Under these disturbances, the hybrid controller was incapable of regulating swarm mean and variance, although the system remained stable and still carried out the mission.

To compensate for the difficulty of the hybrid controller with regards to drift vectors (and any other velocity disturbances), a task-space error integration block was added to the hybrid controller. The compensated system was able to overcome the effects of the external drift factors.

In closing, this project involved the investigation of three distinct techniques for swarm control of robots in the UMCM domain. The simulations shown in this research demonstrated numerous features, including the strengths and weaknesses of the various swarm control techniques. A hybrid controller with qualitative and quantitative improvements over the parent controllers was also created. The investigative portion of this project should serve as a good overview for future work on a real-world implementation of swarm controllers for underwater, surface or land multi-agent controls. Further, the new hybrid controller represents an important contribution to the field of robot swarm control, as it combines systems-theoretic and behavior-based methods into a cohesive framework.

Endnotes

[1] T. Balch and R. Arkin. “Communication in Reactive Multi-Agent Robotic Systems,” *Autonomous Robots*, 1(1): 27-52, 1994.
[2] X.Yun. “Line and Circle Formation of Distributed Physical Mobile Robots.” *Journal of Robotic Systems*, 14(2): 63-81, 1997.
[3]J. Jablonski and J. Posey, “Robotics Terminology,” *Handbook of Industrial Robotics*, ed. S. No, J. Wiley, New York, Pg 1271-1303, 1985
[4] Arkin, “Whence Behavior?” Behavior-Based Robotics. Massachusetts: MIT Press, 1998. Pg 2.
[5] Lueth, Dllman, Dario, and Worn, Distributed Autonomous Robotic Systems 3, Germany, Springer, 1998.
[6] Arkin, “Behavior-Based Architectures,” pg 127.
[7] Ibid, pg 128.
[8] Arkin “Robot Behavior,” pg 66.
[9] R. Smith, “Adaptive Robotics- Behavior-Based Robotics” 12 May 2003. http://www.inel.gov/adaptiverobotics/behaviorbasedrobotics/
[10] Arbib, “Schema Theory,” in the Encyclopedia of Artificial intelligence, 2nd Edition, Ed S. Shapiro, Wiley-Interscience, New York, N.Y. pp 1427-43.
[11] Arbib, “ Perceptual Structures and Distributed Motor Control,” in Handbook of Physiology- The Nervous System II: Motor Control, Ed V.B. Brooks, American Physiological Society, Bethesda, MD, pp 1449-80, 1981; Arkin pg 141.
[12] Arkin R. “Motor Schema Based Navigation for a Mobile Robot, an Approach to Programming by Behavior,” in *IEEE Proc. Robotics and Automation*, pg. 264 -271, 1987.
[13] Arkin pg 131, 1987 “Motor Schema Theory Based Navigation,” Behavior-Based Robotics, pg 142.
[14] Arkin “Behavior-Based Architectures,” pg 131.
[15] Ibid, pg 145-155.
[16] Ibid, pg 144.
[17] R. Arkin, “Motor Schema Based Navigation for a Mobile Robot, an Approach to Programming by Behavior,” in *IEEE Proc. Robotics and Automation*, pg 264-271, 1987.
[18] R. Arkin, *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1998
[19] J.A. Piepmeier and P.A. Morgan. “Uncalibrated Vision–Based Control of Two-Wheeled Mobile Robots.” In *Proc. International Mechanical Engineering Conference and Exposition*, CDROM, New York, NY, 2001.
[20] Arkin, “Behavior-based Architectures”, pg 14-27.
[21] Ibid, pg 150.
[22] B.E. Bishop, Artificial Potential Fields Programs: Repulse.m, Obst.m.
[23] Arkin, “Behavior-Based Architectures,” pg 151-155.
[24] D.J. Montana, D.L. Brock, A.Z. Ceranowicz, “Coordination and Control of Multiple Autonomous Vehicles,” in *Proc. IEEE International Conference on Robotics and Automation* (Nice, France), pg 2725- 2730, 1992.
[25] P. Hsu, J. Hauser and S. Sastry, "Dynamic Control of Redundant Manipulators," in *Proc. IEEE Int'l Conference on Robotics and Automation,* 1998, pp.183-187.
[26] ‘Parallel SAR Pattern.’ *Virtual U.S. Coast Guard Webpage.* 21 March 2004. 24 March 2004. < http://vuscg.org/>
[27] Arkin “Behavior-Based Architectures,” pg 131.
[28] “Modular Robots,” About *AI.* 04 April 2000. 06 April 2004. *<http://www.aboutai.net/DesktopDefault.aspx?tabindex=1&tabid=2&article=aa040400a.htm* >
[29] H. Van Dyke Parunak, “Applications of Distributed Artificial Intelligence in Industry,” *Foundations of Distributed Artificial Intelligence.* Wiley Inter-Science, Chapter 4, 1994. http://www.erim.org/~vparunak/jennings.pdf
[30] Ibid, pg 135.
[31] Brooks, R.A."How to Build Complete Creatures Rather Than Isolated Cognitive Simulators," in K. VanLehn (ed.), Architectures for Intelligence, pp. 225-239, Lawrence Erlbaum Assosiates, Hillsdale, NJ, 1991, http://ai.eecs.umich.edu/cogarch3/Brooks/Brooks_RealWorld.html
[32]C. Reynolds, Boids, 6 Sept 2001,16 Nov 2003 http://www.red3d.com/cwr/boids/
[33] D.J. Stilwell, B.E. Bishop,” Redundant Manipulator Techniques for Path Planning and Control of a Platoon of Autonomous Vehicles,” *IEEE Conference for Decision and Control* (Virginia, USA), pg 2093 -2098, 2002.
[34] N. Nise, Control Systems Engineering, Third Edition, John Wiley & Sons Inc. pg 2-5.

[35] B.E. Bishop, “On the Use of Redundant Manipulator Techniques for Control of Platoons of Cooperating Robotic Vehicles,” *IEEE Transactions: Systems, Man and Cybernetics,* pg 608-615, 2003.
[36] Ibid, pg 608.
[37] S.B Niku. “Jacobian,” Introduction to Robotics: Analysis, Systems, Applications. New Jersey: Prentice Hall. pg 97-99
[38] Bishop, “On Use of Redundant Manipulator Techniques.” pg 608-609.
[39] Ibid, pg 609.
[40] Ibid, pg 609.
[41] Ibid, pg 609.
[42] B.E. Bishop, “Redundant Manipulators Analogs in Robot Swarm Control.” Presentation to Drexel University, 11 Jan 02.
[43] Stilwell, Bishop, “A Strategy for Controlling Autonomous Robot Platoons.” In *Proc. 39th Conference of Decision and Control*. Sydney, Australia. 2000.
[44] Bishop, “On use of Redundant Manipulator Techniques.” pg 609.
[45] C. Reynolds, Boids, 6 Sept 2001,24 March 2004. http://www.red3d.com/cwr/boids/

8. References

[1] G. Dudek, M. Jenkin, E. Milios and D. Wilkes, “A Taxonomy for Swarm Robots,” in *Proc. IEEE International Conference on Intelligent Robots and Systems*, (Yokohama, Japan), pg 441-447, 1993.

[2] D.J. Montana, D.L. Brock, A.Z. Ceranowicz, “Coordination and Control of Multiple Autonomous Vehicles,” in *Proc. IEEE International Conference on Robotics and Automation* (Nice, France), pg 2725- 2730, 1992.

[3] R. Cassinis, G. Banco, A Cavagnini and P. Ransenigo, “Strategies for Navigation of Robot Swarms to be Used in Landmines Detection,” in *Advanced Mobile Robots*, (Zurich, Switzerland), pg 211-218, 1999.

[4] X. Zheng, “Layered Control of a Practical AUV,” ISE Research Ltd

[5] G. Bruzzone, R. Bono, M. Caccia, G. Veruggio, “ A Simulation Environment for Unmanned Underwater Vehicles Development,” *in OCEANS, MTS/IEEE Conference and Exhibition* (Honolulu, HI, USA), pg 1066- 1072, 2001.

[6] R. Arkin, *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1998.

[7] R. Arkin, “Motor Schema Based Navigation for a Mobile Robot, an Approach to Programming by Behavior,” in *IEEE Proc. Robotics and Automation*, pg 264-271, 1987.

[8] R. Brooks, “A Robust Layered Control Scheme for a Mobile Robot,” *IEEE J. Robot. Automation.*, vol. 2, no. 1, pp. 14-23, 1986.

[9] D.J. Stilwell, B.E. Bishop,” Redundant Manipulator Techniques for Path Planning and Control of a Platoon of Autonomous Vehicles,” *IEEE conference for Decision and Control* (Virginia, USA), pg 2093 -2098, 2002.

[10] B.E. Bishop, “On the Use of Redundant Manipulator Techniques for Control of Platoons of Cooperating Robotic Vehicles,” *IEEE Transactions: Systems, Man and Cybernetics*, pg 608-615, 2003.

[11] G. Butler, A. Gantchev, P. Grogono, “Object-Orient Design of the Subsumption Architecture,” Software-Practice and Experience, pg. 911-923, 2001.

[12] H. Van Dyke Parunak, “Applications of Distributed Artificial Intelligence in Industry,” *Foundations of Distributed Artificial Intelligence.* Wiley Inter-Science, Chapter 4, 1994. http://www.erim.org/~vparunak/jennings.pdf , (10 NOV 03)

[13] E. Kobayashi, Aoki, Hirokawa, Ichikawa, Saitou, Miyamoto, Iwasaki, H. Kobayashi, “Development of an Autonomous Underwater Vehicle Maneuvering Simulator,” *OCEANS, MTS/IEEE Conference and Exhibition*, pg 361-368, 2001

[14] Brooks, R.A. "How to Build Complete Creatures Rather Than Isolated Cognitive Simulators," in K. VanLehn (ed.), Architectures for Intelligence, pp. 225-239, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991, http://ai.eecs.umich.edu/cogarch3/Brooks/Brooks_RealWorld.html (10 NOV 03)

[15] R. Smith, "Adaptive Robotics- Behavior-Based Robotics" 12 May 2003. http://www.inel.gov/adaptiverobotics/behaviorbasedrobotics/ (11 OCT 03)

[16] C. Reynolds, Boids, 6 Sept 2001, http://www.red3d.com/cwr/boids/ (7 SEPT 03)

[17] J.A. Piepmeier and P.A. Morgan. "Uncalibrated Vision–Based Control of Two-Wheeled Mobile Robots." In *Proc. International Mechanical Engineering Conference and Exposition*, Pg CDROM, New York, NY, 2001.

[18] D. Stilwell, B.E. Bishop. "A Strategy for Controlling Autonomous Robot Platoons." In *Proc. 39th Conference of Decision and Control*. Sydney, Australia. 2000

[19] S.B Niku. Introduction to Robotics: Analysis, Systems, Applications. New Jersey: Prentice Hall.1998

[20] 'Parallel SAR Pattern.' *Virtual U.S. Coast Guard Webpage*. 21 March 2004. http://vuscg.org/ (24 March 04)

[21] P. Hsu, J. Hauser and S. Sastry, "Dynamic Control of Redundant Manipulators," in *Proc. IEEE Int'l Conference on Robotics and Automation*, 1998.

