Formation Robots

Senior Project Design Report

Lemuel Diaz
Michelle Enyeart
Kristen Kristich
Alan Moore
January 30, 2003

Table of Contents

Introduction	Page 3
Use Cases	Pages 4-5
Requirements	Pages 6-7
Technologies Used	Page 7
Trade Off Analysis	Pages 8-21
Algorithms	Pages 22-25
Behavior Description	Pages 26-30
Test Plan	Pages 31-32
Time Line	Pages 33-34
Appendix A – Functionality and State Diagrams	Pages 35-38
Overall System Diagram	Page 39
Appendix B – Angle of Arrival Diagrams	Pages 40-41
Appendix C – User Manual	Pages 42-51

Introduction

The overall vision of this project is to have an indefinite number of robots move in a predetermined formation from one location to another. This formation will be controlled by a combination of an off-board processor and individual on-board processors with a graphical user interface. The idea of having a set of robots moving in a desired formation with one controller has become more practical with the recently released government technology of Ultra Wideband communication (UWB). Our first exposure to UWB technology came from a Santa Clara alumnus who is currently working at Lockheed Martin and researching UWB. Our project could potentially be used to further his research as well as contribute to the following technologies:

Satellites- UWB could be used in the current image capturing satellites. This technology is not very susceptible to interference, which frequently causes problems in other wireless technologies. UWB would assist satellites in holding a predetermined formation with a great deal a precision. This degree of precision would allow scientists to more accurately measure distant objects in space.

Military- UWB could be used to maneuver vehicles in a set formation without a driver/controller for each vehicle. This could be implemented in all-terrain vehicles (tanks), aircraft, or even in convoys, to eliminate the loss of human life. This is also a valuable tool in surveying unknown or inaccessible terrain.

Industry- When a heavy or awkward load needs to be moved it often takes more than one machine or person to do this. Having the ability to stay in formation will allow the weight of heavy loads to be distributed among more than one machine. This could be especially useful in areas such as steel mills, junkyards, airports, and scrap metal factories.

Use Cases

An immediate application for this project could be to determine the layout of obstacles, such as furniture in a room. Between user prompted commands and artificial intelligence, which allows a memory of locations where obstacles are located to be created, the robots can navigate their way around a room with fairly good accuracy. This can be a fun ~~and~~ experiment which demonstrates the capabilities of the robots.

One interesting application of formation robots occurs in the field of satellite image capturing. Satellite imaging and related fields could use the implications of our project to potentially improve image quality. The UWB technology is not susceptible to the types of common signal interference that frequently cause problems in other wireless technologies. UWB could assist satellites or cameras in forming and holding a formation with a great deal a precision. This degree of precision would allow scientists to more accurately measure distant objects in space, among other things.

Another application involves manipulation of unmanned vehicles. UWB maintained formations could be used to maneuver vehicles in a set formation without a driver or controller for each vehicle. This could be implemented in all-terrain vehicles, aircraft, or even in convoys, in situations that may risk the loss of human life or that are unapproachable for large teams of people. Therefore, this would also be a valuable tool in surveying unknown or inaccessible terrain such as hazardous structures, other planets, or very high altitudes. There are many other uses for formation control applications, one final example comes from the futuristic idea of being able to drive on a highway without paying attention to road in front of you. With a so-called Autopilot, the UWB can control the distance of you and the cars surrounding. These intelligent highways could be the first place where the common man would use government technology UWB.

System Requirements

Formation Requirements

- Must maintain a fixed distance from robot to robot at all times
- Must be capable of maintaining formation determined by user or default setting after the initial formation is made
- A time must be designated as the time out amount, which would result in the formation to stop trying to get around impossible obstacles, or in essence to quit their current task
- At the start up once the user has defined the number of robots, desired formation, and distance between robots, all robots must get into initial formation
- Must maintain the orientation (maintain a front of formation)
- Allow for continuing operation of the modified formation after a single unit failure has occurred
- Be able to move from point A to point B, and if an obstacle is in the way must try to get around it or must alert the user to the futile attempt
- The UWB will be used to check and maintain distances between robots

Software and User side requirements

- Control 1-n robots from a single processing unit
- Need a user friendly interface (i.e., easy to use and understandable)
- Allow for the user to click on the grid to specify the new destination for the robot formation. In this case the user will not need to specify the angle, since they most likely do not know what it is, and the program will calculate it for them.
- Allow the user to enter in a specific distance to be traveled, and at what orientation (angle) to head in. The user will specify a desired angle. To measure the angle, the zero degree point is straight ahead in the direction the formation is facing, and all angles are measured counterclockwise.
- Depending on what obstacles the formation encounters, various error messages will pop up. Options for moving the formation back to its original starting location will be given.
- Display on the grid to the user where the front end of the robots (formation) are
- Either show the sonar to the user, or display a mark on the screen where an obstacle is located
- Allow for a user definable formation
- Allow for clockwise or counterclockwise rotation of the formation when necessary
- Needs to calculate the angle and distance the robot must travel when the user clicks a desired destination on the grid

Technologies Used

- UWB – Property of Lockheed Martin
- ActivMedia Pioneer Robots
- Wireless Network cards
- Multiple Laptops TBD
- Custom Built Battery Packs
- Microsoft Visual C++
- Compass x 3 TBD

Tradeoff Analysis

Powering Units

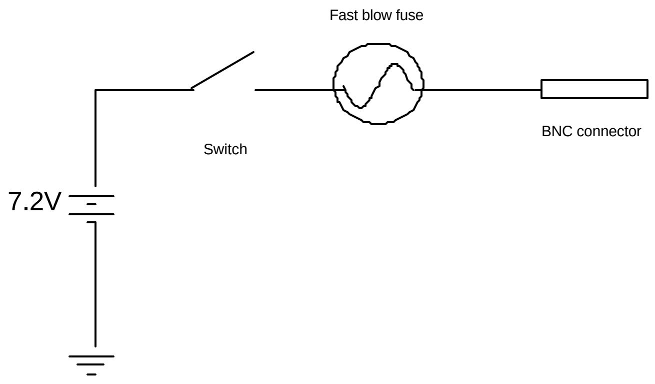
We need to power the UWB unit and the robot in a cost adequate and effective manner.

1) Chord to a power jack- This option would provide us with unlimited power. Having unlimited power has many advantages but the tether will prove to be the ultimate downfall of this option. Having a power chord connected from the robot/UWB unit to power outlet means that we are going to have chords laying on the ground which may hinder the movements of the robots and/or give unnecessary limitations to the distance capabilities.

2) Solar power-Given the scope of this project solar power panels would be too expensive. However this method of powering the units could prove very useful in other applications such as in space or lengthy operations.

3) Tap battery from robot to power UWB unit- While this method eliminates the need for any external connections, it also drains the battery on the robots much faster. If we were to tap the supply on the robot, we would have to create a power management circuit that isolates the UWB unit and provides the necessary voltage for it.

4) Creating power packs for the UWB- Given that the robots already have a power supply system within them, we propose that the best solution would be to create a separate power pack for the UWB units. This reduces the drain on the robot's battery. It also allows the units to move around independent of external supplies. . We have decided to go with a 7.2V battery. These batteries cost approximately $25. They have been tested under full load (applied by UWB unit) to have a lifetime of 1 hour and 15 minutes. Below is a schematic of the battery pack circuit which we have constructed.



Communication

Fast and effective communication between the robot and the user is very important. The robot needs to receive the movement commands and send information about obstacles and position. The only wireless communications devices that we currently have at our disposal are UWB and a wireless area network (WAN)

1) Sending/Receiving information through the UWB unit.- One of the possibilities was to use the payload of the UWB signal to send and receive the necessary commands and information. This proved to be ineffective due to the lack of our knowledge of the UWB operating system. In order to be able to send/receive information we would need to create an interface that would allow the robot's CPU to communicate with the UWB unit's CPU. Since we already have an alternate form of communication in place we decided to stick with that.

2) Sending/Receiving information through a WAN- Using the current set up of a wide area network we would be able to send and receive all the necessary information. This WAN constitutes one transmitter/receive on each of the robots and at the base station. This form of communication is already implemented. Continuing to use this

form of communication helps us accomplish all the tasks required to send and receive signals in a timely manner. Another factor which drove us away from this decision is the fact that we would need another expensive UWB in order to communicate from off board.

~~Obstacle Detection~~Sensors

As the formation moves about it will encounter obstacles. Detecting these obstacles with some type of sensor is the first step in avoiding them.

1) Indirect use of the step motors- We can use our step motor along with our relative positioning algorithm to compare whether the robot has moved since some previous sample. If the unit has moved to a point that we expect then there is no obstacle. On the other hand, if the robot is not where the program ~~excepts~~expects, then we know there is an obstacle in front of it. This method is a very shoddy way to implement an obstacle detection mechanism. In our project we want to detect obstacles before we get to them so that the robot and formation can avoid them entirely. This method would only know of an obstacle after it has run into it.

2) Video- This option requires that there be constant attention given to the images received. An operator would have to interpret the images and navigate around the obstacles manually or there would need to be some sort of sophisticated image processing which is beyond the scope of the project. This would be contrary to the goals of having an autonomous movement. While this method provides a good way to see exactly what is around the robot, it is expensive and unnecessary for this project.

3) Sonar- A system of detection using sonar has already been implemented on the robot. There are currently four sonar detectors on the front. Using these we can see up to

about 40 degrees off north of the field in front of the robot. After implementing two more sonar detectors on each side, we will have additional sight of the field around the rear of the robot. This is the best option because it is cheap and already mostly implemented.

Robot Processors

In order for the robot to carry out commands from the user the commands must be put into a language which the robot can understand. To accomplish this we need to decode the received commands using some sort of onboard processing unit.

1) Internal Processor – Each robot is equipped with an onboard processor. This processor is responsible for controlling the wheels, sonar, and any other designed peripherals. Unfortunately using the onboard processor is not a viable option. This processor was specifically designed only for factory equipment.

2) Use an external microcontroller- We have the option of using an external processor such as an Atmel processing board or equivalent. In order to use this microcontroller, it is required to build a separate power supply for it. Already having two supplies onboard, building a third would be excessive.