[22] N. Nise, Control Systems Engineering, Third Edition, John Wiley & Sons Inc, 2000.

```
% Appendix A
% Yong Chye Tan
% Trident Project SImulation1
% Simulation of 1 robot in a 2-D field avoiding obstacles

function ldot = oas(t,p)
global Obst

Katt = 1;   %attractive gain
Krep = 1; %repulsive gain
Xobst = 2;   %x coodinate of obstacle
Yobst = 3 ;    %y coodinate of obstacle
r=0.1;          % sensing radius of holonomic robot
R= 0.5;        %magnitude of radius of obstacle
rho = R+ 0.5;     %maginitude of radius of influence of obstacle
Xt= 3;       %x coodinate of target
Yt=5;          %y coodinate of target
x=p(1);     %x coordinate of holonomic robot
y=p(2);     %y coordinate of holonomic robot

plot(Xobst, Yobst,'*');     % plot center of obstacle
theta_att= atan2(Yt -y, Xt-x); % attractive angle
xatt = Xt- x; ; % x dist of attractive vector
yatt = Yt - y;  % y dist of attractive vector

L = [xatt ; yatt];  % vector to show length of attractive vector
Mag_L= sqrt(L'*L);  %magnitude of length
attctrl = Katt*[(Mag_L)*cos(theta_att) ;(Mag_L)*sin(theta_att)];   %attractive
control, potential

%repulsive control
theta_rep= atan2(y-Yobst, x-Xobst);   %angle of repulsion
xdist = x-Xobst-R*cos(theta_rep) ;   %x distance of vector
ydist = y-Yobst-R*sin(theta_rep);  %y distance of vector
Length = [xdist ; ydist];   % vector to delineate length
Mag_Leng=sqrt(Length'*Length);  %magnitude of Length

if ( Mag_Leng<(rho-R))

repctrl=Krep*[(1/Mag_Leng)*cos(theta_rep)-(1/(rho-R))*cos(theta_rep);
         (1/Mag_Leng)*sin(theta_rep)-(1/(rho-R))*sin(theta_rep)] ;  % repulsion
control. potential

 else
     repctrl=[0;0];

 end

ldot = repctrl + attctrl;   %controller
```

```
% Appendix B
% Yong Tan
% Trident Scholar Project
% distcalc to be used to find distance from each other
function ldot= distcalc(t,p)


[n,m]=size(p); %converting the column vector into a 3*2 vector
count=n/2;
r=zeros(count,2);
A=0;
for i=1:n           %conversion of a 6 by 1 column vector into a 3 by 2
    if rem(i,2)==0  %if i is even move to 2nd column
          y=i/2;,
        r(y,2)=p(i,1);
    else            %if i is odd, move to 1st column
        y=(i+1)/2;
      r(y,1)=p(i,1);
    end

end

Katt=1; % attractive constant

Krep=10; % repulsion constant

% find centroid of all 3 robots
[n, m]=size(r); % size of robo vector
A=sum(r,1);     % sum of all x,y coordinates
Xc= A(1,1)/n;     % x-coordinate of centroid of all robots
Yc=A(1,2)/n;        % y-coordinate of centroid of all robots
plot(Xc, Yc,'*');
hold on;

%Attractive control
att=zeros(n,m);  %create matrix

for i=1:n
    att(i,1)=Xc-r(i,1);    % find x attractive potential
    att(i,2)=Yc-r(i,2);     % find y attractive potential
end

attctrl= Katt* att;

rep=zeros(n,2);
qa=zeros(1,2);
qb=zeros(1,2);

for i=1:(n-1)       % Matrix to write the repulsive potential
    for j=2:n       % by reducing the number of times each point is added, we run
the matrix 1/2 the required times
       if i==j      % if i=j, same point, thus discard
       else
           Xa=r(i,1);
            Ya=r(i,2);
            Xb=r(j,1);
            Yb=r(j,2);

           dist_ab=sqrt(((Xa-Xb)^2) +((Ya-Yb)^2));

           theta_ba=atan2(Ya-Yb, Xa-Xb);
           qa=[(1/dist_ab)*cos(theta_ba),(1/dist_ab)*sin(theta_ba)];
           rep(i,:)=rep(i,:)+qa;

           qb= [(1/dist_ab)*cos(theta_ba+ pi),(1/dist_ab)*sin(theta_ba+pi)];
           rep(j,:)=rep(j,:)+qb;

       end
   end
end

repctrl= Krep*rep;

q=zeros(n,m);
qspec=zeros((n*m),1);

 q = attctrl + repctrl;   %sum of attractive and repulsive controller
 counter=0;
    for i=1:n
        for j=1:m
            counter=counter+1;
            qspec(counter,1)=q(i,j);

        end
    end

 ldot=qspec;
```

```
% Appendix C
% Yong Tan
% Trident Scholar Project

% distcalc to be used to find distance from each other
function ldot= multiobst(t,p)
%list constants
global obst

Katt=1; % attractive constant
Krep=5; % repulsion constant
Kattarget=5;    % attractive constant to target
Kreptar=15;     % repulsion constant from obstacle
Xt=20;   %x-coordinate of target
Yt=20;   %y-coordinate of target

R=0.5;  %radius of obstacle
rho=R+1;  %radius of influence
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[n,m]=size(p); %converting the column vector into a 3*2 vector
count=n/2;
r=zeros(count,2);
A=0;
for i=1:n           %conversion of a 6 by 1 column vector into a 3 by 2
    if rem(i,2)==0  %if i is even move to 2nd column
          y=i/2;,
        r(y,2)=p(i,1);
    else            %if i is odd, move to 1st column
        y=(i+1)/2;
      r(y,1)=p(i,1);
    end
end

% find centroid of all 3 robots
[n, m]=size(r); % size of robo vector
A=sum(r,1);     % sum of all x,y coordinates
Xc= A(1,1)/n;     % x-coordinate of centroid of all robots
Yc=A(1,2)/n;        % y-coordinate of centroid of all robots
%plot(Xc, Yc,'*');
%hold on;
%Attractive control
att=zeros(n,m);  %create matrix

for i=1:n
    att(i,1)=Xc-r(i,1);    % find x attractive potential
    att(i,2)=Yc-r(i,2);     % find y attractive potential
end

attctrl= Katt* att;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% attractive potential of individual points to target
attargetctrl=zeros(n,m);

for i=1:n
    Xattar=(Xt-r(i,1));
    Yattar=(Yt-r(i,2));
    theta_target = atan2(Yattar,Xattar); % angle of attracion to target

    l=[Xattar ; Yattar];  % vectorto show length of attractive vector to target
    Mag_l=sqrt(l'*l);   %magnitude of length

    attargetctrl(i,1)= (Mag_l)*cos(theta_target);
    attargetctrl(i,2)= (Mag_l)*sin(theta_target);
end
attargetctrl=Kattarget*attargetctrl;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%repulsive potential away from each other
rep=zeros(n,2);
```

```
qa=zeros(1,2);
qb=zeros(1,2);

for i=1:(n-1)       % Matrix to write the repulsive potential
    for j=2:n       % by reducing the number of times each point is added, we run
the matrix 1/2 the required times
       if i==j      % if i=j, same point, thus discard
       else
           Xa=r(i,1);
            Ya=r(i,2);
            Xb=r(j,1);
            Yb=r(j,2);
           dist_ab=sqrt(((Xa-Xb)^2) +((Ya-Yb)^2));  % distance from each point
           theta_ba=atan2(Ya-Yb, Xa-Xb);
           qa=[(1/dist_ab)*cos(theta_ba),(1/dist_ab)*sin(theta_ba)];
           rep(i,:)=rep(i,:)+qa;

           qb= [(1/dist_ab)*cos(theta_ba+ pi),(1/dist_ab)*sin(theta_ba+pi)];
           rep(j,:)=rep(j,:)+qb;
       end
   end
end

repctrl= Krep*rep;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%repulsive control away multiple obstaclesfrom the obstacle
reptarget=zeros(n,m);

totalrepobst=zeros(n,m);
[w,v]=size(obst);

for k=1:w
    for i=1:n
        theta_obst = atan2(r(i,2)-obst(k,2),r(i,1)-obst(k,1)); % angle of repulsion
form obstacle
        Xdist=(r(i,1)-obst(k,1)-R*cos(theta_obst));
        Ydist=(r(i,2)-obst(k,2)-R*sin(theta_obst));

        L=[Xdist ; Ydist];  % vectorto show length of repulsion vector to obstacle

        Mag_L=sqrt(L'*L);   %magnitude of length

        if (Mag_L<(rho-R))
            repobst(i,1)= ((1/Mag_L)*cos(theta_obst))-((1/(rho-R))*cos(theta_obst));
            repobst(i,2)= ((1/Mag_L)*sin(theta_obst))-((1/(rho-R))*sin(theta_obst));
        else
            repobst(i,1)= 0;
            repobst(i,2)=0;
        end
    end
totalrepobst=totalrepobst+repobst;
end
reptarget=Kreptar*totalrepobst;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
q=zeros(n,m);
qspec=zeros((n*m),1);

 q = attctrl + repctrl + attargetctrl + reptarget;   %sum of attractive and
repulsive controller
 counter=0;
    for i=1:n
        for j=1:m
            counter=counter+1;
            qspec(counter,1)=q(i,j);
        end
    end

 ldot=qspec;
```

```
% Appendix D
%Yong Tan
%Trident project
%behavior based: program to run simulation with multiple robots

global obst% list obstacles as a global variable

R=0.5;
rho=1+R;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%create obstacles insimulation

obst=obst1;   %using preset data points as references for sqobst
robot=[0 0 6 0 3 4 9 7 6 5 0 1]';
[t,p]=ode45('multiobst', [0,60],robot);    %run program for specified robot points

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[n,m]=size(robot); %converting the column vector into a 3*2 vector
count=n/2;
newrobot=zeros(count,2);
A=0;
for i=1:n           %conversion of a 6 by 1 column vector into a 3 by 2
    if rem(i,2)==0 %if i is even move to 2nd column
          y=i/2;,
        newrobot(y,2)=robot(i,1);
    else            %if i is odd, move to 1st column
        y=(i+1)/2;
      newrobot(y,1)=robot(i,1);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[w,m]=size(newrobot);

for i=1:w
    plot(newrobot(i,1),newrobot(i,2),'+');
end

for i=1:n
    if rem(i,2)==0

    else
        next=i+1;
        plot(p(:,i),p(:,next)) ;
    end
end

title( '3 robots movig and avoiding obstacle');
 xlabel('x-coodinates');
 ylabel('y-coodinates');
 grid;

 figure(2);
 %end

 plot3(t,p(:,1),p(:,2),':');
 hold on;
 plot3(t,p(:,3),p(:,4),'p--');
 plot3(t,p(:,5),p(:,6),'o-');
 plot3(t,p(:,7),p(:,8),'y-.');
 plot3(t,p(:,9),p(:,10),'m-');
 plot3(t,p(:,11),p(:,12),'g-*');
 title(' individual tracks taken by each robot');
 xlabel('time-coodinates');
 ylabel('x-coodinates');
 zlabel('y-coodinates');
 grid;
 legend('Track of 1st robot','Track of 2nd robot','Track of 3rd robot','Track of 4th
robot','Track of 5th robot','Track of 6th robot')
```

```
%Appendix E1
% Yong Tan
% Trident Scholar Project

function ob=obst()
%  This file sets up vetices
%  for a number of obstacles
%  The C-space equivalent is
%  drawn at runtime.
global NUM


ans = input('Do you wish to generate new obstacles (y/n)? ', 's');
if (ans == 'y')
   NUM = [];
   N = input('How many obstacles do you wish to generate? ');
   figure(1);
   clf;
   axis([0 20 0 20]);
   hold;
   OBST = [];
   for i = 1:N
      temp = ['Obstacle #', num2str(i), ':  Click on each vertex in CLOCKWISE order.
Double-click final vertex.'];
      disp(temp);
      input('Press ENTER when ready');

      [xo, yo] = getpts;
      [n, m] = size(OBST);
      xo = [xo; xo(1)];
      yo = [yo; yo(1)];
      fill(xo, yo, 'r');
      l_new = length(xo);
      if (l_new < 3)
         error('An obstacle must have at least 3 vertices');
      end
      NUM(i) = l_new - 1;
      if (m < l_new)&(m > 0)
         OBST = [OBST, zeros(n, l_new - m)];
      end
      if (m > l_new)
         xo = [xo; zeros(m-l_new, 1)];
         yo = [yo; zeros(m-l_new, 1)];
      end
      OBST = [OBST; xo'; yo'];
   end
else
   ans = input('Choose: 1 Last set of obstacles, 2 Standard: ');
   if (ans == 2)
                    NUM = [4 3 3];  % Number of vertices for each
                       obstacle

                    O1 = [1 1;      %  that is an 'oh1', not a
                                  'zero1'
                    1.1 2;
                    2 2.1;
                    2.1 1.1;];

                    O1 = [O1; O1(1, :)];

                    O2 = [4 4;
                    6 2;
                    4.1 2.1];

                    O2 = [O2; O2(1, :)];
                    O3 = [6 6;
                    8 6.1;
                    8.1 4];
                              O3 = [O3; O3(1, :)];
                              OBST = [O1'; O2' [0 0]'; O3' [0 0]']
                    end
                    figure(1);
                    clf;
             axis([0 20 0 20]);
                    hold;
          end
          ob=OBST;
```

```
%Appendix E2
%Yong Tan
%Trident project
%behavior based: program to run simulation with multiple robots

global obs% list obstacles as a global variable
global target
global NUM

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%create obstacles in simulation

obs=obst;   %create obstacles in plot
[row,col]=size(obs);

 %matrix to store command positions
pts=inputpoints; %Get points
[roww,coll]=size(pts);
robot=zeros((roww-2),1);

for i=1:(roww-2)
     robot(i,1)=pts(i,1);
end
target=zeros(1,2);
target(1,1)=pts(roww-1,1);
target(1,2)=pts(roww,1);

[t,p]=ode45('avoidobst', [0,60],robot); %run program for specified robot points


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[n,m]=size(robot); %converting the column vector into a 3*2 vector
count=n/2;
newrobot=zeros(count,2);

for i=1:n           %conversion of a 6 by 1 column vector into a 3 by 2
    if rem(i,2)==0 %if i is even move to 2nd column
          y=i/2;
        newrobot(y,2)=robot(i,1);
    else            %if i is odd, move to 1st column
        y=(i+1)/2;
      newrobot(y,1)=robot(i,1);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plotting position of the new robots
[w,m]=size(newrobot);

for i=1:w
    plot(newrobot(i,1),newrobot(i,2),'+');
end
% plotting positions of the paths
for i=1:n
    if rem(i,2)==0
    else
        next=i+1;
        plot(p(:,i),p(:,next)) ;
    end
end

title( '3 robots moving and avoiding new obstacle');
 xlabel('x-coodinates');
 ylabel('y-coodinates');
 grid;

[row,col]=size(p);

robot=p(row,:)';

%end

 figure(2);

 plot3(t,p(:,1),p(:,2),'r:');
 hold on;
 plot3(t,p(:,3),p(:,4),'g--');
 plot3(t,p(:,5),p(:,6),'c-');
 title(' individual tracks taken by each robot in new obstacle');
 xlabel('time-coodinates');
 ylabel('x-coodinates');
 zlabel('y-coodinates');
 grid;
 legend('Track of 1st robot','Track of 2nd robot','Track of 3rd robot')
```

```
%Appendix E3
% Yong Tan
% Trident Scholar Project
%edited from Prof Bishop

%  This function takes a robot position (x,y)
%  and a vector of obstacle vertices OBST as
%  well as a vector NUM that contains the number
%  of vertices per obstacle.  It returns the
%  repulsive force vector Frep
% No obstacle should have two vertices on the
% a horizontal line.

function Frep = repulse(x, y, OBST, NUM)

eta = 0.5;          % Coefficient for repulsive potential
rho = 1.5;                              % Maximum distance for repulsive field of
obstacles
min_dist = 100000;    % initialize to 'infinity'
for i = 1:length(NUM)
   for j = 1:NUM(i)
         x1 = OBST(2*i-1, j);              % current vertex x
         x2 = OBST(2*i-1, j+1);            % next vertex x
         y1 = OBST(2*i, j);                % current vertex y
         y2 = OBST(2*i, j+1);              % next vertex y
         if (x2 ~= x1)
            slope = (y2-y1)/(x2-x1);          % side slope
         else
            slope = Inf;
        end
         y_inter = y1 - x1*slope;          % side y-intercept
         A = slope;
         B = -1;
         C = y_inter;
         dist = abs((A*x + B*y + C)/sqrt(A*A + B*B));    % distance from robot (x,y)
to side
         if (dist < min_dist)
            angle_edge = atan2(y2 - y1, x2 - x1);        % angle of edge
            angle_test = angle_edge + pi/2;              % angle of force vector
            %pt = [x1 + cos(angle_test); y1 + sin(angle_test)];  %
            pob = [x + cos(angle_test + pi)*dist;
                   y + sin(angle_test + pi)*dist;];   % closest point on extended
edge
            d1 = norm([pob(1) - x1; pob(2) - y1]);    % distance from extended edge
intercept
                                                      % to current vertex
            d2 = norm([pob(1) - x2; pob(2) - y2]);    % distance from extended edge
intercept
                                                      % to current vertex
            l = norm([x1 - x2; y1 - y2]);             % distance from vertex to
vertex
            if (max([d1, d2]) >  l)                   % extended edge intercept NOT
between
                                                      % vertices... do nothing
            else
               min_dist = dist;                       % closest point is between
vertices
               angle_push = angle_test;               % push away from edge
            end
         end
         dist = norm([x - x1, y - y1]);               % distance to vertex
         if (dist < min_dist)                         % closest point is at vertex
            min_dist = dist;
            angle_push = atan2(y - y1, x - x1);       % push away from vertex
         end
      end
   end
   if (min_dist < rho)                              % inside radius of influence
      Frep = [cos(angle_push) sin(angle_push)]*eta*(1/(min_dist-0.25) - 1/(rho-
0.25))*1/(min_dist^2);
   else
      Frep = [0 0];                                      % outside radius of
influence
   end
```

```
% Appendix F
%Yong Tan
%Trident project
%behavior based: program to run simulation with multiple robots

global OBST% list obstacles as a global variable
global target
global NUM
global MINES

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%create obstacles in simulation

obstone;%create obstacles in plot
MINES=mines;
OBST=ob;
[row,col]=size(OBST);

 %matrix to store command positions
pts=inputpoints; %Get points
[row,col]=size(pts);
robots=zeros((row-2),1);

for i=1:(row-2)
     robot(i,1)=pts(i,1);

 end
target=zeros(1,2);
target(1,1)=pts(row-1,1);
target(1,2)=pts(row,1);

[t,p]=ode45('findmine', [0,50],robot); %run program for specified robot points


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[n,m]=size(robot); %converting the column vector into a 3*2 vector
count=n/2;
newrobot=zeros(count,2);

for i=1:n           %conversion of a 6 by 1 column vector into a 3 by 2
    if rem(i,2)==0 %if i is even move to 2nd column
          y=i/2;
        newrobot(y,2)=robot(i,1);
    else            %if i is odd, move to 1st column
        y=(i+1)/2;
      newrobot(y,1)=robot(i,1);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[w,m]=size(newrobot);

for i=1:w
    plot(newrobot(i,1),newrobot(i,2),'+');
end
for i=1:n
    if rem(i,2)==0
    else
        next=i+1;
        plot(p(:,i),p(:,next)) ;
    end
end

title( '3 robots movig and avoiding obstacle');
 xlabel('x-coodinates');
 ylabel('y-coodinates');


[row,col]=size(p);
robot=p(row,:)';
%end

%  figure(2);
%
%  plot3(t,p(:,1),p(:,2),':');
%  hold on;
%  plot3(t,p(:,3),p(:,4),'--');
%  plot3(t,p(:,5),p(:,6),'-');
 %title(' individual tracks taken by each robot');
 %xlabel('time-coodinates');
 %ylabel('x-coodinates');
 %zlabel('y-coodinates');
 %grid;
 %legend('Track of 1st robot','Track of 2nd robot','Track of 3rd robot')
```

```
%Appendix G
% Yong Tan
% Trident Scholar Project

% This program is used to determine how the subsumption architecture affects the
movement of the
% robot since it is rule based and subjected to its location within the hierachy.

function pdot= subsumption_wp(t,p)
%list constants

global T
global brownian
global leg
global OBST
global ROBOT
global INITIAL_ROBOT
global WAYPTS
global NUM
global MINES
global counter

%initiliase gains
Katt=2; % attractive constant
Krep=5; % repulsion constant
Kreptar=15;     % repulsion constant from obstacle

R=0.3;  %radius of obstacle
rl=0.2;  %sensing radius
rho=R+0.2;  %radius of influence of robot

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[n,m]=size(p); %converting the column vector into a 3*2 vector
count=n/2;
ROBOT=zeros(count,2);
for i=1:n           %conversion of a 6 by 1 column vector into a 3 by 2
    if rem(i,2)==0  %if i is even move to 2nd column
        y=i/2;
        ROBOT(y,2)=p(i,1);
    else            %if i is odd, move to 1st column
        y=(i+1)/2;
        ROBOT(y,1)=p(i,1);
    end
end
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% begin subsumption architecture,
% highest order, avoid obstacle; detect mine; gathering; random motion

% matrix to store resultant vector
resultant=zeros(size(ROBOT));


%1. find closest mines
Notarget=disttarget(MINES,ROBOT);

%2. find shortest distance
minobstdist=calculatemindist(OBST,ROBOT,NUM);
Frep=[];

%3. find separation of robots
[n, m]=size(ROBOT);
 robodist=robotdist(ROBOT);
 close_d=min(robodist);
count=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %****************subsumption part of codes*******************************
for i=1:n
    position=0;
```

```
    % for every robot calculate closest mine
    %   1. HIghest order: Avoid mines, tag them and store positions
    if (Notarget(i,1)<=0.3)
        % repulse away from target and change them into a potential
            position=(Notarget(i,2));
             %find the index of the closest mine
            counter=counter+1;
            if (counter>5);

                plot(MINES(position,1),MINES(position,2),'gd');
                counter=0;
            else
            end
            [rows,cols]=size(MINES);    % finding size of mine
            totalrep=zeros(n, m);
            repmine=zeros(n, m);

            theta_mine = atan2(ROBOT(i,2)-MINES(position,2),ROBOT(i,1)-
MINES(position,1)); % angle of repulsion form obstacle

            Xdist=(ROBOT(i,1)-MINES(position,1)-rl*cos(theta_mine));
            Ydist=(ROBOT(i,2)-MINES(position,2)-rl*sin(theta_mine));

            % vectorto show length of repulsion vector to obstacle
            Mag_L=norm([Xdist ; Ydist]); %normalize vector for magnitude of length

            repmine(i,1)= ((1/Mag_L)*cos(theta_mine))-((1/(rho-
rl))*cos(theta_mine));
            repmine(i,2)= ((1/Mag_L)*sin(theta_mine))-((1/(rho-
rl))*sin(theta_mine));

            random(i,:)= randn(1,2);
            totalrep(i,:)=Kreptar*(repmine(i,:)+ random(i,:));
            resultant(i,:)=totalrep(i,:);

      else
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% #2 AVoid closest obstacles
            if (minobstdist<0.8)

             %repulsive control away from  multiple obstacles
             temp1=ROBOT(i,1);   %  set first x coor of robot to temp1
             temp2=ROBOT(i,2);    %  set first y coor of robot to temp2
             Frep(i,:) = repulse(temp1, temp2, OBST, NUM);    % find repulsive force
away from obstacles

             %random_var=0.3*randn(1,2);  %add random variable
             Repelobst(i,:)=Frep(i,:);%+brownian;  %create final  vector
             resultant(i,:)=Repelobst(i,:);      % store in new vector

            % disp('Avoid')

             else

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                if (close_d>3)
                     % if the distance between robots were greater than 5,less than
20,
                     %ignore, computational error
                     resultant=zeros(n,m);

                 elseif ((close_d>1) & (close_d<=2))
                     % find centroid of all 3 robots
                     [n, m]=size(ROBOT); % size of robo vector
                     A=sum(ROBOT,1);     % sum of all x,y coordinates
                     Xc= A(1,1)/n;     % x-coordinate of centroid of all robots
                     Yc=A(1,2)/n;        % y-coordinate of centroid of all robots
                     %Attractive control
                     att=zeros(n,m);  %create matrix
                     Mag_change=norm([Xc-ROBOT(i,1),Yc-ROBOT(i,2)]);
                     theta_att=atan2(Yc-ROBOT(i,2),Xc-ROBOT(i,1));

                     att(i,1)=Mag_change*cos(theta_att);    % find x attractive
potential
                     att(i,2)=Mag_change*sin(theta_att);     % find y attractive
potential

                     attctrl(i,:)= Katt* att(i,:);         % attractive potential
towards each other
                     resultant(i,:)=attctrl(i,:);          % assigning matix to
resultant
             %       disp('gather')

                 elseif ((close_d<0.5) & (close_d>0))
                     %repulsive potential away from each other
                     rep=zeros(n,2);
                     qa=[];  %value to store repulsive vector to be added in one
direction
                     qb=[];  %value to store repulsive vector to be added in other
direction(+pi)
                     if (i<= (n-1))      % Matrix to write the repulsive potential
                         for k=2:n       % by reducing the number of times each
point is added, we run the matrix 1/2 the required times
                             if i==k      % if i=j, same point, thus discard