2) Use a laptop- Laptops are the equivalent of having an Atmel board, but they already have a battery inside them. We decided to go with a laptop primarily for the ease of use and compatibility with the current system.

Direction Recognition

All units must have knowledge of absolute and relative reference points in order to move successfully as a rigid body. For example: to have an initial movement north, all units must first be facing north. Also, to move as a rigid body each unit must know where they are relative to each other unit. To ensure a rigid body movement each unit must always be at a similar angle and distance away from one and another. The goal here is to develop a way for all robots to know their heading relative to some known reference heading.

1) Use a compass- Having a compass on each unit would ensure a similar absolute reference between each unit. The relative positions could be derived from the information provided by each compass. We will attempt to find a way to avoid using compasses on each unit.

2) UWB angle of arrival (AOA) feature- Each UWB unit has the option to have two antennas and provide an angle of arrival between two units. Using this feature would allow us to calculate relative angles and positions fairly easily. The downfall comes when we place the robots at random directions, the UWB has no absolute position reference. The only way that this method would be correct is if the robots are placed in a very specific initial direction. Due to the high possibility of error this method has been discarded. (See Appendix B for diagrams)

3) UWB and Predefined absolute direction using AOA- To solve the problem of not having an absolute reference we can place an extra UWB unit at a predefined position and use that as the absolute point of reference. This is impractical for two reasons. The first reason is because the UWB units are not useful at long distances. If we were to send

position information from the base UWB to the robot's UWB at distances greater than ~~50 (may be different)~~ feet, the accuracy of the units drops off significantly. The second reason is apparent when we consider costs. At $45K per unit, it would be financially impractical to buy an extra unit if other, cheaper, methods are available. (See Appendix B for diagrams)

4) GPS- Global Position is a technology which would allow control of the position of the robots through a satellite tracking system. GPS would solve the problem of relative and absolute references but sacrifice accuracy. Current outdoor GPS technology give positions of objects with an error factor of 3 feet. This technology may be useful for similar large scale experiments, but is not useful for our current project.

5) AOA and compass combination- The most feasible and economical solution is a combination of a compass and the UWB angle of arrival option. For most earth based applications the compass would provide an absolute point of reference at a very low cost. The UWBs AOA, as stated above, solves the relative references problem. We would only need to purchase a single compass and use the relative information to derive other necessary information. Using this combination of technologies gives us a cheap and practicable solution.

Robots

Which robots to use for the project was one of the first questions our group was faced with.

1) The idea of building our own robots was quickly discarded. Building a reliable robot is, in itself, a tremendous project that we did not wish to tackle. Instead, we wanted to focus on the problem of formation.

2) After, talking to our design advisors, we were given the opportunity to use robots that the school owned. Initially, we wanted to use the small and mobile robots known as AmigoBots. The AmigoBots are manufactured by ActivMedia and have many interesting attributes including obstacle avoidance and a graphical user interface. However, when we realized that the Ultra Wide Band units were rather large and heavy, it became clear that the AmigoBots could not hold the extra weight.

3) So we had to upgrade to the larger Pioneer Robot, also manufactured by ActivMedia, that can carry loads up to 30kg. The Pioneer comes with most of the same software and programming interfaces that the AmigoBot came with, which makes the Pioneer a desirable alternative to the smaller AmigoBots.

GUI

One goal of the project is to create a user interface that is easy to use. In order to achieve this goal, we want to make the GUI as simple as possible while maintaining a great deal of user input in the action of the robots.

1) The first issue is whether the user should enter the desired position of the formation by clicking on the screen, or by entering in the new coordinates. In the end, our group decided to go with both options. In this way, the user can be very precise by entering in the desired coordinates, or they can be more informal by simply clicking the screen. It seems that both functions are highly desirable.

2) The second issue was whether the interface should display the range of the sonar sensors. There occurs a problem when all of the sensors are displayed because the sensors will often overlap or will sense an obstacle where there is actually another robot. However, by displaying the sensors, the user will have a clearer sense of the position of potential obstacles. Again, our group has decided on a compromise. The interface will display only those sensors on the periphery of the formation. In this way we get the best of both worlds, however, there may arise some problems when determining which sensors to display~~. In this way we get the best of both worlds, however, there may arise some problems when determining which sensors to display. In this way we get the best of both worlds, however, there may arise some problems when determining which sensors to display. In this way we get the best of both worlds, however, there may arise some problems when determining which sensors to display.~~

3) It is also important to consider which parameters should be fixed and which should be input by the user. Examples of various parameters include, the number of robots in the system, the formation of the robots, the distance between each robot in the

formation, the initial direction that the robots are facing, the location of the formation when it is first formed, time spent attempting to reach destination before giving it up as impossible (timing out), and the maximum speed the robots should travel. Some of these parameters will always be user defined due to the goals of the project, such as the number of robots in the system, the formation that the system will maintain, and the distance between robots. Other parameters, such as the initial direction and the location of the initial formation will be fixed because it does not add value to the system to change these items. And finally, some parameters will have defaults that can be modified by the user in order to make the system more versatile and robust, such as the length spent searching before timing out and the maximum speed the robots will be allowed to travel~~how fast the robots can travel~~.

Formation Reference Point

Formations require a specific reference frame to determine relative positions and angles. There are a few ways to accomplish this task.

1) Centroid/Center of mass – There is a simple algorithm to calculate the center point that will work for our purposes. This algorithm adds up the x coordinates of all the bots and divides by the number of bots in the system. Then it does the same thing for the y coordinates. The benefit to this method is to have a reference point within the formation, which will facilitate the problem of getting into formation. The simple algorithm to calculate the center of mass is very useful to get

2) Master Robot – The master robot would carry the axis and point of reference within it. One robot will be designated as the (0,0) in order to facilitate building a grid of the environment and all the robots in it. A protocol can be set up which would automatically assign a new master robot if the first one malfunctions.

Discussion of Algorithms

There are three algorithms that our project needs in order to function as desired. These three algorithms include a method for getting into formation, a method moving from the current position to the desired position, and a method for avoiding obstacles

1) The first of these, getting into formation, is quite easily solved. We thought of two different methods for positioning the robots in their formation. The first method we thought of was to assign a position to a robot randomly. Then the other robots would move towards the first robot in small steps, one at a time, while continually checking for obstacles. In this way they could avoid running into each other. They would stop once they were within a certain distance from the first robot. Then each robot would move, one at a time, into formation. This could be accomplished by moving the closest robot into the next position of the formation.

The second method we thought of was to use a variation of the Assignment Algorithm. This well-defined algorithm is used to assign a number of items to the same number of positions by finding a minimum cost solution for the system. Our system will implement Munkres' Assignment Algorithm (also known as the Hungarian Algorithm) described at the following website:

http://campus.murraystate.edu/academic/faculty/bob.pilgrim/445/algorithms_7.html. Our system will use the distance formula as the weight factor to determine the best fit.

2) The second algorithm to be considered is the method of moving from one place to another. First, the system will calculate the desired coordinates. The formation will then attempt to get to the new location by moving in a straight line while using obstacle avoidance techniques and a behavioral approach to keep from running into things. The distance traveled will be measured by using the length calculated by the rotation of the wheels. This straightforward approach is very simple and will be consistent for any formation, making it unnecessary to consider many alternatives.

3) The third method that needs to be considered is the problem of obstacle avoidance. This is a dilemma that has been highly discussed by many in the field of robotics and artificial intelligence. One method that our group has discussed is very simple~~, and should prove effective,~~ though it is not very efficient. This method involves searching for a set amount of time in one direction until a way around the obstacle is discovered. If no clear path is discovered within the set time, the formation will return to its original position and then search in the other direction for a set amount of time. If no clear path is found in this time, the system gives an alert of an impossible obstacle and gives up. Although this method may waste a lot of time and effort, and may not find a clear path that exists just outside the boundaries examined, it will work in simple cases.

While the above algorithm is very simple, it is more efficient to utilize an obstacle avoidance algorithm that involves looking ahead. Any such avoidance algorithm will require the use some mapping function. Thrun et al.[1] ~~[2]~~ describe two different types of

[1] Thrun, Sebastian et al. "Map Learning and High-Speed Navigation in RHINO." Arificial Intelligence and Mobile Robots. Ed. David Kortenkamp. Menlo Park: 1998. 21-52.

maps. The first is map-based and has vertices and edges stored in a table. The second is grid-based and uses a grid to represent the environment. Our system will be using a grid-based mapping function because grids are easier to maintain and should correspond well with our grid-based graphical interface.

There exist several methods for obstacle avoidance that look ahead and try to navigate around obstacles while in motion. In a 1991 article, Borenstein and Koren identify three common types of algorithms as well as describe a technique of their own~~.~~[~~3~~]
.[2]

The first of these methods is known as the edge-detection method. In this case, the system attempts to detect the edges of an obstacle and then steer around it. Often, the robot must stop in order to sense its surroundings. In this type of obstacle avoidance, it is important that the sensor data is accurate. This type of algorithm is best suited towards an interpretation of visual or video data, and hence is not suitable for our project.

The second method involves using a certainty grid. A certainty grid is an array that represents the robots environment. Each element of the grid contains a value (the certainty value) that represents the probability or confidence that there exists an obstacle in that position. Whenever the sensors detect an obstacle, the value in the grid that represents that obstacle's location is incremented. Then the system would maneuver the robot in a manner that avoids the locations with high certainty values. This method

---

~~[3] [33] Thrun, Sebastian et al. "Map Learning and High-Speed Navigation in RHINO." Arificial Intelligence and Mobile Robots. Ed. David Kortenkamp. Menlo Park: 1998. 21-52.~~

~~[3]~~ [2] Borenstein, J., and Y. Koren. "The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots." IEEE Journal of Robotics and Automation 7.3 (1991): 278-288.

avoids the necessity for high accuracy in sensor data, making it more effective for sonar sensors.

Borenstein and Koren identify one method to navigate obstacles. In this case, the potential field is used, but there is an additional grid, called the polar histogram, that helps to guide the robot's movements. The polar histogram breaks the environment into segments of five degrees. In each segment, the polar histogram adds up the certainty values in that range, producing a new certainty value for each segment. The segments with certainty values below a pre-defined threshold are considered safe to steer through. Although perhaps the most complicated, we feel that this method provides the most accurate reading of the robots surroundings as well as the most thorough method for navigating and determining the next move.