                             else
                                 Xa=ROBOT(i,1);
                                 Ya=ROBOT(i,2);
                                 Xb=ROBOT(k,1);
                                 Yb=ROBOT(k,2);

                                 dist_ab=sqrt(((Xa-Xb)^2) +((Ya-Yb)^2));  % distance
from each point

                                 theta_ba=atan2(Ya-Yb, Xa-Xb);

qa=[(1/dist_ab)*cos(theta_ba),(1/dist_ab)*sin(theta_ba)];
                                 rep(i,:)=rep(i,:)+qa; % vector from one robot to
another

                                 qb= [(1/dist_ab)*cos(theta_ba+
pi),(1/dist_ab)*sin(theta_ba+pi)];
                                 rep(k,:)=rep(k,:)+qb;    % reciprocal vector from
one robot to another
                             end
                         end
                     else
                     end
                     repctrl(i,:)= Krep*rep(i,:);
                     resultant(i,:)=repctrl(i,:);
              %       disp('spread')

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %successful code for random motion
        %Create random points
                elseif( (close_d>0.3) & (close_d<1.0))
                    %separate x,y components into different matrices
                     [roww,col]=size(WAYPTS) ;
                      brownian = randn(1,2);

                     x_d = WAYPTS(leg, 1);
                     y_d = WAYPTS(leg, 2);

                     [n,m]=size(ROBOT);

                     att_d=zeros(1,m);
```

```
                     x = ROBOT(i,1);
                     y = ROBOT(i,2);

                     att_d(i,:) = [x_d - x, y_d - y]; % attractive vector towards
desired position
                     magnitude=sqrt((x_d - x)^2+ (y_d - y)^2);
                     if (magnitude > 1)
                         att_d(i,:) = (att_d(i,:)+ brownian)/magnitude;   %
normalize vector, <1
                     else
                         if (magnitude < 0.1) % if position is <0.1 to desired
waypoint
                             if (leg < length(WAYPTS))
                                leg = leg + 1  % change waypoint
                            else
                                leg = leg;
                            end
                        end
                     end

                     if (t > T)
                         T = T + 0.5;
                         resultant(i,:) = att_d(i,:) + 3*brownian;
                     else
                        resultant(i,:) = att_d(i,:);
                     end
                     resultant;
               %     disp('brownian')
                   end
               end
           end
       end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[n,m]=size(resultant);% convert the n,m matrix = n*m,1 matrix
counter=0;
qspec=zeros(n*m,1);

for i=1:n
    for j=1:m
        counter=counter+1;
        qspec(counter,1)=resultant(i,j);
    end
end
pdot=qspec;
```

```
% Appendix H
%Yong Tan
% Trident Scholar Project
% This file runs the program, improvedstat, to demonstrate the effects of 6 robots
% performing obstace avoidance while maintaining the mean and variance of the swarm
global mudx vardx mudxf vardxf % global variables of changes in mean and variance of
x
global mudy vardy mudyf vardyf % global variables of changes in mean and variance of
y
global Km Kv Ka % motion, velocity and acceleration constants
global R RAD RRAD Radmine% xoffset, yoffset, mean offset,
global Xobst Yobst Cx Cy Cmx Cmy num
global leg tf Xmine Ymine
global traj_hist

traj_hist = [];
num=6;             % number of robots
R = 1;           % Radius of obstacle
mudxf = [2 12 12];
mudyf = [10 10 0];
vardxf = [5 3 5];
vardyf = [3 5 3];
Km = 3;             % 3     motion constant
Kv = 3;             % 3     velocity constant
Ka = 5;   % 1         % acceleration constant
RAD = 2;            % minimum separtion distance between robots
RRAD = 0.5;  % 1      % minimum distance between individual robots to obstacle
Radmine=0.3;
record = 0;
Xobst= [3 6 10 ];   %6
Yobst = [4 8 4 ];   %8
Xmine = [ 7 10];
Ymine = [ 10 6];
leg=1;

tf =18; % final time in sec
tspan = [0 tf]; % time span from 0-20 s

q0=[0 1 2 3 4 3.8105 0 2 0 2 1 2.5];

mudx = mean(q0(1:num));
mudy = mean(q0(num+1: 2*num));
vardx = var(q0(1:num));
vardy = var(q0(num+1: 2*num));

for zz = 1 : 3
    tfl = tf/3 + tf/3*(zz-1);
    til = tfl - tf/3;
    %polynomial interpolation
    if (zz > 1)
        mudx = mudxf(zz-1);
        mudy = mudyf(zz-1);
        vardx = vardxf(zz-1);
        vardy = vardyf(zz-1);
    end
    Cx = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[vardx; 0; vardxf(zz); 0];
    Cy = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[vardy; 0; vardyf(zz); 0];

    Cmx = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[mudx; 0; mudxf(zz); 0];
    Cmy = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[mudy; 0; mudyf(zz); 0];
    traj_hist = [traj_hist; Cx' Cy' Cmx' Cmy' tfl];
end

options = odeset('RelTol', 1e-12, 'AbsTol', 1e-10*[ones( 2*num,1)]');
```

```
[t, q] = ode45('improvedstat', tspan, q0); % calculates the x, y position based on
time

figure(1);
clf(1);
    nextT = 0;
     axis([-8 18 -3 13]);
    axis('manual');
    xlabel('X (m)');
    ylabel('Y (m)');
    title('Motion of Platoon Units with the Statistical Controller');
    hold;
    theta = 0:0.01:2*pi;
    [a,b]=size ( Xobst);
    for (i=1:b)
        obst = [R*cos(theta)+Xobst(i); R*sin(theta)+Yobst(i)]; % showing the
obstacle
        plot(obst(1, :), obst(2, :));
    end
     [a,c]=size (Xmine);
    for (i=1:c)
        mine = [Radmine*cos(theta)+Xmine(i); Radmine*sin(theta)+Ymine(i)]; % showing
the obstacle
        plot(mine(1, :), mine(2, :));
    end
    robot = [-0.03 -0.03; 0.03 -0.03; 0.03 0.03; -0.03 0.03];% using little squares
to represent the  robots
    clear M;
   col = ['g', 'r', 'b', 'c', 'm', 'y'];
    for (i = 1:length(t))
        if t(i) >= nextT     % everytime, T, draw and color robot..
            for (j = 1:num)
                fill(robot(:, 1)+q(i, j), robot(:, 2)+q(i, j+num), col(j)); % fill
in shape and color of the simulated robots.
            end
            nextT = nextT + 0.05; % at next time T
        end
        M(i) = getframe;
        % pause
    end
     axis('equal');
    axis([min(min(q(:, 1:6)))-1, max(max(q(:,1:6)))+1,min(min(q(:, 7:12)))-1,
max(max(q(:,7:12)))+1]);
    hold off;

    %labelling the plot
    for h=1:(length(Xobst))
        gtext(['Obstacle #',num2str(h)]);
    end
    for p=1:(length(Xmine))
        gtext(['Mine #',num2str(p)]);
    end
    for j=1:num
        gtext(['Unit #',num2str(j)]);
    end

figure(2)
clf(2);
for i = 1:6
    subplot(3,2,i);
    axis([0 15 0 15]);
    axis('manual');
    title(['Unit #', num2str(i)]);
    ylabel('Y (m)');
    if i>4
    xlabel('X (m)');
    end
    hold
    plot(q(:, 1+(i-1)), q(:, 1+num+(i-1))); % plot individual paths
    [a,b]=size ( Xobst); %plot individual obstacles
    theta = 0:0.01:2*pi;
    for (i=1:b)
        obst = [R*cos(theta)+Xobst(i); R*sin(theta)+Yobst(i)]; % showing the
obstacle
        plot(obst(1, :), obst(2, :));
    end
    [c,d]=size ( Xmine); % plot individual mines
    for (i=1:d)
        mine = [Radmine*cos(theta)+Xmine(i); Radmine*sin(theta)+Ymine(i)]; % showing
the obstacle
        plot(mine(1, :), mine(2, :));
    end
    hold off
end
xmean = mean(q(:, 1:num)')';
ymean = mean(q(:, num+1:2*num)')';
xvar = var(q(:, 1:num)')';
yvar = var(q(:, num+1:2*num)')';

figure(3)
clf(3);
subplot(2,2,1)
plot(t, xmean,'b:');
hold;
[xx, qq] = size(traj_hist);
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cmx = traj_hist(i, 9:12)';
    plot(T, Cmx(1)*T.^3 + Cmx(2)*T.^2 + Cmx(3)*T + Cmx(4), 'r-');
    title('Platoon Tracking Mean X Errors');
    xlabel('Time (s)');
    ylabel('Mean X (m)');
end
hold
subplot(2,2,2)
plot(t, ymean,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cmy = traj_hist(i, 13:16)';
    plot(T, Cmy(1)*T.^3 + Cmy(2)*T.^2 + Cmy(3)*T + Cmy(4), 'r-');
      title('Platoon Tracking Mean Y Errors');
    xlabel('Time (s)');
    ylabel('Mean Y (m)');
end
hold
subplot(2,2,3)
plot(t, xvar,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cx = traj_hist(i, 1:4)';
    plot(T, Cx(1)*T.^3 + Cx(2)*T.^2 + Cx(3)*T + Cx(4), 'r-');
    title('Platoon Tracking Variance X Errors');
    xlabel('Time (s)');
    ylabel('Variance X ');
end
hold
subplot(2,2,4)
plot(t, yvar,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cy = traj_hist(i, 5:8)';
    plot(T, Cy(1)*T.^3 + Cy(2)*T.^2 + Cy(3)*T + Cy(4), 'r-');
   title('Platoon Tracking Variance Y Errors');
    xlabel('Time (s)');
    ylabel('Variance Y ');
end
legend('actual values', 'desired values');

hold
%create movie avi
%figure(4)
%movie2avi(M,'test2.avi','FPS', 10,'Quality',99);

collision = 0;
min_dist = 1000;
for i = 1:length(q)
    X = [];
    for j = 1:num
      X = [X, [q(i, j); q(i, j+num)]];
    end
    for k = 1:2
        for l = k+1:num
            dist = X(:, k) - X(:, l);
            dist = sqrt(dist'*dist);
            if (dist < min_dist)
                min_dist = dist;
            end
            if (dist < 0.06*sqrt(2))
                error('Collison!');
            end
        end
    end
end
```

```
%Appendix I
% Yong Tan
%Tridnet Scholar project
function qdot = plain6test(t, q)
% function show the statistical controllers work in an
%environment with mines and obstacles.
global Km Kv Ka
global mudx vardx mudxf vardxf mux
global mudy vardy mudyf vardyf muy
global R RAD RRAD Radmine
global Xobst Yobst RADOBST Cx Cy Cmx Cmy num
global leg tf Xmine Ymine
global traj_hist
%C terms are used to control tragectories.
%Cx,Cy are variance tragectory controls
%Cmx,Cmy are mean trajectory controls

feedforward = 1;% drive robots forward in jacobian
%check distance between robots and the mines,
findmine=checkdist(q, Xmine, Ymine, Radmine); %find cloest dist and position
closestdistmine=findmine(1,:);    %find closest mine dist
posmine=findmine(2,:);            % position of mine

%check distance between robots and the obstacles
findobst=checkdist(q, Xobst, Yobst, R);
closestobst=findobst(1,:);
posobst=findobst(2,:);
mux = mean(q(1:num)); % x=q... represent the states x, y, x= 1:4, y 5:8
muy = mean(q(num+1:2*num)); % x(5:8 ) represent the y values
varx = var(q(1:num)); % variance of x
vary = var(q(num+1:2*num)); % variance of y
x_d = mudxf(leg);
y_d = mudyf(leg);

  magnitude=sqrt((x_d - mux)^2+ (y_d - muy)^2); % distance from centroid of swarm to
waypoint
   %waypoint changes if the time is
   for mm = 1:length(mudxf)
       if (t < traj_hist(mm, 17))
           break;
       end
   end
   leg = mm;
    %calculated trajectory projection based on each leg
    Cx = traj_hist(leg, 1:4)';
    Cy = traj_hist(leg, 5:8)';
    Cmx = traj_hist(leg, 9:12)';
    Cmy = traj_hist(leg, 13:16)';

    mudx = Cmx'*[t^3 t^2 t 1]'; %desired x mean position
    mudy = Cmy'*[t^3 t^2 t 1]'; %desired y mean position
    dmudx = Cmx'*[3*t^2 2*t 1 0]';  %desired vx mean velocity, derivatives
    dmudy = Cmy'*[3*t^2 2*t 1 0]';  %desired vy mean velocity, derivatives

    varxt = Cx'*[t^3 t^2 t 1]'; %desired x position variance
    varyt = Cy'*[t^3 t^2 t 1]'; %desired y position variance
    dvarxt = Cx'*[3*t^2 2*t 1 0]';  %desired vx velocity variance, derivatives
    dvaryt = Cy'*[3*t^2 2*t 1 0]';  %desired vy velocity variance, derivatives
  t
    J = [1/num*(ones(1,num)); 2/(num-1)*((q(1:num))'-mux)]; % matrix defined by
state and task function: difference from mean in x
    Jy = [J(1, :); 2/(num-1)*(q(num+1:2*num)' - muy)]; %alternate matrix defined by
state -diff from mean in y
    J = [J zeros(2, num); zeros(2, num) Jy]; %create matrix of no. by 8 to discribe
x and y characteristics for no. roobots
    Jp = J'*(J*J')^(-1);        % Moore-Penrose pseudo inverse T ( resultant- 3by 8
matrix)
```

```
    % attractive vectors to keep swarm moving forward
    xa = [Km*(mudx - mux)+ feedforward*dmudx; Kv*(varxt - varx) + dvarxt];  %the