The final method uses potential fields to steer the robots around obstacles. When an object is detected, the system uses the distance the object is from the robot and calculates a vector that acts as a force, pushing the robot away from the obstacle. At the same time, there is an imaginary force pulling the robot towards its destination. The direction the robot moves is determined by the sum of all the vectors. One major drawback of this method is that in environments with many obstacles, the robot tends to have many fluctuations if the repulsive vectors become large. However, we feel that the behavioral technique involved in this algorithm makes this method highly desirable.

Coordinate system

There are two choices when it comes to a two-dimensional coordinate system. The first is to use Cartesian coordinates and the second is to use polar coordinates. The

Cartesian plane is more desirable to our project because we will be calculating many distances and will have many points on the plane at all times. One point for each robot and each wheel, and one point for the ~~center of the formations, and one for the center of the~~ new location the robots are moving towards. The ease of use for calculating distance and angles makes Cartesian coordinates highly desirable.

Orientation vs. Rotation

There was some discussion amongst our group about the best way to change the direction that the robots are facing before they move to the next desired location.

1) We originally thought that it would be best to turn by rotating the robots in place until they are facing the desired direction. This way we could save time and keep from making unnecessary movement. Furthermore, when examining the process of rotating the formation as a whole, it was obvious that the rotation would be difficult because some robots would need to travel farther and faster than other robots of the formation whenever a turn is made. Additionally, calculating the movements for an effective turn would be far more difficult than the calculations for a simple rotation in place. However, a simple rotation would cause a change in the orientation of the formation whenever the formation is not symmetrical. It is not worth the effort to determine when a formation is symmetric so that we can take advantage of the shortcut to rotate the robots in place.

2) The change in orientation is obviously an undesirable consequence of simple rotation, so we discarded the idea in favor of the rotation of the entire formation as a single unit as described above. Although this method may result in more movement and take more

time, it is consistent and will ~~always~~ work to change the direction the formation is facing regardless of the shape of the formation.

Algorithms

Getting into Formation

The problem of getting into formation will utilize the Hungarian assignment algorithm and will move each robot in small increments to avoid collision.

Given:

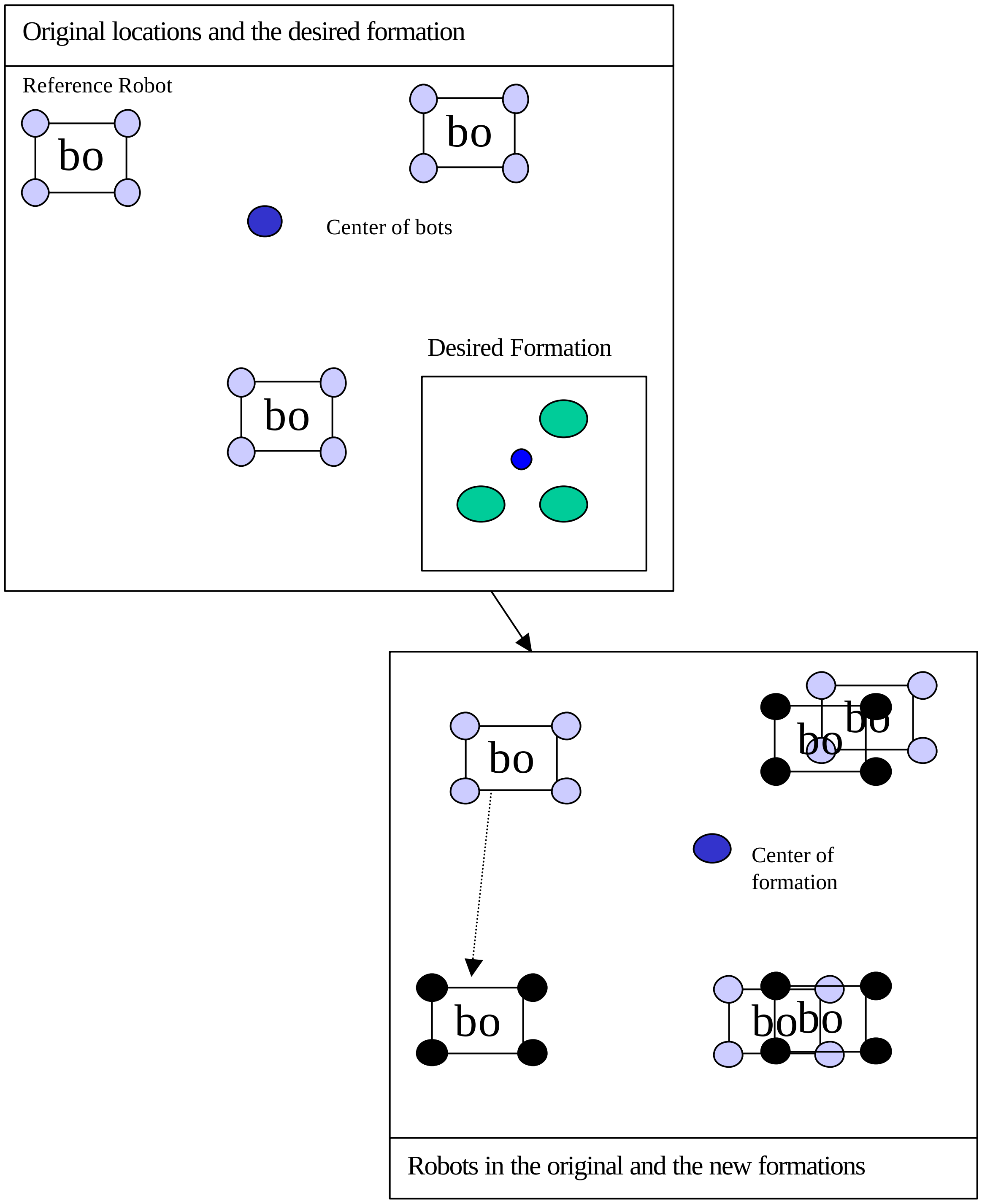
1. The distances and angles between all the robots (from the UWB)
2. The distances and angles between the robots of the desired formation

Steps:

1. Select one robot to be the reference robot (0,0)
2. Calculate the center of the robots' current positions ((Sum of all x's / number of robots), (Sum of all y's / number of robots))
3. Overlay the center of the current position and the center of the desired formation (see diagram)~~.~~
4. Create an nxn matrix for n robots
5. Calculate the distance to move each bot to each position – this is the cost function for the assignment algorithm
6. For each row in the matrix, find the smallest element in each row and subtract that amount from every element in the row
7. If there is no marked zero, find a zero and mark it
8. Repeat step seven for each row
9. "Cover" the columns containing a marked zero. If n columns are covered, go to step 13. If not, go to step 10.
10. Find an uncovered zero and prime it. If there are no marked zeroes in this row, go to step 11. Else, cover this row and uncover the column with the marked zero. Repeat until there are no more zeroes. Save the smallest uncovered value and go to step 12.
11. Construct a series of alternating marked and primed zeroes until there is a primed zero without a marked zero in its column. Then unmark each marked zero and mark each primed zero. Uncover and un-prime everything else and return to step 9.
12. Add the value from step 10 to every element of each covered row and subtract it from every element of each uncovered column. Go to step 4.
13. Done – Assignment pairs are indicated by the positions of the starred zeroes.

Output:

1. The bots in the appropriate formation


Original locations and the desired formation
Reference Robot
bo
bo
Center of bots
bo
Desired Formation
bo
bo
bo
Center of formation
bo
bobo
Robots in the original and the new formations

Obstacle Avoidance

A complex but highly probable scenario involves manipulating the system around an obstacle detected in the formation's path. This algorithm uses a behavioral approach.

Given:

1. Current location on the grid
2. Sonar reading of the environment giving distance to nearest obstacle

Steps:

1. Calculate "forces" from objects detected by sonar
2. Add together all the vector forces, including the force pulling the formation toward its destination
3. Steer the robot in the direction indicated by the calculated forces

Output:

1. The robots safely move to the desired location

Moving Towards a Desired Location

Moving from one location to another is perhaps the most basic, and most important applications of our system. The movement algorithm is very simple, but it uses the other behaviors to react to the environment. The output will be the original desired location if a straight-line movement can reach the position. However, if that is not possible, the output will be ~~the~~ newer and closer coordinates that will allow for obstacle avoidance or rotation as necessary.

Given:

1. Current coordinates
2. Coordinates of desired location

Steps:

1. Move in a straight line towards the new coordinates
2. Monitor obstacle avoidance behavior
3. Monitor constant distance and angle behaviors

Output:

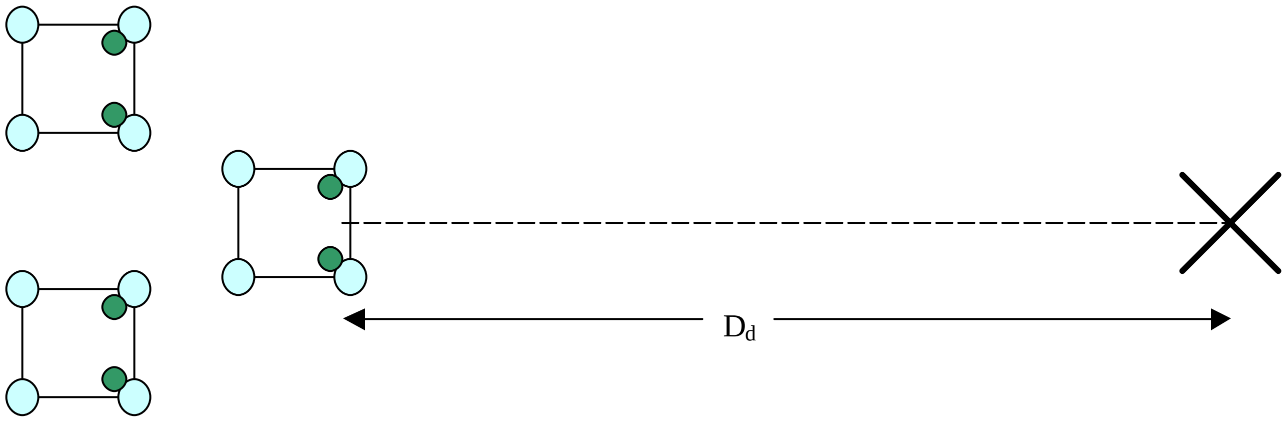
1. Coordinates to head towards

Behavior Description

The robots will be programmed by a set of behaviors that will constantly monitor the status of the formation and adjust accordingly. Each formation is controlled by a gain factor. This gain factor will determine how much strength each behavior will have over the actions of the robot. For example, if it is more important to maintain a rigid formation than to get to the destination, then the distance and angle behaviors will have a higher gain than the destination behavior. The next few pages outline the four most important behaviors of the system.

Destination Behavior

GOAL: To get the lead robot to the destination point

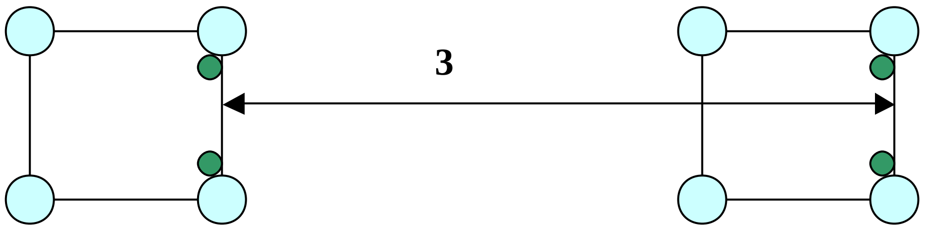


D_d = Distance from current robot position to ending destination.
G_g = Gain factor for the velocity to the goal destination.

$V_g = G_g * D_d$

Sustained Distance Behavior

GOAL: To maintain a predefined distance "d" from other robots.

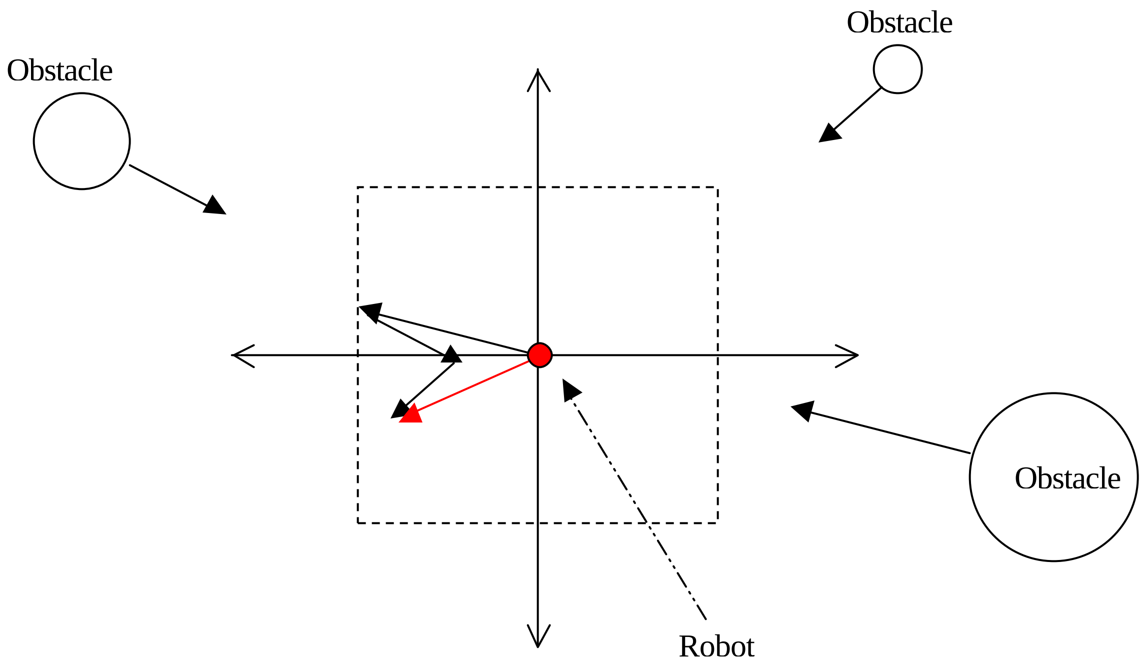


D = ideal distance to maintain between two robots
Δ_d = difference of actual from ideal distance (may be negative)
G_d = gain factor for distance
V_d = instantaneous velocity

Δ_d = d – (actual distance calculated from UWB)
$V_d = G_d * \Delta_d$

<u>Obstacle Avoidance Behavior</u>

Goal: All vectors from the obstacles will add to the overall force exerted on the system
and dictating the velocity and direction of the system



The red vector is the resultant velocity of the robot after all obstacle vectors are considered.

Let V_o = instantaneous velocity due to objects

Let F_n = velocity vector due to the Nth obstacle (xyz coordinates)

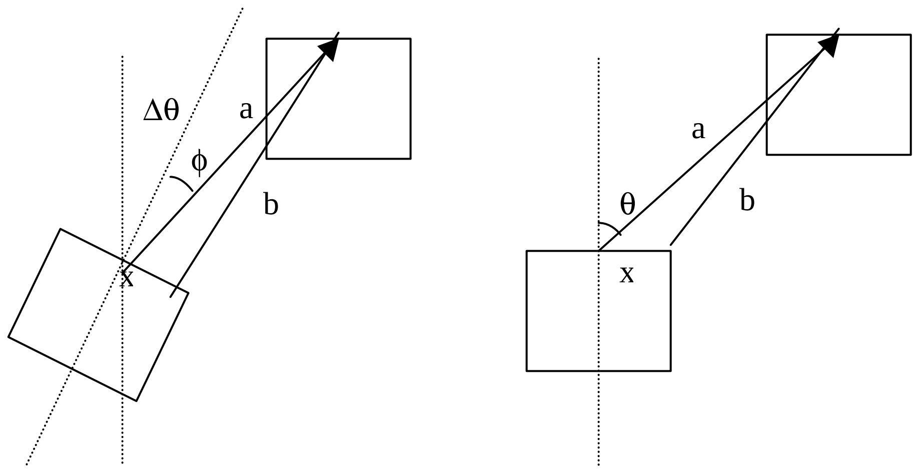
Let G_o = gain factor for obstacle avoidance

$$V_o = F_n * G_o$$

$G_o \rightarrow$ sensitivity for obstacle avoidance behavior

Relative Angle

Goal: Maintain a specified angle from a reference position using UWB angle of arrival.



θ = ideal angle of arrival
ϕ = actual angle of arrival
Δθ = the difference between the ideal and the actual

$\phi = 90 - (\cos^{-1} ((a^2 + x^2 - b^2) / (2*a*x))$
$\Delta\theta = \theta - \phi$

$V_\theta = \Delta\theta * G_\theta$

The robot needs to turn Δθ to maintain the desired (ideal) angle at the rotational velocity indicated by V_θ.

~~**Formation Rotation Error Algorithm (FREA)**~~

~~**The algorithm outputs the ratio of the number of steps between the outer and inner wheel. Realizing that this ratio might not always end up being a perfect integer, we will have to implement a correction algorithm. The input to the FREA will be mod of the step ratio. The output will be a correction factor. This correction factor will then be the input to the TCA, which takes into account other errors such as: distance between robots, relative angles and referenced angles.**~~

~~**Relative Distance Error Algorithm (RDEA)**~~

~~**This algorithm is very similar to the FREA. It inputs the actual and desired distances between two robots and outputs a correction factor. This correction factor in then becomes the input for the TCA.**~~

~~**Total Correction Algorithm (TCA)**~~

~~**This algorithm takes into account all of the error factors from each other algorithm and calculates the correction movement of the wheels.**~~

Test Plan

Testing the overall functionality of the system obviously requires a breakdown into individual subsystems. These systems include, but are not limited to: the program (code), the graphical user interface, the functionality of the robots, the effectiveness of the wide band communication system, and the quality of movement of the formation. Listed below are the predicted methods of test for each subsystem. The methods discussed below are purposely vague ~~do~~ due to the inevitability of change in subsystem designs

The Program:

The program code is the key element of all interactions between the graphical interface, the user, and the robots. Code, like this project, can be broken down into subsystems. Each subsystem has its own particular contribution to the overall program. These subsystems are algorithms which have an expected output based on their individual inputs. Thus, when we test the performance of the program, we will test each individual subsystem by observing the ~~known~~ output for a given input.

The GUI:

The Graphical User Interface (GUI) is the portion of the system that is most accessible to the user. There are two main issues that need to be tested for the GUI. The first is clearly the accurate interaction between the commands given by the user and what the robots actually do. The GUI is a subsystem of the entire program and will be tested as described above. The second key component in a successful user interface is the relative ease of operation. The GUI should be easy to comprehend and manipulate. To test this we will employ the services of our peers. The best way to determine whether a GUI is good enough is through peer evaluation~~. ~~, ~~T~~thus the testing of our GUI will be done by our peers. We will take into consideration their recommendations and modify our interface accordingly.

The robots:

In order for our entire system to be complete the robots must have certain capabilities which are intrinsic to the code. For example, if the user calls out for a particular movement which requires the robot to rotate 45 degrees, the robot must have the ability to do just that. Given current software our team can test all of the capabilities of the robots. While testing we can see which directions the robots are capable of moving in and also whether or not they can rotate.

Ultra Wideband Communication:

The primary purposes for having ultra wideband communication is to gain information such as how far apart the robots are relative to each other~~-~~ and their relative angles~~and to communicate to the central processing unit~~. To test these two things we are going to enter a known distance between two robots and then confirm the actual distance manually. This will give us an idea of the amount of deviation we will be working with throughout the project.

Quality of movement:

Due to the fact that quality is not entirely quantifiable, our tests will be based on accuracy and efficiency of movement. For example, we want to avoid jittery and extraneous movements.