changes in mean and variance in x with feedforward term
    xay = [Km*(mudy - muy)+ feedforward*dmudy; Kv*(varyt - vary) + dvaryt]; %the
changes in mean and variance in y with feedforward term
    xd = [xa; xay];%

%*************** generating subsumption in a single robot
***************************
% function to calculate repulsion from each robot
irrep = zeros(2*num, 1);
for (i = 1:num)

   q1=q(i);
   q2=q(i+num);

   Xactm=Xmine(posmine(i));            % actual X position of mine
   Yactm=Ymine(posmine(i));              % actual Y position of mine

   repelmine=findreppot(q1,q2,Xactm,Yactm,Radmine); % repulsive vectors from mines
   pot(i)=repelmine(1);
   pot(i+num)=repelmine(2);

   % repulsive vectors away from obstacles
   Xactobst=Xobst(posobst(i)); % designating the mines
   Yactobst=Yobst(posobst(i));

   repobst=findreppot(q1,q2,Xactobst,Yactobst,R); % repulsive vector from obstacles
   potx(i)=repobst(1);
   potx(i+num)=repobst(2);

   totalpot(i)=pot(i)+potx(i);
   totalpot(i+num)=pot(i+num)+potx(i+num);
   min_dis = 10000;
   if ( i <= 0)
   for j = 1:num
       if (i ~= j)
           dis = sqrt((q(i) - q(j))^2 + (q(i+num) - q(j+num))^2) - 0.2;
           if ((dis < min_dis)& (dis < RRAD))
              angle = atan2(q(j+num) - q(i+num), q(j) - q(i));
              min_dis = dis;
          end
      end
  end
  if (min_dis < RRAD)
      irrep(i) = cos(angle)*(1/(min_dis) - 1/(RRAD));  % added 1/RAD term  repulsive
potential in x
      irrep(i+num) = sin(angle)*(1/(min_dis) - 1/(RRAD));  % added
  end
  totalpot(i) = totalpot(i) + 0.5*irrep(i);
  totalpot(i+num) = totalpot(i+num) + 0.5*irrep(i+num);
    end
end

qdot = Jp*xd + Ka*(eye(2*num) - Jp*J)*(totalpot');
```

```
% Appendix J
% Yong Tan, Trident Scholar Project
% This file  shows how the hybrid controller was implemented with subsumption
implemented in the
%statistical controller
global mudx vardx mudxf vardxf % global variables of changes in mean and variance of
x
global mudy vardy mudyf vardyf % global variables of changes in mean and variance of
y
global Km Kv Ka % motion, velocity and acceleration constants
global R RAD RRAD Radmine% xoffset, yoffset, mean offset,
global Xobst Yobst Cx Cy Cmx Cmy num
global leg tf Xmine Ymine
global traj_hist
global A E F
global avoid_hist XR YR
global xydifferences

traj_hist = [];
num=6;             % number of robots
avoid_hist = zeros(num, 1);
R = 1;           % Radius of obstacle
% XOFF = 3;           % x offset fixed
% YOFF = 1;           % y offset fixed
mudxf = [2 12 12];
mudyf = [10 10 0];
vardxf = [5 3 5];
vardyf = [3 5 3];
Km = 3;             % 3     motion constant
Kv = 3;             % 3     velocity constant
Ka = 5;   % 1         % acceleration constant
RAD = 2;            % minimum separtion distance between robots
RRAD = 0.5;  % 1      % minimum distance between individual robots to obstacle
Radmine=0.3;
record = 0;
Xobst= [3 6 10 ];
Yobst = [4 8 4 ];
Xmine = [ 7 10];
Ymine = [ 10 6];
leg=1;

tf =18; % final time in sec
tspan = [0 tf]; % time span from 0-20 s

q0=[0 1 2 3 4 3.8105 0 2 0 2 1 2.5];

XR = [     0.9501    0.4103    0.8462    0.1509    0.8385  0.1365
    0.2311    0.8936    0.5252    0.6979    0.5681  0.0118
    0.6068    0.0579    0.2026    0.3784    0.3704  0.8939
    0.4860    0.3529    0.6721    0.8600    0.7027  0.1991
    0.8913    0.8132    0.8381    0.8537    0.5466  0.2987
    0.7621    0.0099    0.0196    0.5936    0.4449  0.6614
    0.4565    0.1389    0.6813    0.4966    0.6946  0.2844
    0.0185    0.2028    0.3795    0.8998    0.6213  0.4692
    0.8214    0.1987    0.8318    0.8216    0.7948  0.0648
    0.4447    0.6038    0.5028    0.6449    0.9568  0.9883
    0.6154    0.2722    0.7095    0.8180    0.5226  0.5828
    0.7919    0.1988    0.4289    0.6602    0.8801  0.4235
    0.9218    0.0153    0.3046    0.3420    0.1730  0.5155
    0.7382    0.7468    0.1897    0.2897    0.9797  0.3340
    0.1763    0.4451    0.1934    0.3412    0.2714  0.4329
    0.4057    0.9318    0.6822    0.5341    0.2523  0.2259
    0.9355    0.4660    0.3028    0.7271    0.8757  0.5798
    0.9169    0.4186    0.5417    0.3093    0.7373  0.7600];

YR =[    0.9669    0.4608    0.4199    0.6273    0.7036 0.7009
    0.6649    0.4574    0.7537    0.6991    0.4850  0.9623
    0.8704    0.4507    0.7939    0.3972    0.1146  0.7505
    0.0099    0.4122    0.9200    0.4136    0.6649  0.7400
    0.1370    0.9016    0.8447    0.6552    0.3654  0.4319
    0.8188    0.0056    0.3678    0.8376    0.1400  0.6343
    0.4302    0.2974    0.6208    0.3716    0.5668  0.8030
    0.8903    0.0492    0.7313    0.4253    0.8230  0.0839
    0.7349    0.6932    0.1939    0.5947    0.6739   0.9455
    0.6873    0.6501    0.9048    0.5657    0.9994  0.9159
    0.3461    0.9830    0.5692    0.7165    0.9616  0.6020
    0.1660    0.5527    0.6318    0.5113    0.0589  0.2536
    0.1556    0.4001    0.2344    0.7764    0.3603  0.8735
    0.1911    0.1988    0.5488    0.4893    0.5485  0.5134
    0.4225    0.6252    0.9316    0.1859    0.2618  0.7327
    0.8560    0.7334    0.3352    0.7006    0.5973  0.4222
    0.4902    0.3759    0.6555    0.9827    0.0493  0.9614
    0.8159    0.0099    0.3919    0.8066    0.5711  0.0721];

mudx = mean(q0(1:num)); % mean  x at starting time
mudy = mean(q0(num+1: 2*num));% mean y at starting time
vardx = var(q0(1:num)); %initial varx
vardy = var(q0(num+1: 2*num));

% calculate trajetory history for each leg!!!!! % to be used in function
for zz = 1 : 3 %number of waypoints
    tfl = tf/3 + tf/3*(zz-1);
    til = tfl - tf/3;
    %polynomial interpolation
    if (zz > 1)
        mudx = mudxf(zz-1);
        mudy = mudyf(zz-1);
        vardx = vardxf(zz-1);
        vardy = vardyf(zz-1);
    end
    Cx = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[vardx; 0; vardxf(zz); 0];
    Cy = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[vardy; 0; vardyf(zz); 0];

    Cmx = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[mudx; 0; mudxf(zz); 0];
    Cmy = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[mudy; 0; mudyf(zz); 0];
    traj_hist = [traj_hist; Cx' Cy' Cmx' Cmy' tfl];
end

figure(1);
clf(1);
hold;
options = odeset('RelTol', 1e-12, 'AbsTol', 1e-10*[ones( 2*num,1)]');
[t, q] = ode45('statsump', tspan, q0); % calculates the x, y position based on time

    nextT = 0;
    axis([-2 16 -3 14]);
    axis('manual');
    xlabel('X (m)');
    ylabel('Y (m)');
    title('Motion of Platoon Units with the Hybrid Controller ');
    %hold;
    theta = 0:0.01:2*pi;
    [a,b]=size ( Xobst);
    for (i=1:b)
        obst = [R*cos(theta)+Xobst(i); R*sin(theta)+Yobst(i)]; % showing the
obstacle
        plot(obst(1, :), obst(2, :));
    end
     [a,c]=size (Xmine);
    for (i=1:c)
        mine = [Radmine*cos(theta)+Xmine(i); Radmine*sin(theta)+Ymine(i)]; % showing
the obstacle
```

```
        plot(mine(1, :), mine(2, :));
    end
    robot = [-0.03 -0.03; 0.03 -0.03; 0.03 0.03; -0.03 0.03];% using little squares
to represent the  robots
    clear M;
    col = ['g', 'r', 'b', 'c', 'm', 'y'];
    for (i = 1:length(t))
        if t(i) >= nextT     % everytime, T, draw and color robot..
            for (j = 1:num)
                fill(robot(:, 1)+q(i, j), robot(:, 2)+q(i, j+num), col(j)); % fill
in shape and color of the simulated robots.
            end
            nextT = nextT + 0.10; % at next time T
        end
        %M(i) = getframe;
        %pause
    end
    axis('equal');
    axis([min(min(q(:, 1:6)))-1, max(max(q(:,1:6)))+1,min(min(q(:, 7:12)))-1,
max(max(q(:,7:12)))+1]);
    hold off;
%     %labelling the plot
%     for h=1:(length(Xobst))
%         gtext(['Obstacle #',num2str(h)]);
%     end
%     for p=1:(length(Xmine))
%         gtext(['Mine #',num2str(p)]);
%     end
%     for j=1:num
%         gtext(['Unit #',num2str(j)]);
%     end
figure(2)
clf(2);
for i = 1:6
    subplot(3,2,i);
    axis([-2 16 -2 16]);
    axis('manual');
    title(['Unit #', num2str(i)]);
    ylabel('Y (m)');
     if i>4
    xlabel('X (m)');
    end
    hold
    plot(q(:, 1+(i-1)), q(:, 1+num+(i-1))); % plot individual paths
    [a,b]=size ( Xobst); %plot individual obstacles
    theta = 0:0.01:2*pi;
    for (i=1:b)
        obst = [R*cos(theta)+Xobst(i); R*sin(theta)+Yobst(i)]; % showing the
obstacle
        plot(obst(1, :), obst(2, :));
    end
    [c,d]=size ( Xmine); % plot individual mines
    for (i=1:d)
        mine = [Radmine*cos(theta)+Xmine(i); Radmine*sin(theta)+Ymine(i)]; % showing
the obstacle
        plot(mine(1, :), mine(2, :));
    end
    hold off
end
xmean = mean(q(:, 1:num)')';
ymean = mean(q(:, num+1:2*num)')';
xvar = var(q(:, 1:num)')';
yvar = var(q(:, num+1:2*num)')';

figure(3)
clf(3);
subplot(2,2,1)
plot(t, xmean,'b:');
hold;
[xx, qq] = size(traj_hist);
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cmx = traj_hist(i, 9:12)';
    plot(T, Cmx(1)*T.^3 + Cmx(2)*T.^2 + Cmx(3)*T + Cmx(4), 'r-');
    title('Platoon Tracking Mean X Errors');
    xlabel('Time (s)');
    ylabel('Mean X (m)');
end
hold
subplot(2,2,2)
plot(t, ymean,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cmy = traj_hist(i, 13:16)';
    plot(T, Cmy(1)*T.^3 + Cmy(2)*T.^2 + Cmy(3)*T + Cmy(4), 'r-');
      title('Platoon Tracking Mean Y Errors');
    xlabel('Time (s)');
    ylabel('Mean Y (m)');
end
hold
subplot(2,2,3)
plot(t, xvar,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cx = traj_hist(i, 1:4)';
    plot(T, Cx(1)*T.^3 + Cx(2)*T.^2 + Cx(3)*T + Cx(4), 'r-');
    title('Platoon Tracking Variance X Errors');
    xlabel('Time (s)');
    ylabel('Variance X ');
end
hold
subplot(2,2,4)
plot(t, yvar,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cy = traj_hist(i, 5:8)';
    plot(T, Cy(1)*T.^3 + Cy(2)*T.^2 + Cy(3)*T + Cy(4), 'r-');
   title('Platoon Tracking Variance Y Errors');
    xlabel('Time (s)');
    ylabel('Variance Y ');
end
legend('actual values', 'desired values');
hold
collision = 0;
min_dist = 1000;
for i = 1:length(q)
    X = [];
    for j = 1:num
      X = [X, [q(i, j); q(i, j+num)]];
    end
    for k = 1:2
        for l = k+1:num
            dist = X(:, k) - X(:, l);
            dist = sqrt(dist'*dist);
            if (dist < min_dist)
                min_dist = dist;
            end
            if (dist < 0.06*sqrt(2))
                error('Collison!');
            end
        end
    end
end
```

```
%Appendix K
% Yong Tan, Trident scholar project
% function to find the position of each individual robots
% based on their current location.
function qdot = statsump(t, q)

global Km Kv Ka
global mudx vardx mudxf vardxf mux
global mudy vardy mudyf vardyf muy
global R RAD RRAD Radmine
global Xobst Yobst RADOBST Cx Cy Cmx Cmy num
global leg tf Xmine Ymine
global traj_hist
global avoid_hist avoid_obst XR YR
global A E F
global xydifferences

%C terms are used to control tragectories.
%Cx,Cy are variance tragectory controls
%Cmx,Cmy are mean trajectory controls
Ki=0.8;% gain of the mean

feedforward = 1;% drive robots forward in jacobian

%check distance between robots and the mines,
findmine=checkdist(q, Xmine, Ymine, Radmine); %find cloest dist and position
closestdistmine=findmine(1,:);    %find closest mine dist
posmine=findmine(2,:);            % position of mine

%check distance between robots and the obstacles
findobst=checkdist(q, Xobst, Yobst, R);
closestobst=findobst(1,:);
posobst=findobst(2,:);

mux = mean(q(1:num)); % x=q... represent the states x, y, x= 1:4, y 5:8
muy = mean(q(num+1:2*num)); % x(5:8 ) represent the y values
varx = var(q(1:num)); % variance of x
vary = var(q(num+1:2*num)); % variance of y
x_d = mudxf(leg);
y_d = mudyf(leg);

   magnitude=sqrt((x_d - mux)^2+ (y_d - muy)^2); % distance from centroid of swarm
to waypoint
   %waypoint changes if the time is
   for mm = 1:length(mudxf)
       if (t < traj_hist(mm, 17))
           break;
       end
   end
   leg = mm;
    %calculated trajectory projection based on each leg