Timeline

December

5th – Experiment with test capabilities of the UWB

By 13th – (Alan) Understand the Robots, and how they are controlled

By 20th – Test Angle of arrival on the UWB units (two antenna system)

- Test Robots with UWB on them (distance control)
- Ability to calculate the geometric center of mass of any formation, so that the center point can be found. The center point is then used as the part of the formation which should end of at the exact location of the desired destination
- Research on adding more sonar to the existing robots to allow for better obstacle detection
-

January

By 10th – Begin Coding Algorithms~~Designed~~

- Test the compass on the Robot formation direction verification
- Continue testing two Robots manually with UWB units for distance control

By 24th – Test One Robot with program

    - Must be able to move from point A to point B, and if an obstacle is in the way must try to get around it or must alert the user to the futile attempt

- Start implementing GUI to allow for a user definable formation

February

By 7th – Test Two Robots with program

    - Must be able to move from point A to point B, and hold a formation
    - Test for clockwise or counterclockwise rotation of the formation when necessary

- GUI finished
    - Test with Robots
- Test sonar with one Robot with program

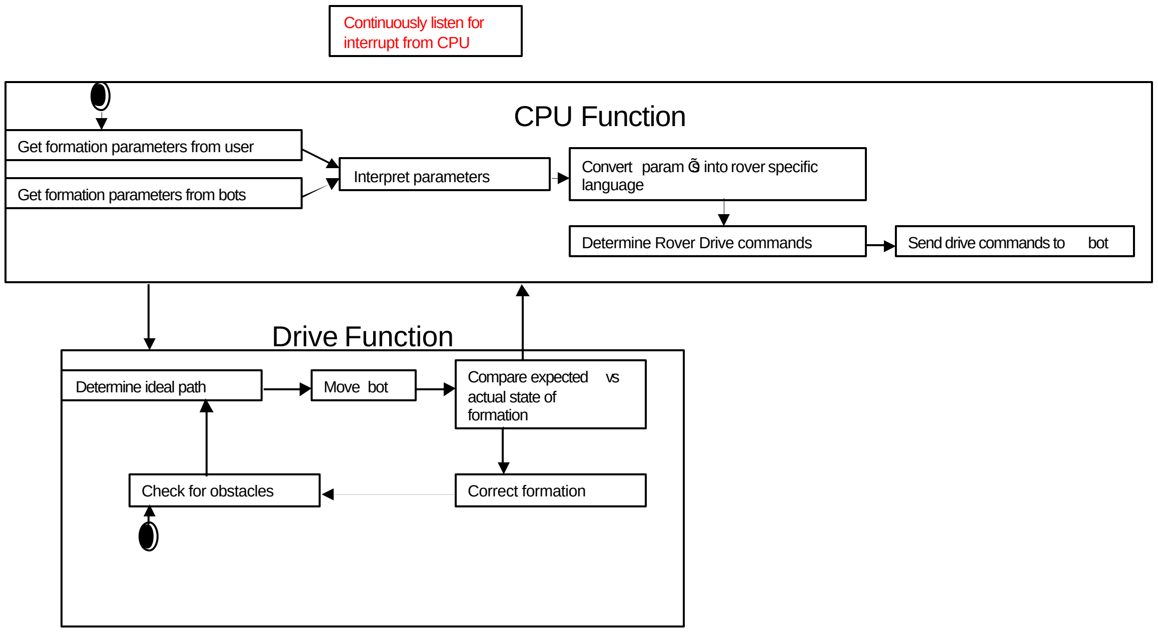
By 21st – Be able to calculate the angle and distance from the robot must travel when the user clicks a desired destination on the grid.

- Test Three Robots with program
    - Maintain speed and orientation
    - Obstacles
    - GUI displays movement

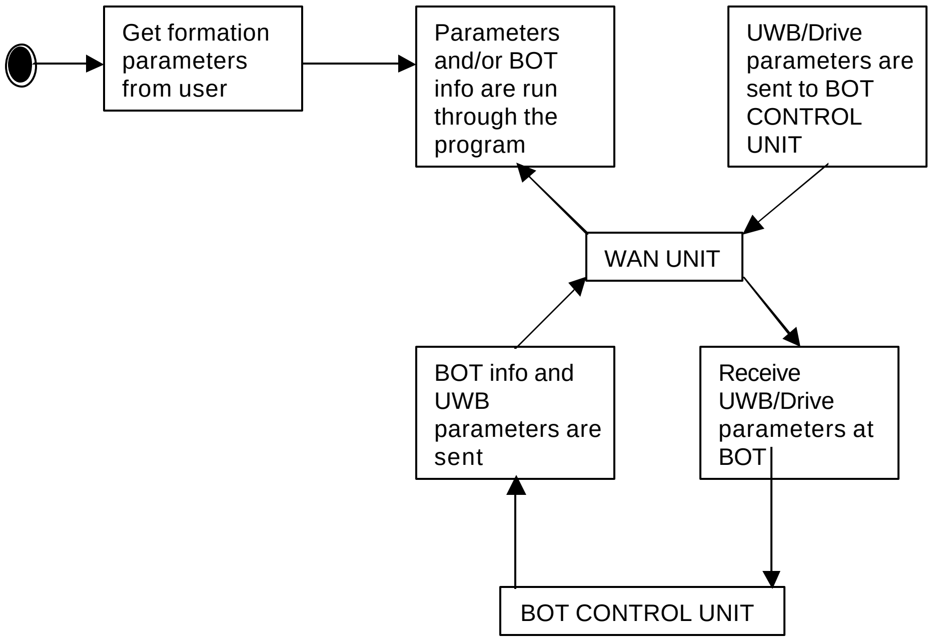
March

Finished with ~~Designing~~ Implementation – now test

Appendix A
Functionality
and
State Diagrams

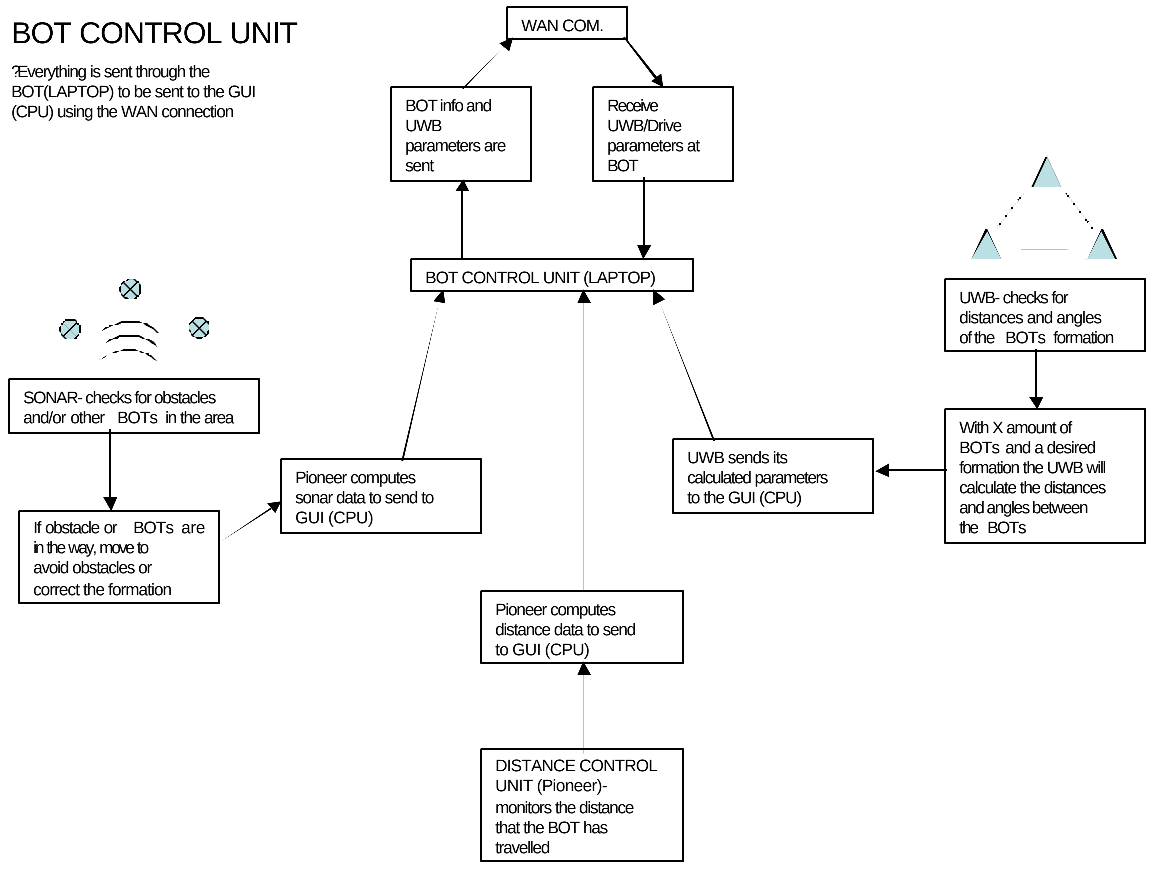

Continuously listen for interrupt from CPU
CPU Function
Get formation parameters from user
Get formation parameters from bots
Interpret parameters
Convert param's into rover specific language
Determine Rover Drive commands
Send drive commands to bot
Drive Function
Determine ideal path
Move bot
Compare expected vs actual state of formation
Check for obstacles
Correct formation

GUI -CPU Function

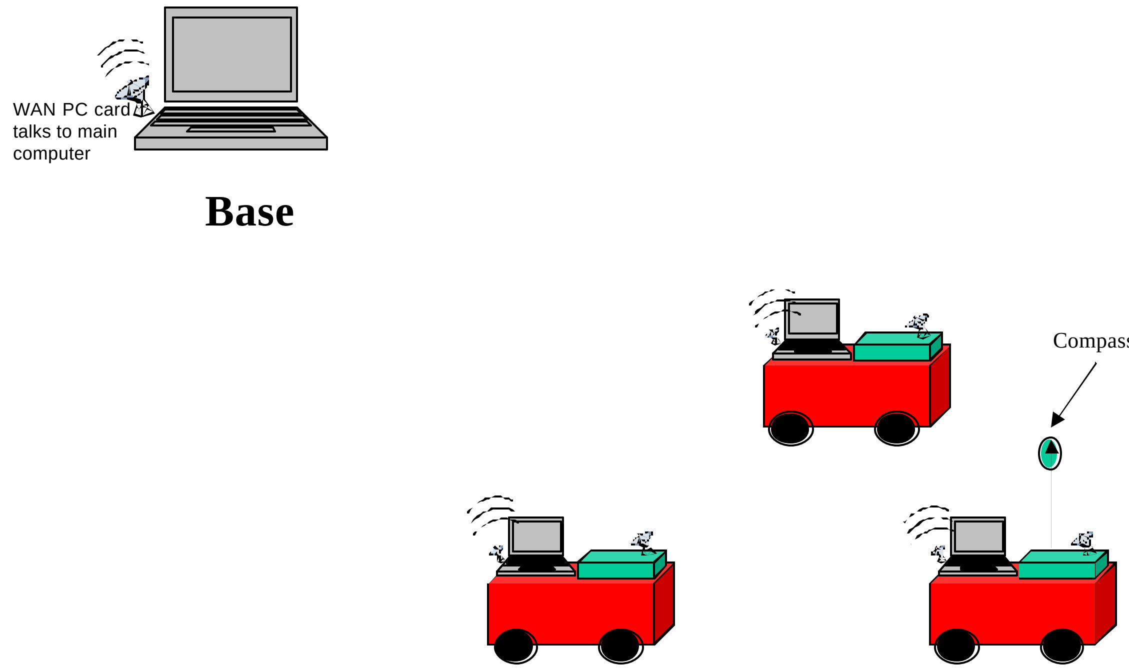

Get formation parameters from user
Parameters and/or BOT info are run through the program
UWB/Drive parameters are sent to BOT CONTROL UNIT
WAN UNIT
BOT info and UWB parameters are sent
Receive UWB/Drive parameters at BOT
BOT CONTROL UNIT

BOT CONTROL UNIT

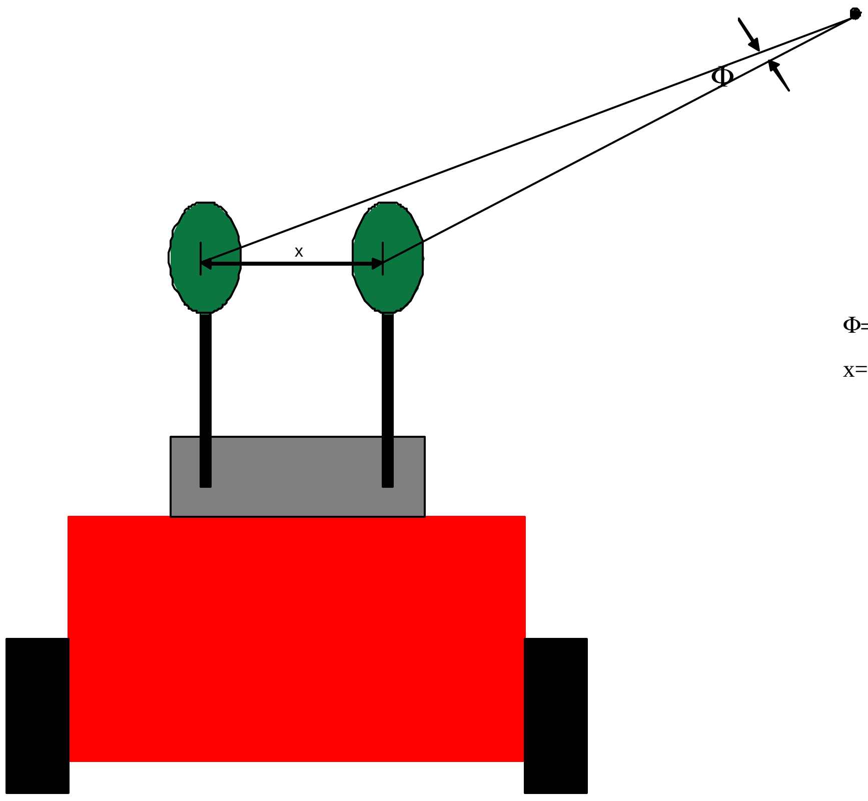
?Everything is sent through the BOT(LAPTOP) to be sent to the GUI (CPU) using the WAN connection


WAN COM.
BOT info and UWB parameters are sent
Receive UWB/Drive parameters at BOT
BOT CONTROL UNIT (LAPTOP)
UWB- checks for distances and angles of the BOTs formation
SONAR- checks for obstacles and/or other BOTs in the area
With X amount of BOTs and a desired formation the UWB will calculate the distances and angles between the BOTs
UWB sends its calculated parameters to the GUI (CPU)
Pioneer computes sonar data to send to GUI (CPU)
If obstacle or BOTs are in the way, move to avoid obstacles or correct the formation
Pioneer computes distance data to send to GUI (CPU)
DISTANCE CONTROL UNIT (Pioneer)- monitors the distance that the BOT has travelled