    Cx = traj_hist(leg, 1:4)';
    Cy = traj_hist(leg, 5:8)';
    Cmx = traj_hist(leg, 9:12)';
    Cmy = traj_hist(leg, 13:16)';

    mudx = Cmx'*[t^3 t^2 t 1]'; %desired x mean position
    mudy = Cmy'*[t^3 t^2 t 1]'; %desired y mean position
    dmudx = Cmx'*[3*t^2 2*t 1 0]';  %desired vx mean velocity, derivatives
    dmudy = Cmy'*[3*t^2 2*t 1 0]';  %desired vy mean velocity, derivatives

    varxt = Cx'*[t^3 t^2 t 1]'; %desired x position variance
    varyt = Cy'*[t^3 t^2 t 1]'; %desired y position variance
    dvarxt = Cx'*[3*t^2 2*t 1 0]';  %desired vx velocity variance, derivatives
    dvaryt = Cy'*[3*t^2 2*t 1 0]';  %desired vy velocity variance, derivatives
  t
    J = [1/num*(ones(1,num)); 2/(num-1)*((q(1:num))'-mux)]; % matrix defined by
state and task function: difference from mean in x
    Jy = [J(1, :); 2/(num-1)*(q(num+1:2*num)' - muy)]; %alternate matrix defined by
state -diff from mean in y
    J = [J zeros(2, num); zeros(2, num) Jy]; %create matrix of no. by 8 to discribe
x and y characteristics for no. roobots
    Jp = J'*(J*J')^(-1);        % Moore-Penrose pseudo inverse T ( resultant- 3by 8
matrix)

    % attractive vectors to keep swarm moving forward
    xa = [Km*(mudx - mux)+ feedforward*dmudx; Kv*(varxt - varx) + dvarxt];  %the
changes in mean and variance in x with feedforward term
    xay = [Km*(mudy - muy)+ feedforward*dmudy; Kv*(varyt - vary) + dvaryt]; %the
changes in mean and variance in y with feedforward term
    xd = [xa; xay];%

    anglevec = zeros(1,6); % store angles of the vectors
    % find angle for each vector in grid
    for f=1:num
        q1=q(f);
        q2=q(f+num);
        gridx=round(q1);
        gridy=round(q2);

        anglemat=[];
        anglemat=set_angle;
        L_F= length(F);
        L_A= length ( A);
        %A=ones( length(F));

        for z=1:L_A
            for j=1:L_F
                if ((z==gridx) && (j==gridy))
                    anglevec(f)=anglemat(z,j);
                    break;
                end
            end
        end
        anglevec(f);

    end
    distbtwnrobots=checksepdistance(q); % check distance between each robot

%*************** generating subsumption in a single robot
***************************
% function to calculate repulsion from each robot
%irrep = zeros(2*num, 1);
num_active = 0;
for (i = 1:num)

   q1=q(i);
   q2=q(i+num);

%%  BEHAVIOR:  AVOID MINE
if ((closestdistmine(i)<=0.4)||((closestdistmine(i) <=0.8)&&(avoid_hist(i)==1)))
    avoid_hist(i) = 1;
    Xactm=Xmine(posmine(i));            % actual X position of mine
    Yactm=Ymine(posmine(i));              % actual Y position of mine
    figure(1);
    plot( Xactm, Yactm,'gd');

    repelmine=findreppot(q1,q2,Xactm,Yactm,Radmine);
    pot(i)=repelmine(1);
    pot(i+num)=repelmine(2);
    num_active = num_active + 1;
else
    %% BEHAVIOR:  AVOID OBSTACLE
    if ((closestobst(i)<=0.5)||((closestobst(i) <=1)&&(avoid_hist(i)==1)))
        avoid_hist(i) = 1;
        Xactobst=Xobst(posobst(i));
    Yactobst=Yobst(posobst(i));
    figure(1);
    plot( Xactobst, Yactobst,'r*');

    repobst=findreppot(q1,q2,Xactobst,Yactobst,R);
    pot(i)=repobst(1);
    pot(i+num)=repobst(2);
    num_active = num_active + 1;
else
    avoid_hist(i) = 0;
    %find largest distance between robots and the current mean
    sepdist=findmaxsep(q1,q2,mudx, mudy);

    %% BEHAVIOR:  AGGREGATION
    if ((sepdist(1))> 6)
        lengthc= sepdist(1);
        anglec= sepdist(2);

        pot(i)=lengthc * cos(anglec);
        pot(i+num)=lengthc * sin(anglec);
        num_active = num_active + 1;
    else
        %% BEHAVIOR:  SEPARATION
        checkclosest=[];
        checkclosest=distbtwnrobots(i,:);
        [Val, posi]=min(checkclosest);
        px=[];
        eachrobotrepx=0;
        eachrobotrepy=0;

        if Val<1;
            %check which robots are closer than 1
            for ic=1:num
                valuedist =checkclosest(1,ic);
                if valuedist<1
                    xxd=xydifferences(i,ic);
                    xyd=xydifferences(i,ic+num) ;

                    theta_ba=atan2(xyd, xxd);
                    px=(1/valuedist)*cos(theta_ba);
                    px2=(1/valuedist)*sin(theta_ba);

                else
                    px=0;
                    px2=0;
                end
                eachrobotrepx=eachrobotrepx+px;
                eachrobotrepy=eachrobotrepy+px2;
            end

        pot(i)=eachrobotrepx;
        pot(i+num)=eachrobotrepy;
        num_active = num_active + 1;

        %% BEHAVIOR:  RANDOM
        else

            if ((t > 0)&&(num_active < 3))
               pot(i) = 0.5*XR(ceil(t), i);
               pot(i+num) = 0.5*YR(ceil(t), i);
               num_active = num_active + 1;
           else
                pot(i) = 0;
                pot(i+num) = 0;
            end
        end
    end
end
```

```
end
end

qdot = Jp*xd + Ka*(eye(2*num) - Jp*J)*(pot');
```

```
% Appendix L
% Yong Tan, Trident Scholar Project
% This file simulates the hybrid controller in a drift environment
global mudx vardx mudxf vardxf % global variables of changes in mean and variance of
x
global mudy vardy mudyf vardyf % global variables of changes in mean and variance of
y
global Km Kv Ka % motion, velocity and acceleration constants
global R RAD RRAD Radmine% xoffset, yoffset, mean offset,
global Xobst Yobst Cx Cy Cmx Cmy num
global leg tf Xmine Ymine
global traj_hist
global A E F
global avoid_hist XR YR
global xydifferences

traj_hist = [];
num=6;            % number of robots
avoid_hist = zeros(num, 1);
R = 1;           % Radius of obstacle
mudxf = [2 12 12];
mudyf = [10 10 0];
vardxf = [5 3 5];
vardyf = [3 5 3];
Km = 3;             % 3     motion constant
Kv = 3;             % 3     velocity constant
Ka = 5;   % 1         % acceleration constant
RAD = 2;            % minimum separtion distance between robots
RRAD = 0.5;  % 1      % minimum distance between individual robots to obstacle
Radmine=0.3;
record = 0;
Xobst= [3 6 10 ];
Yobst = [4 8 4 ];
Xmine = [ 7 10];
Ymine = [ 10 6];
leg=1;

tf =18; % final time in sec
tspan = [0 tf]; % time span from 0-20 s

q0=[0 1 2 3 4 3.8105 0 2 0 2 1 2.5];

XR = [     0.9501    0.4103    0.8462    0.1509    0.8385  0.1365
    0.2311    0.8936    0.5252    0.6979    0.5681  0.0118
    0.6068    0.0579    0.2026    0.3784    0.3704  0.8939
    0.4860    0.3529    0.6721    0.8600    0.7027  0.1991
    0.8913    0.8132    0.8381    0.8537    0.5466  0.2987
    0.7621    0.0099    0.0196    0.5936    0.4449  0.6614
    0.4565    0.1389    0.6813    0.4966    0.6946  0.2844
    0.0185    0.2028    0.3795    0.8998    0.6213  0.4692
    0.8214    0.1987    0.8318    0.8216    0.7948  0.0648
    0.4447    0.6038    0.5028    0.6449    0.9568  0.9883
    0.6154    0.2722    0.7095    0.8180    0.5226  0.5828
    0.7919    0.1988    0.4289    0.6602    0.8801  0.4235
    0.9218    0.0153    0.3046    0.3420    0.1730  0.5155
    0.7382    0.7468    0.1897    0.2897    0.9797  0.3340
    0.1763    0.4451    0.1934    0.3412    0.2714  0.4329
    0.4057    0.9318    0.6822    0.5341    0.2523  0.2259
    0.9355    0.4660    0.3028    0.7271    0.8757  0.5798
    0.9169    0.4186    0.5417    0.3093    0.7373  0.7600];

YR =[    0.9669    0.4608    0.4199    0.6273    0.7036 0.7009
    0.6649    0.4574    0.7537    0.6991    0.4850  0.9623
    0.8704    0.4507    0.7939    0.3972    0.1146  0.7505
    0.0099    0.4122    0.9200    0.4136    0.6649  0.7400
    0.1370    0.9016    0.8447    0.6552    0.3654  0.4319
    0.8188    0.0056    0.3678    0.8376    0.1400  0.6343
    0.4302    0.2974    0.6208    0.3716    0.5668  0.8030
    0.8903    0.0492    0.7313    0.4253    0.8230  0.0839
    0.7349    0.6932    0.1939    0.5947    0.6739   0.9455
    0.6873    0.6501    0.9048    0.5657    0.9994  0.9159
    0.3461    0.9830    0.5692    0.7165    0.9616  0.6020
    0.1660    0.5527    0.6318    0.5113    0.0589  0.2536
    0.1556    0.4001    0.2344    0.7764    0.3603  0.8735
    0.1911    0.1988    0.5488    0.4893    0.5485  0.5134
    0.4225    0.6252    0.9316    0.1859    0.2618  0.7327
    0.8560    0.7334    0.3352    0.7006    0.5973  0.4222
    0.4902    0.3759    0.6555    0.9827    0.0493  0.9614
    0.8159    0.0099    0.3919    0.8066    0.5711  0.0721];

mudx = mean(q0(1:num)); % mean  x at starting time
mudy = mean(q0(num+1: 2*num));% mean y at starting time
vardx = var(q0(1:num)); %initial varx
vardy = var(q0(num+1: 2*num));

% calculate trajetory history for each leg!!!!! % to be used in function
for zz = 1 : 3 %number of waypoints
    tfl = tf/3 + tf/3*(zz-1);
    til = tfl - tf/3;
    %polynomial interpolation
    if (zz > 1)
        mudx = mudxf(zz-1);
        mudy = mudyf(zz-1);
        vardx = vardxf(zz-1);
        vardy = vardyf(zz-1);
    end
    Cx = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[vardx; 0; vardxf(zz); 0];
    Cy = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[vardy; 0; vardyf(zz); 0];

    Cmx = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[mudx; 0; mudxf(zz); 0];
    Cmy = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[mudy; 0; mudyf(zz); 0];
    traj_hist = [traj_hist; Cx' Cy' Cmx' Cmy' tfl];
end

figure(1);
clf(1);
hold;
options = odeset('RelTol', 1e-12, 'AbsTol', 1e-10*[ones( 2*num,1)]');
[t, q] = ode45('newstatsump', tspan, q0); % calculates the x, y position based on
time
%plot vector fields
test;
    nextT = 0;
    axis([-2 16 -3 14]);
    axis('manual');
    xlabel('X (m)');
    ylabel('Y (m)');
    title('Motion of Platoon Units with the Hybrid Controller Without Vector
Compensation');
    %hold;
    theta = 0:0.01:2*pi;
    [a,b]=size ( Xobst);
    for (i=1:b)
        obst = [R*cos(theta)+Xobst(i); R*sin(theta)+Yobst(i)]; % showing the
obstacle
        plot(obst(1, :), obst(2, :));
    end
     [a,c]=size (Xmine);
    for (i=1:c)
        mine = [Radmine*cos(theta)+Xmine(i); Radmine*sin(theta)+Ymine(i)]; % showing
the obstacle
        plot(mine(1, :), mine(2, :));
    end
    robot = [-0.03 -0.03; 0.03 -0.03; 0.03 0.03; -0.03 0.03];% using little squares
to represent the  robots
    clear M;
    col = ['g', 'r', 'b', 'c', 'm', 'y'];
    for (i = 1:length(t))
        if t(i) >= nextT     % everytime, T, draw and color robot..
            for (j = 1:num)
                fill(robot(:, 1)+q(i, j), robot(:, 2)+q(i, j+num), col(j)); % fill
in shape and color of the simulated robots.
            end
            nextT = nextT + 0.10; % at next time T
        end
        %M(i) = getframe;
        %pause
    end
    axis('equal');
    axis([min(min(q(:, 1:6)))-1, max(max(q(:,1:6)))+1,min(min(q(:, 7:12)))-1,
max(max(q(:,7:12)))+1]);
    hold off;
%     %labelling the plot
%     for h=1:(length(Xobst))
%         gtext(['Obstacle #',num2str(h)]);
%     end
%     for p=1:(length(Xmine))
%         gtext(['Mine #',num2str(p)]);
%     end
%     for j=1:num
%         gtext(['Unit #',num2str(j)]);
%     end
figure(2)
clf(2);
for i = 1:6
    subplot(3,2,i);
    axis([-2 16 -2 16]);
    axis('manual');
    title(['Unit #', num2str(i)]);
    ylabel('Y (m)');
     if i>4
    xlabel('X (m)');
    end
    hold
    plot(q(:, 1+(i-1)), q(:, 1+num+(i-1))); % plot individual paths
    [a,b]=size ( Xobst); %plot individual obstacles
    theta = 0:0.01:2*pi;
    for (i=1:b)
        obst = [R*cos(theta)+Xobst(i); R*sin(theta)+Yobst(i)]; % showing the
obstacle
        plot(obst(1, :), obst(2, :));
    end
    [c,d]=size ( Xmine); % plot individual mines
    for (i=1:d)
        mine = [Radmine*cos(theta)+Xmine(i); Radmine*sin(theta)+Ymine(i)]; % showing
the obstacle
        plot(mine(1, :), mine(2, :));
    end
    hold off
end
xmean = mean(q(:, 1:num)')';
ymean = mean(q(:, num+1:2*num)')';
xvar = var(q(:, 1:num)')';
yvar = var(q(:, num+1:2*num)')';

figure(3)
clf(3);
subplot(2,2,1)
plot(t, xmean,'b:');
hold;
[xx, qq] = size(traj_hist);
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cmx = traj_hist(i, 9:12)';
    plot(T, Cmx(1)*T.^3 + Cmx(2)*T.^2 + Cmx(3)*T + Cmx(4), 'r-');
    title('Platoon Tracking Mean X Errors');
    xlabel('Time (s)');
    ylabel('Mean X (m)');
end
hold
subplot(2,2,2)
plot(t, ymean,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cmy = traj_hist(i, 13:16)';
    plot(T, Cmy(1)*T.^3 + Cmy(2)*T.^2 + Cmy(3)*T + Cmy(4), 'r-');
      title('Platoon Tracking Mean Y Errors');
    xlabel('Time (s)');
    ylabel('Mean Y (m)');
end
hold
subplot(2,2,3)
plot(t, xvar,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cx = traj_hist(i, 1:4)';
    plot(T, Cx(1)*T.^3 + Cx(2)*T.^2 + Cx(3)*T + Cx(4), 'r-');
    title('Platoon Tracking Variance X Errors');
    xlabel('Time (s)');
    ylabel('Variance X ');
end
hold
subplot(2,2,4)
plot(t, yvar,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cy = traj_hist(i, 5:8)';
    plot(T, Cy(1)*T.^3 + Cy(2)*T.^2 + Cy(3)*T + Cy(4), 'r-');
   title('Platoon Tracking Variance Y Errors');
    xlabel('Time (s)');
    ylabel('Variance Y ');
end
legend('actual values', 'desired values');
hold

collision = 0;
min_dist = 1000;
for i = 1:length(q)
    X = [];
    for j = 1:num
      X = [X, [q(i, j); q(i, j+num)]];
    end
    for k = 1:2
        for l = k+1:num
            dist = X(:, k) - X(:, l);
            dist = sqrt(dist'*dist);
            if (dist < min_dist)
                min_dist = dist;
            end
            if (dist < 0.06*sqrt(2))
                error('Collison!');
            end
        end
    end
end
```

```
%Appendix M
% Yong Tan, Trident Scholar Project
% Function to run program that simulates movement of robots
% in a drift environment

function qdot = newstatsump(t, q)
% function show the statistical controllers work in an
%environment with mines and obstacles.

global Km Kv Ka
global mudx vardx mudxf vardxf mux
global mudy vardy mudyf vardyf muy
global R RAD RRAD Radmine
global Xobst Yobst RADOBST Cx Cy Cmx Cmy num
global leg tf Xmine Ymine
global traj_hist
global avoid_hist avoid_obst XR YR
global A E F
global xydifferences

%C terms are used to control tragectories.
%Cx,Cy are variance tragectory controls
%Cmx,Cmy are mean trajectory controls
Ki=0.8;% gain of the mean
% Kivx=0.25; % gain of the variance
% Kivy=0.8;

feedforward = 1;% drive robots forward in jacobian

%check distance between robots and the mines,
findmine=checkdist(q, Xmine, Ymine, Radmine); %find cloest dist and position
closestdistmine=findmine(1,:);    %find closest mine dist
posmine=findmine(2,:);            % position of mine

%check distance between robots and the obstacles
findobst=checkdist(q, Xobst, Yobst, R);
closestobst=findobst(1,:);
posobst=findobst(2,:);

mux = mean(q(1:num)); % x=q... represent the states x, y, x= 1:4, y 5:8
muy = mean(q(num+1:2*num)); % x(5:8 ) represent the y values
varx = var(q(1:num)); % variance of x
vary = var(q(num+1:2*num)); % variance of y
x_d = mudxf(leg);
y_d = mudyf(leg);

   magnitude=sqrt((x_d - mux)^2+ (y_d - muy)^2); % distance from centroid of swarm
to waypoint

   %waypoint changes if the time is
   for mm = 1:length(mudxf)
       if (t < traj_hist(mm, 17))
           break;
       end
   end
   leg = mm;
    %calculated trajectory projection based on each leg
    Cx = traj_hist(leg, 1:4)';
    Cy = traj_hist(leg, 5:8)';
    Cmx = traj_hist(leg, 9:12)';
    Cmy = traj_hist(leg, 13:16)';

    mudx = Cmx'*[t^3 t^2 t 1]'; %desired x mean position
    mudy = Cmy'*[t^3 t^2 t 1]'; %desired y mean position
    dmudx = Cmx'*[3*t^2 2*t 1 0]';  %desired vx mean velocity, derivatives
    dmudy = Cmy'*[3*t^2 2*t 1 0]';  %desired vy mean velocity, derivatives

    varxt = Cx'*[t^3 t^2 t 1]'; %desired x position variance
    varyt = Cy'*[t^3 t^2 t 1]'; %desired y position variance
    dvarxt = Cx'*[3*t^2 2*t 1 0]';  %desired vx velocity variance, derivatives
    dvaryt = Cy'*[3*t^2 2*t 1 0]';  %desired vy velocity variance, derivatives

  t
    J = [1/num*(ones(1,num)); 2/(num-1)*((q(1:num))'-mux)]; % matrix defined by
state and task function: difference from mean in x
    Jy = [J(1, :); 2/(num-1)*(q(num+1:2*num)' - muy)]; %alternate matrix defined by
state -diff from mean in y
    J = [J zeros(2, num); zeros(2, num) Jy]; %create matrix of no. by 8 to discribe
x and y characteristics for no. roobots
    Jp = J'*(J*J')^(-1);        % Moore-Penrose pseudo inverse T ( resultant- 3by 8
matrix)

    % attractive vectors to keep swarm moving forward
    xa = [Km*(mudx - mux)+ feedforward*dmudx; Kv*(varxt - varx) + dvarxt];  %the
changes in mean and variance in x with feedforward term
    xay = [Km*(mudy - muy)+ feedforward*dmudy; Kv*(varyt - vary) + dvaryt]; %the
changes in mean and variance in y with feedforward term
    xd = [xa; xay];%

    anglevec = zeros(1,6); % store angles of the vectors
    % find angle for each vector in grid
    for f=1:num
        q1=q(f);
        q2=q(f+num);
        gridx=round(q1);
        gridy=round(q2);

        anglemat=[];
        anglemat=set_angle;
        L_F= length(F);
        L_A= length ( A);
        %A=ones( length(F));

        for z=1:L_A
            for j=1:L_F
                if ((z==gridx) && (j==gridy))
                    anglevec(f)=anglemat(z,j);
                    break;
                end
            end
        end
        anglevec(f);

    end
    distbtwnrobots=checksepdistance(q); % check distance between each robot

%*************** generating subsumption in a single robot
***************************
% function to calculate repulsion from each robot
%irrep = zeros(2*num, 1);
num_active = 0;
for (i = 1:num)
   q1=q(i);
   q2=q(i+num);

%%  BEHAVIOR:  AVOID MINE
if ((closestdistmine(i)<=0.4)||((closestdistmine(i) <=0.8)&&(avoid_hist(i)==1)))
    avoid_hist(i) = 1;
    Xactm=Xmine(posmine(i));            % actual X position of mine
    Yactm=Ymine(posmine(i));              % actual Y position of mine
    figure(1);
    plot( Xactm, Yactm,'gd');

    repelmine=findreppot(q1,q2,Xactm,Yactm,Radmine);
    pot(i)=repelmine(1);
    pot(i+num)=repelmine(2);
    num_active = num_active + 1;
else
    %% BEHAVIOR:  AVOID OBSTACLE
    if ((closestobst(i)<=0.5)||((closestobst(i) <=1)&&(avoid_hist(i)==1)))
        avoid_hist(i) = 1;
        Xactobst=Xobst(posobst(i));
        Yactobst=Yobst(posobst(i));
        figure(1);
        plot( Xactobst, Yactobst,'r*');

        repobst=findreppot(q1,q2,Xactobst,Yactobst,R);
        pot(i)=repobst(1);
        pot(i+num)=repobst(2);
        num_active = num_active + 1;
    else
        avoid_hist(i) = 0;
        %find largest distance between robots and the current mean
        sepdist=findmaxsep(q1,q2,mudx, mudy);

        %% BEHAVIOR:  AGGREGATION
        if ((sepdist(1))> 6)
            lengthc= sepdist(1);
            anglec= sepdist(2);

            pot(i)=lengthc * cos(anglec);
            pot(i+num)=lengthc * sin(anglec);
            num_active = num_active + 1;
        else
            %% BEHAVIOR:  SEPARATION
            checkclosest=[];
            checkclosest=distbtwnrobots(i,:);
            [Val, posi]=min(checkclosest);
            px=[];
            eachrobotrepx=0;
            eachrobotrepy=0;

            if Val<1;
                %check whcih robots are closer than 1
                for ic=1:num
                    valuedist =checkclosest(1,ic);
                    if valuedist<1
                        xxd=xydifferences(i,ic);
                        xyd=xydifferences(i,ic+num) ;

                        theta_ba=atan2(xyd, xxd);
                        px=(1/valuedist)*cos(theta_ba);
                        px2=(1/valuedist)*sin(theta_ba);
                    else
                        px=0;
                        px2=0;
                    end
                    eachrobotrepx=eachrobotrepx+px;
                    eachrobotrepy=eachrobotrepy+px2;
                end

            pot(i)=eachrobotrepx;
            pot(i+num)=eachrobotrepy;
            num_active = num_active + 1;
            %% BEHAVIOR:  RANDOM
            else
                %
                if ((t > 0)&&(num_active < 3))
                   pot(i) = 0.5*XR(ceil(t), i);
                   pot(i+num) = 0.5*YR(ceil(t), i);
                   num_active = num_active + 1;
               else
                    pot(i) = 0;
                    pot(i+num) = 0;
                            end
                        end
                    end
                end
            end
        end
qdot = Jp*xd + Ka*(eye(2*num) - Jp*J)*(pot')+5*[cos(anglevec'); sin(anglevec')] ;
```

```
% Appendix N
% Yong Tan, Tridnet Scholar project
% This file compensates for the drift vector of hybrid controller
global mudx vardx mudxf vardxf % global variables of changes in mean and variance of
x
global mudy vardy mudyf vardyf % global variables of changes in mean and variance of
y
global Km Kv Ka % motion, velocity and acceleration constants
global R RAD RRAD Radmine% xoffset, yoffset, mean offset,
global Xobst Yobst Cx Cy Cmx Cmy num
global leg tf Xmine Ymine
global traj_hist
global A E F
global avoid_hist XR YR
global xydifferences

traj_hist = [];
num=6;            % number of robots
avoid_hist = zeros(num, 1);
R = 1;           % Radius of obstacle

mudxf = [2 12 12];
mudyf = [10 10 0];
vardxf = [5 3 5];
vardyf = [3 5 3];
Km = 3;             % 3     motion constant
Kv = 5;             % 3     velocity constant
Ka = 5;   % 1         % acceleration constant
RAD = 2;            % minimum separtion distance between robots
RRAD = 0.5;  % 1      % minimum distance between individual robots to obstacle
Radmine=0.3;
record = 0;
Xobst= [3 6 10 ];
Yobst = [4 8 4 ];
Xmine = [ 7 10];
Ymine = [ 10 6];
leg=1;

tf =18; % final time in sec
tspan = [0 tf]; % time span from 0-20 s

q0=[0 1 2 3 4 3.8105 0 2 0 2 1 2.5 0 0 0 0];

XR = [     0.9501    0.4103    0.8462    0.1509    0.8385  0.1365
    0.2311    0.8936    0.5252    0.6979    0.5681  0.0118
    0.6068    0.0579    0.2026    0.3784    0.3704  0.8939
    0.4860    0.3529    0.6721    0.8600    0.7027  0.1991
    0.8913    0.8132    0.8381    0.8537    0.5466  0.2987
    0.7621    0.0099    0.0196    0.5936    0.4449  0.6614
    0.4565    0.1389    0.6813    0.4966    0.6946  0.2844
    0.0185    0.2028    0.3795    0.8998    0.6213  0.4692
    0.8214    0.1987    0.8318    0.8216    0.7948  0.0648
    0.4447    0.6038    0.5028    0.6449    0.9568  0.9883
    0.6154    0.2722    0.7095    0.8180    0.5226  0.5828
    0.7919    0.1988    0.4289    0.6602    0.8801  0.4235
    0.9218    0.0153    0.3046    0.3420    0.1730  0.5155
    0.7382    0.7468    0.1897    0.2897    0.9797  0.3340
    0.1763    0.4451    0.1934    0.3412    0.2714  0.4329
    0.4057    0.9318    0.6822    0.5341    0.2523  0.2259
    0.9355    0.4660    0.3028    0.7271    0.8757  0.5798
    0.9169    0.4186    0.5417    0.3093    0.7373  0.7600];

YR =[    0.9669    0.4608    0.4199    0.6273    0.7036 0.7009
    0.6649    0.4574    0.7537    0.6991    0.4850  0.9623
    0.8704    0.4507    0.7939    0.3972    0.1146  0.7505
    0.0099    0.4122    0.9200    0.4136    0.6649  0.7400
    0.1370    0.9016    0.8447    0.6552    0.3654  0.4319
    0.8188    0.0056    0.3678    0.8376    0.1400  0.6343
    0.4302    0.2974    0.6208    0.3716    0.5668  0.8030
    0.8903    0.0492    0.7313    0.4253    0.8230  0.0839
    0.7349    0.6932    0.1939    0.5947    0.6739   0.9455
    0.6873    0.6501    0.9048    0.5657    0.9994  0.9159
    0.3461    0.9830    0.5692    0.7165    0.9616  0.6020
    0.1660    0.5527    0.6318    0.5113    0.0589  0.2536
    0.1556    0.4001    0.2344    0.7764    0.3603  0.8735
    0.1911    0.1988    0.5488    0.4893    0.5485  0.5134
    0.4225    0.6252    0.9316    0.1859    0.2618  0.7327
    0.8560    0.7334    0.3352    0.7006    0.5973  0.4222
    0.4902    0.3759    0.6555    0.9827    0.0493  0.9614
    0.8159    0.0099    0.3919    0.8066    0.5711  0.0721];

mudx = mean(q0(1:num)); % mean  x at starting time