Overall System



Appendix B
Angle of Arrival Diagrams

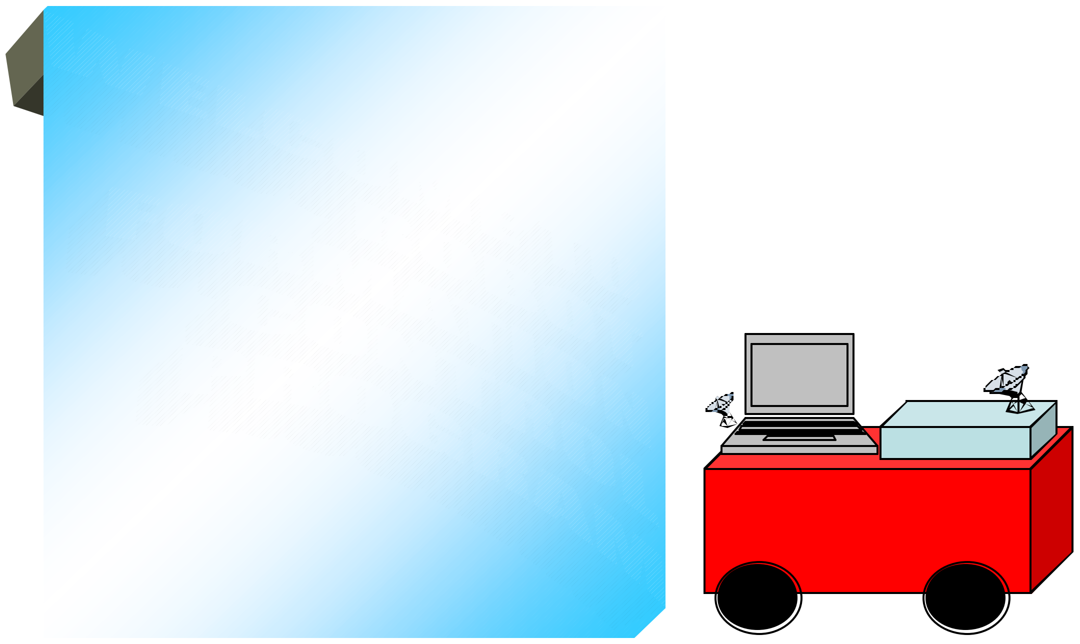


Φ=Angle of Arrival

x=Distance between antennas

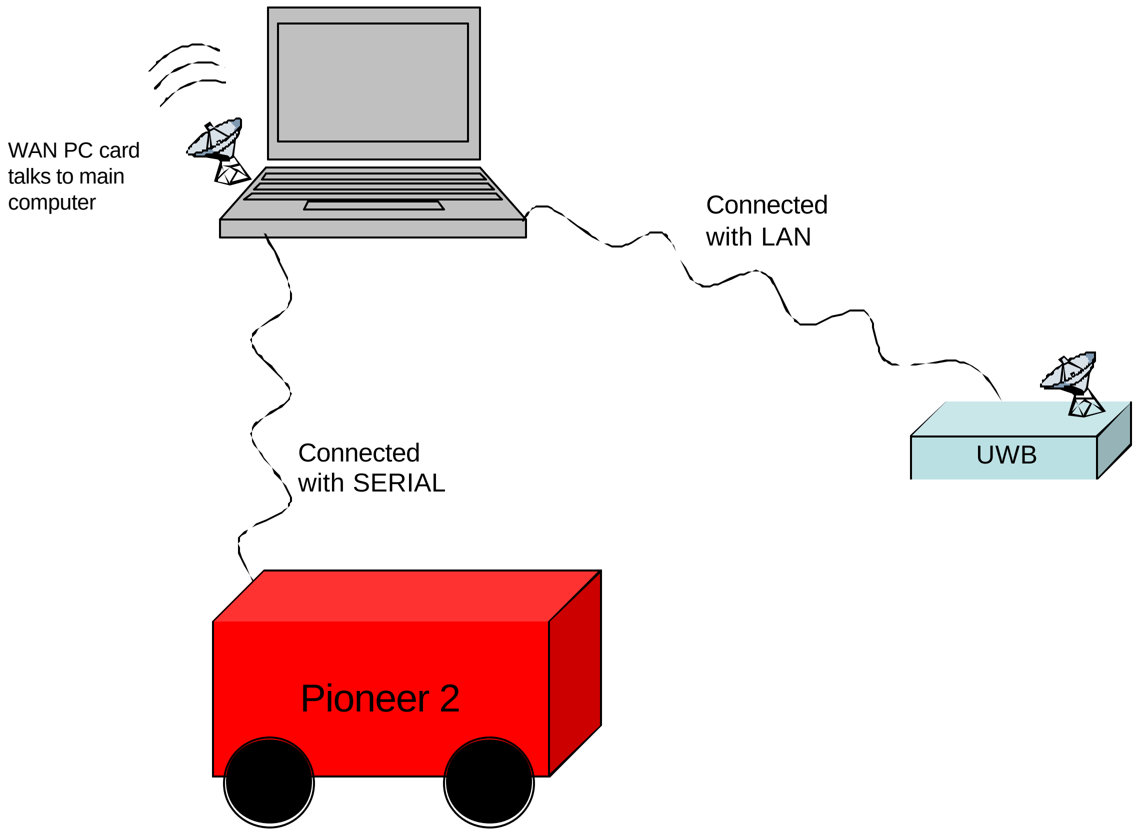
Appendix C
User Manual

User Manual



1. Setup

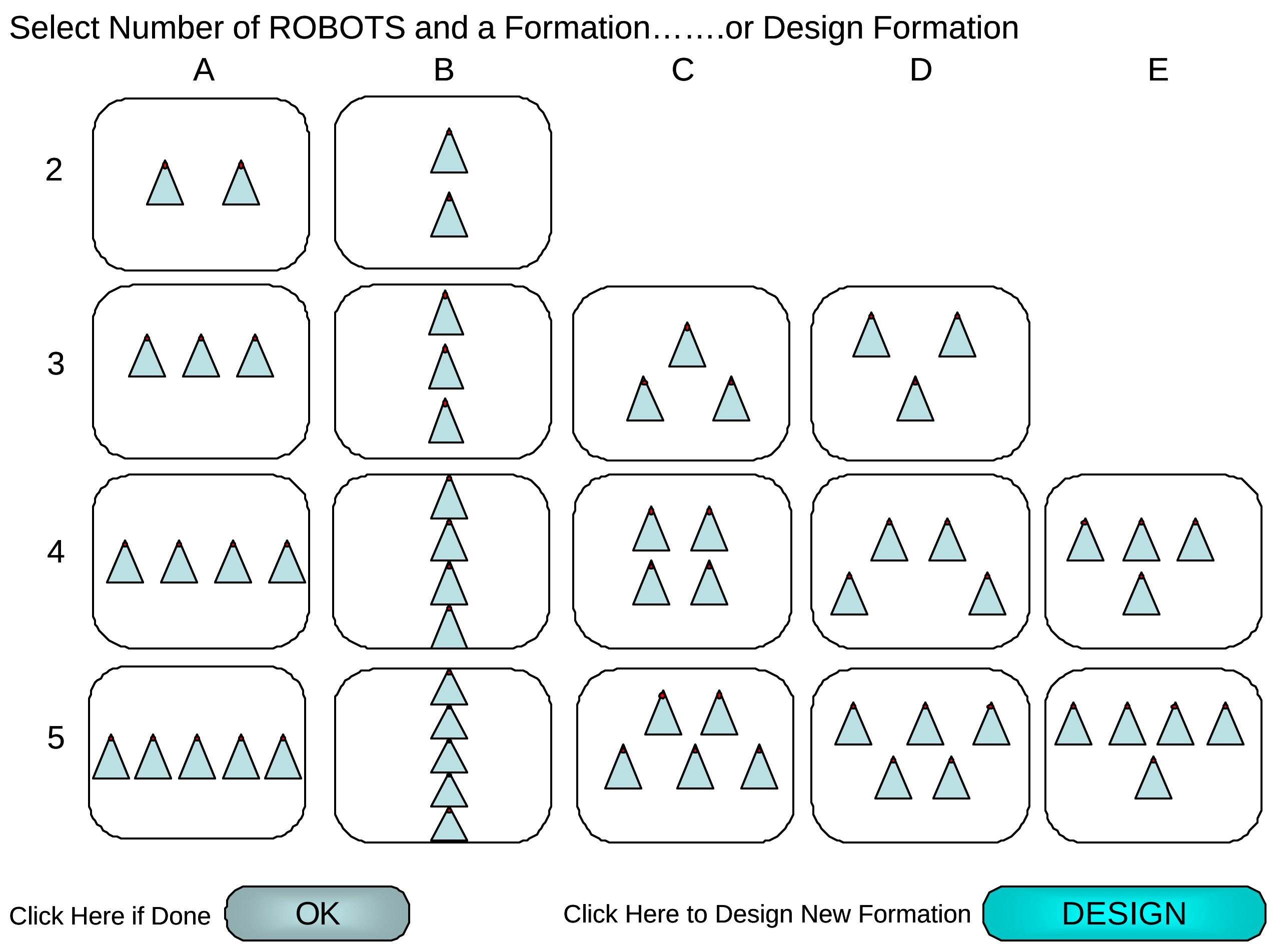
Using Serial cord, connect the laptop with the Pioneer robot. The Pioneer serial port is located on the back by the battery bay. Next connect the Ultra Wide Band unit to the laptop using the Ethernet LAN cord. Last of all, insert Wireless Network card into the PCI slot. View diagram below for help.



2. Before Operation

Before you can begin giving commands to your robots via a GUI program on your main CPU a few functionality checks should be made. Check to see that all battery packs mounted on the robots are fully charged and that the robots are turned on. Also, make sure the laptop mounted on each robot is turned on with a fully charged battery as well. The laptops will function as communicators between the GUI run on the main CPU and the robots. Once this is done switch on your main CPU and begin the program. You are now ready to begin directing your Pioneer robot formation!

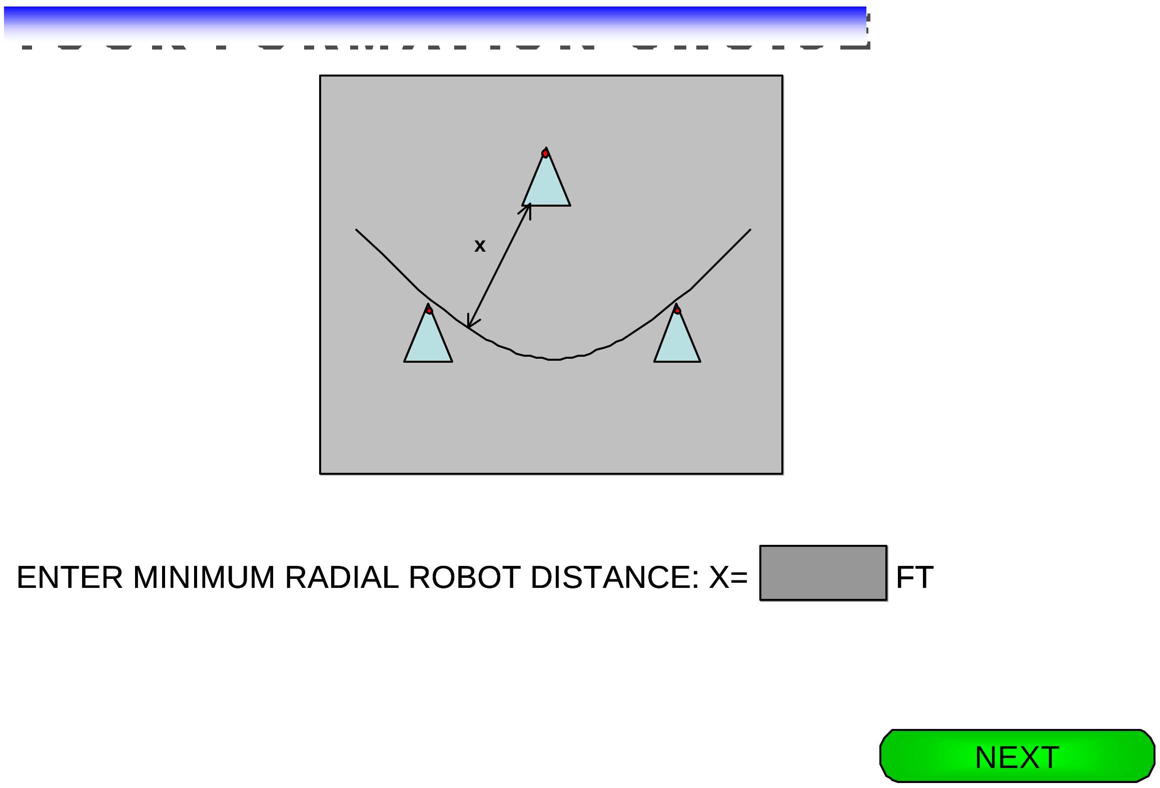
3. Defining a Robot Formation


Select Number of ROBOTS and a Formation…….or Design Formation
A
B
C
D
E
2
3
4
5
Click Here if Done
OK
Click Here to Design New Formation
DESIGN

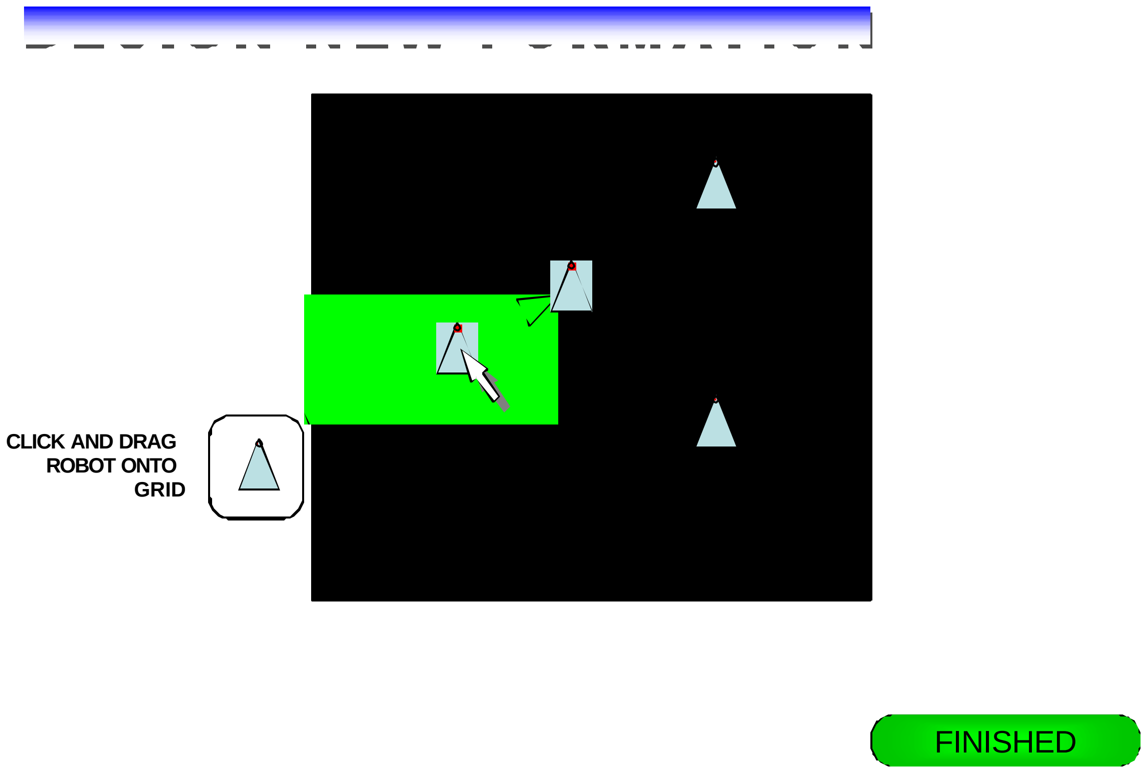
Once you have opened your Robot Formation program, you will be brought to a page looking similar to the screen shot above. Click on the formation you wish to choose, and then click **OK**. By selecting one of the predetermined formations, you are choosing the number of robots and formation type portrayed in the picture. If you wish to design your own robot formation, click on **DESIGN** and you can create your own formation. This will be explained later in the section.