mudy = mean(q0(num+1: 2*num));% mean y at starting time
vardx = var(q0(1:num)); %initial varx
vardy = var(q0(num+1: 2*num));

% calculate trajetory history for each leg!!!!! % to be used in function
for zz = 1 : 3 %number of waypoints
    tfl = tf/3 + tf/3*(zz-1);
    til = tfl - tf/3;
    %polynomial interpolation
    if (zz > 1)
        mudx = mudxf(zz-1);
        mudy = mudyf(zz-1);
        vardx = vardxf(zz-1);
        vardy = vardyf(zz-1);
    end
    Cx = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[vardx; 0; vardxf(zz); 0];
    Cy = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[vardy; 0; vardyf(zz); 0];

    Cmx = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[mudx; 0; mudxf(zz); 0];
    Cmy = [til^3 til^2 til 1; 3*til^2 2*til 1 0; tfl^3 tfl^2 tfl 1; 3*tfl^2 2*tfl 1
0]^(-1)*[mudy; 0; mudyf(zz); 0];
    traj_hist = [traj_hist; Cx' Cy' Cmx' Cmy' tfl];
end

figure(1);
clf(1);
hold;
options = odeset('RelTol', 1e-12, 'AbsTol', 1e-10*[ones( 2*num,1)]');
[t, q] = ode45('veccompensate', tspan, q0); % calculates the x, y position based on
time
test; %plot vector fields
    nextT = 0;
    axis([-2 16 -3 14]);
    axis('manual');
    xlabel('X (m)');
    ylabel('Y (m)');
    title('Motion of Platoon Units with the Hybrid Controller and Vector
Compensation');
    %hold;
    theta = 0:0.01:2*pi;
    [a,b]=size ( Xobst);
    for (i=1:b)
        obst = [R*cos(theta)+Xobst(i); R*sin(theta)+Yobst(i)]; % showing the
obstacle
        plot(obst(1, :), obst(2, :));
    end
     [a,c]=size (Xmine);
    for (i=1:c)
        mine = [Radmine*cos(theta)+Xmine(i); Radmine*sin(theta)+Ymine(i)]; % showing
the obstacle
        plot(mine(1, :), mine(2, :));
    end
    robot = [-0.03 -0.03; 0.03 -0.03; 0.03 0.03; -0.03 0.03];% using little squares
to represent the  robots
    clear M;
    col = ['g', 'r', 'b', 'c', 'm', 'y'];
    for (i = 1:length(t))
        if t(i) >= nextT     % everytime, T, draw and color robot..
            for (j = 1:num)
                fill(robot(:, 1)+q(i, j), robot(:, 2)+q(i, j+num), col(j)); % fill
in shape and color of the simulated robots.
            end
            nextT = nextT + 0.10; % at next time T
        end
        %M(i) = getframe;
        %pause
    end
    axis('equal');
    axis([min(min(q(:, 1:6)))-1, max(max(q(:,1:6)))+1,min(min(q(:, 7:12)))-1,
max(max(q(:,7:12)))+1]);
    hold off;
%     %labelling the plot
%     for h=1:(length(Xobst))
%         gtext(['Obstacle #',num2str(h)]);
%     end
%     for p=1:(length(Xmine))
%         gtext(['Mine #',num2str(p)]);
%     end
%     for j=1:num
%         gtext(['Unit #',num2str(j)]);
%     end
figure(2)
clf(2);
for i = 1:6
    subplot(3,2,i);
    axis([-2 16 -2 16]);
    axis('manual');
    title(['Unit #', num2str(i)]);
    ylabel('Y (m)');
     if i>4
    xlabel('X (m)');
    end
    hold
    plot(q(:, 1+(i-1)), q(:, 1+num+(i-1))); % plot individual paths
    [a,b]=size ( Xobst); %plot individual obstacles
    theta = 0:0.01:2*pi;
    for (i=1:b)
        obst = [R*cos(theta)+Xobst(i); R*sin(theta)+Yobst(i)]; % showing the
obstacle
        plot(obst(1, :), obst(2, :));
    end
    [c,d]=size ( Xmine); % plot individual mines
    for (i=1:d)
        mine = [Radmine*cos(theta)+Xmine(i); Radmine*sin(theta)+Ymine(i)]; % showing
the obstacle
        plot(mine(1, :), mine(2, :));
    end
    hold off
end
xmean = mean(q(:, 1:num)')';
ymean = mean(q(:, num+1:2*num)')';
xvar = var(q(:, 1:num)')';
yvar = var(q(:, num+1:2*num)')';
figure(3)
clf(3);
subplot(2,2,1)
plot(t, xmean,'b:');
hold;
[xx, qq] = size(traj_hist);
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cmx = traj_hist(i, 9:12)';
    plot(T, Cmx(1)*T.^3 + Cmx(2)*T.^2 + Cmx(3)*T + Cmx(4), 'r-');
    title('Platoon Tracking Mean X Errors');
    xlabel('Time (s)');
    ylabel('Mean X (m)');
end
hold
subplot(2,2,2)
plot(t, ymean,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cmy = traj_hist(i, 13:16)';
    plot(T, Cmy(1)*T.^3 + Cmy(2)*T.^2 + Cmy(3)*T + Cmy(4), 'r-');
      title('Platoon Tracking Mean Y Errors');
    xlabel('Time (s)');
    ylabel('Mean Y (m)');
end
hold
subplot(2,2,3)
plot(t, xvar,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cx = traj_hist(i, 1:4)';
    plot(T, Cx(1)*T.^3 + Cx(2)*T.^2 + Cx(3)*T + Cx(4), 'r-');
    title('Platoon Tracking Variance X Errors');
    xlabel('Time (s)');
    ylabel('Variance X ');
end
hold
subplot(2,2,4)
plot(t, yvar,'b:');
hold;
for i = 1:xx
    T = [traj_hist(i, 17)-tf/3: 0.01: traj_hist(i,17)]';
    Cy = traj_hist(i, 5:8)';
    plot(T, Cy(1)*T.^3 + Cy(2)*T.^2 + Cy(3)*T + Cy(4), 'r-');
   title('Platoon Tracking Variance Y Errors');
    xlabel('Time (s)');
    ylabel('Variance Y ');
end
legend('actual values', 'desired values');
hold

collision = 0;
min_dist = 1000;
for i = 1:length(q)
    X = [];
    for j = 1:num
      X = [X, [q(i, j); q(i, j+num)]];
    end
    for k = 1:2
        for l = k+1:num
            dist = X(:, k) - X(:, l);
            dist = sqrt(dist'*dist);
            if (dist < min_dist)
                min_dist = dist;
            end
            if (dist < 0.06*sqrt(2))
                error('Collison!');

            end
        end
    end
end
```

```
% Appendix O
% Yong Tan, Trident Scholar Project
% FUnction that runs to show how robots compensate for drift.

function qdot = veccompensate(t, q)
% function show the statistical controllers work in an
%environment with mines and obstacles.

global Km Kv Ka
global mudx vardx mudxf vardxf mux
global mudy vardy mudyf vardyf muy
global R RAD RRAD Radmine
global Xobst Yobst RADOBST Cx Cy Cmx Cmy num
global leg tf Xmine Ymine
global traj_hist
global avoid_hist avoid_obst XR YR
global A E F
global xydifferences
%C terms are used to control tragectories.
%Cx,Cy are variance tragectory controls
%Cmx,Cmy are mean trajectory controls
Ki=1;% gain of the mean
Kivx=1; % gain of the variance
Kivy=1;

feedforward = 1;% drive robots forward in jacobian

%check distance between robots and the mines,
findmine=checkdist(q, Xmine, Ymine, Radmine); %find cloest dist and position
closestdistmine=findmine(1,:);    %find closest mine dist
posmine=findmine(2,:);            % position of mine

%check distance between robots and the obstacles
findobst=checkdist(q, Xobst, Yobst, R);
closestobst=findobst(1,:);
posobst=findobst(2,:);

mux = mean(q(1:num)); % x=q... represent the states x, y, x= 1:4, y 5:8
muy = mean(q(num+1:2*num)); % x(5:8 ) represent the y values
varx = var(q(1:num)); % variance of x
vary = var(q(num+1:2*num)); % variance of y
x_d = mudxf(leg);
y_d = mudyf(leg);

   magnitude=sqrt((x_d - mux)^2+ (y_d - muy)^2); % distance from centroid of swarm
to waypoint
   %waypoint changes if the time is
   for mm = 1:length(mudxf)
       if (t < traj_hist(mm, 17))
           break;
       end
   end
   leg = mm;
    %calculated trajectory projection based on each leg
    Cx = traj_hist(leg, 1:4)';
    Cy = traj_hist(leg, 5:8)';
    Cmx = traj_hist(leg, 9:12)';
    Cmy = traj_hist(leg, 13:16)';

    mudx = Cmx'*[t^3 t^2 t 1]'; %desired x mean position
    mudy = Cmy'*[t^3 t^2 t 1]'; %desired y mean position
    dmudx = Cmx'*[3*t^2 2*t 1 0]';  %desired vx mean velocity, derivatives
    dmudy = Cmy'*[3*t^2 2*t 1 0]';  %desired vy mean velocity, derivatives

    varxt = Cx'*[t^3 t^2 t 1]'; %desired x position variance
    varyt = Cy'*[t^3 t^2 t 1]'; %desired y position variance
    dvarxt = Cx'*[3*t^2 2*t 1 0]';  %desired vx velocity variance, derivatives
    dvaryt = Cy'*[3*t^2 2*t 1 0]';  %desired vy velocity variance, derivatives
```

```
  t
    J = [1/num*(ones(1,num)); 2/(num-1)*((q(1:num))'-mux)]; % matrix defined by
state and task function: difference from mean in x
    Jy = [J(1, :); 2/(num-1)*(q(num+1:2*num)' - muy)]; %alternate matrix defined by
state -diff from mean in y
    J = [J zeros(2, num); zeros(2, num) Jy]; %create matrix of no. by 8 to discribe
x and y characteristics for no. roobots
    Jp = J'*(J*J')^(-1);        % Moore-Penrose pseudo inverse T ( resultant- 3by 8
matrix)

    % attractive vectors to keep swarm moving forward
    xa = [Km*(mudx - mux)+ feedforward*dmudx; Kv*(varxt - varx) + dvarxt];  %the
changes in mean and variance in x with feedforward term
    xay = [Km*(mudy - muy)+ feedforward*dmudy; Kv*(varyt - vary) + dvaryt]; %the
changes in mean and variance in y with feedforward term
    xd = [xa; xay];%

    anglevec = zeros(1,6); % store angles of the vectors
    % find angle for each vector in grid
    for f=1:num
        q1=q(f);
        q2=q(f+num);
        gridx=round(q1);
        gridy=round(q2);

        anglemat=[];
        anglemat=set_angle;
        L_F= length(F);
        L_A= length ( A);
        %A=ones( length(F));

        for z=1:L_A
            for j=1:L_F
                if ((z==gridx) && (j==gridy))
                    anglevec(f)=anglemat(z,j);
                    break;
                end
            end
        end
        anglevec(f);
    end
    distbtwnrobots=checksepdistance(q); % check distance between each robot

%*************** generating subsumption in a single robot
% function to calculate repulsion from each robot
%irrep = zeros(2*num, 1);
num_active = 0;
for (i = 1:num)

   q1=q(i);
   q2=q(i+num);

%%  BEHAVIOR:  AVOID MINE
if ((closestdistmine(i)<=0.4)||((closestdistmine(i) <=0.8)&&(avoid_hist(i)==1)))
    avoid_hist(i) = 1;
    Xactm=Xmine(posmine(i));            % actual X position of mine
    Yactm=Ymine(posmine(i));              % actual Y position of mine
    figure(1);
    plot( Xactm, Yactm,'gd');

    repelmine=findreppot(q1,q2,Xactm,Yactm,Radmine);
    pot(i)=repelmine(1);
    pot(i+num)=repelmine(2);
    num_active = num_active + 1;
else
    %% BEHAVIOR:  AVOID OBSTACLE
    if ((closestobst(i)<=0.5)||((closestobst(i) <=1)&&(avoid_hist(i)==1)))
    avoid_hist(i) = 1;
    Xactobst=Xobst(posobst(i));
    Yactobst=Yobst(posobst(i));
    figure(1);
    plot( Xactobst, Yactobst,'r*');

    repobst=findreppot(q1,q2,Xactobst,Yactobst,R);
    pot(i)=repobst(1);
    pot(i+num)=repobst(2);
    num_active = num_active + 1;
else
    avoid_hist(i) = 0;
    %find largest distance between robots and the current mean
    sepdist=findmaxsep(q1,q2,mudx, mudy);

    %% BEHAVIOR:  AGGREGATION
    if ((sepdist(1))> 6)
        lengthc= sepdist(1);
        anglec= sepdist(2);

        pot(i)=lengthc * cos(anglec);
        pot(i+num)=lengthc * sin(anglec);
        num_active = num_active + 1;
    else
        %% BEHAVIOR:  SEPARATION
        checkclosest=[];
        checkclosest=distbtwnrobots(i,:);
        [Val, posi]=min(checkclosest);
        px=[];
        eachrobotrepx=0;
        eachrobotrepy=0;

        if Val<1;
            %check whcih robots are closer than 1
            for ic=1:num
                valuedist =checkclosest(1,ic);
                if valuedist<1
                    xxd=xydifferences(i,ic);
                    xyd=xydifferences(i,ic+num) ;

                    theta_ba=atan2(xyd, xxd);
                    px=(1/valuedist)*cos(theta_ba);
                    px2=(1/valuedist)*sin(theta_ba);
                else
                    px=0;
                    px2=0;
                end
                eachrobotrepx=eachrobotrepx+px;
                eachrobotrepy=eachrobotrepy+px2;
            end

        pot(i)=eachrobotrepx;
        pot(i+num)=eachrobotrepy;
        num_active = num_active + 1;

        %% BEHAVIOR:  RANDOM
        else
          if ((t > 0)&&(num_active < 3))
               pot(i) = 0.5*XR(ceil(t), i);
               pot(i+num) = 0.5*YR(ceil(t), i);
               num_active = num_active + 1;
           else
                pot(i) = 0;
                pot(i+num) = 0;
            end
        end
    end
end
```

```
end

end


qdot = Jp*(xd+ [Ki*q(num*2+1); Kivx*q(num*2+2);Ki *q(num*2+3);Kivy*q(num*2+4)]) +
Ka*(eye(2*num) - Jp*J)*(pot')+4*[cos(anglevec'); sin(anglevec')] ;;
err = [mudx - mux; (varxt)-(varx); mudy - muy; (varyt)-(vary)];

qdot = [qdot; err];
```