3.1 Choosing a Predetermined Formation

Once you have selected your formation choice, you will be taken to a screen which looks similar to the screen shot below. You will be asked to define a **MINIMUM RADIAL ROBOT DISANCE**. This distance will be the minimum distance between any two robots. It is recommended to keep a minimum distance of one and a half times the robots' length between your robots. Do not worry about the other distances, the program will calculate them for you.



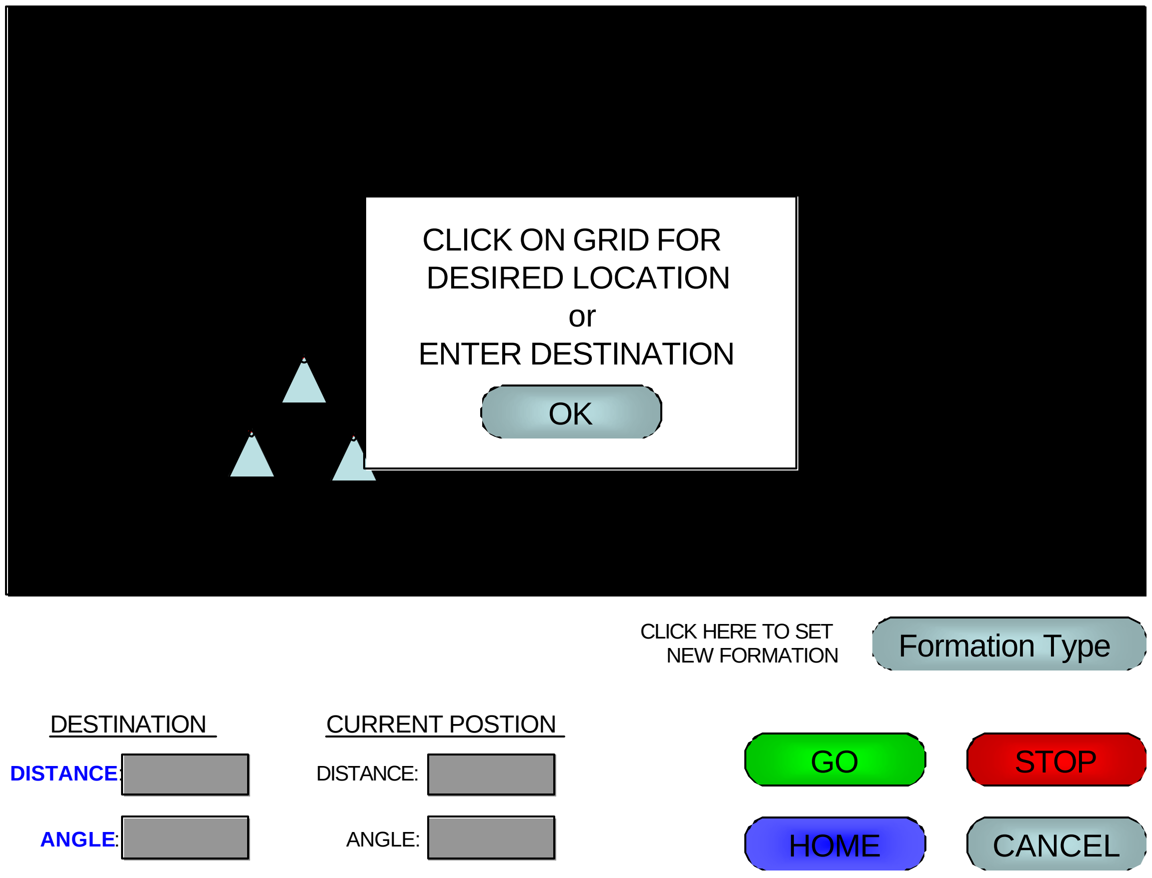
Once you have made your selections, click the **NEXT** button at the bottom of the window. This will close the window so that only the main screen with the grid and main user options will be viewable. As soon as you have clicked on **NEXT**, the robots will get into the formation you have just designated. At any time if you are unhappy with your decisions and wish to reselect the number of robots, distance between them, or formation type, you may click the **Formation Type** button located on the bottom right side below the grid on the main page.



3.2 Designing your Own Formation

If you do not want to select one of the predetermined formations, click on the Design button shown in the screen shot located at the beginning of section 3. Now, you will be taken to a new screen looking similar to the one above. To the left of the grid is a button with a picture of a robot on it. To add a new robot to your formation simply click the button, and while still holding down on the mouse, drag the robot to the place on the grid where you wish to place it. Once you have put the robot in the proper location, release the mouse button. Continue this step until you have the number of robots you want, and in their proper positions on the screen. If you wish to relocate a robot, once again click on it and drag it to its new location. To remove a robot, click on it once and then hit the delete button. When you have finished designing your formation, click on **FINISHED.** You will be taken to the "Your Formation Choice Page." If you have any questions on this part, refer to Section 3.1.

4. Moving your Formation

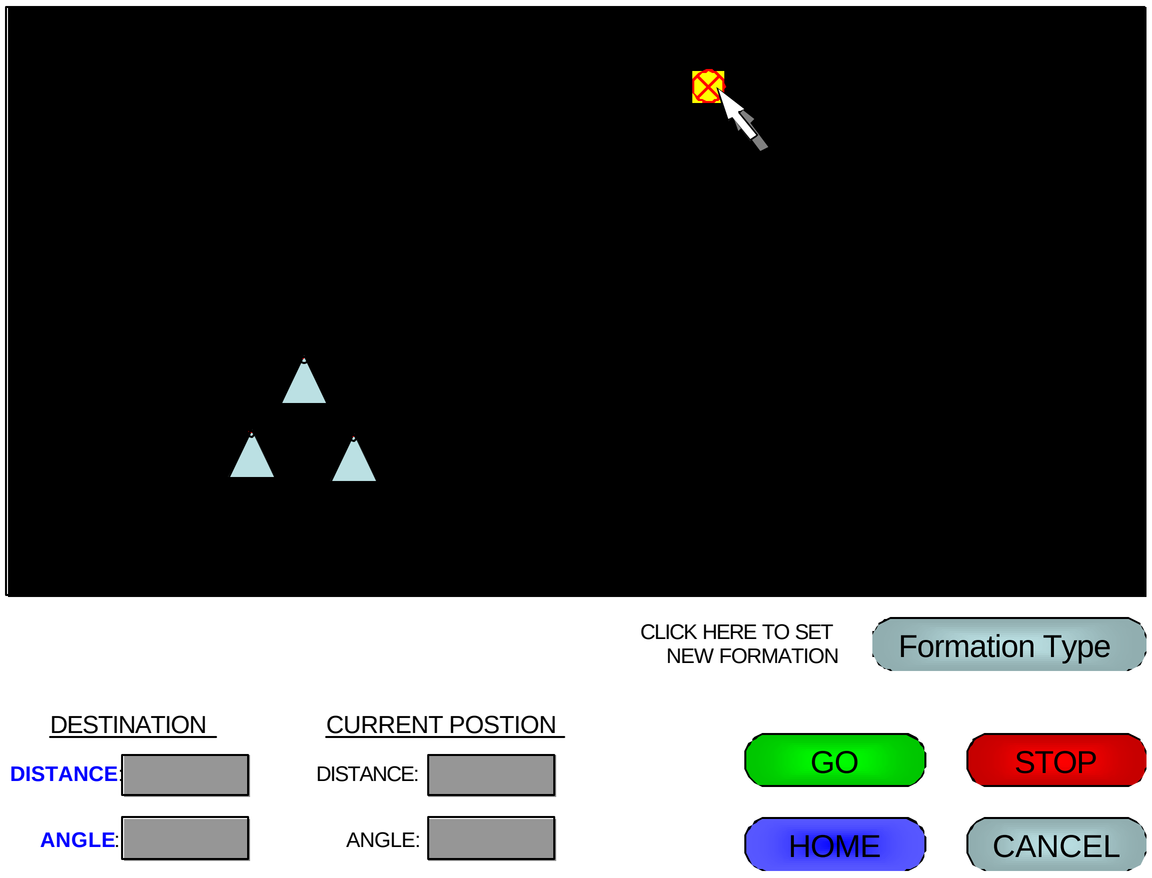


4.1 Click and Move

For beginning users, and when an exact destination for the robot formation cannot be determined, the grid is the recommended option. Most of your window is taken up by a grid, which will show the location of your robots. The grid has limited ability and can only represent the areas detected by the robots. When the robot formation originally shows up on the screen, it will represent the formation's relative position.

Once you have entered your desired robot formation and you can see the robots on the grid, you are ready to move the formation. While looking at the grid, sonar sent out by the robots should be seen (not shown above). Only the sonar on the periphery of the formation will be displayed, so as to not cause confusion. The longer sonar rays mean that there is no obstacle in the formation's path, and the shorter rays mean that something is in the way of a clear path. To actually move the formation, click somewhere on the screen and the robots will attempt to move to the new location. When using the click and move function of the program, you do not need to specify an angle for the robots to travel. The formation will attempt a straight line of travel to get to the desired location if possible and the front end of the formation will end up covering the place on the screen clicked by the mouse.

If it is not possible to get all robots to the new location, you will receive a warning message. Also, don't worry if your formation does not get to the new location immediately. It may take a while for a clear path to be found, and the time out mechanism will not come into effect until either all options have been exhausted, or the time allotted for the movement has expired. You will be able to view on the screen your robot formation's progress from the old location to the new one. The orientation of the formation can be discerned by a small red circle located on the front end of each robot. The red circle should always be located in the point, facing in the direction the robots are facing. Once your formation has reached its new location you can again click a new place on the screen and the formation will move accordingly. If at any time you wish your formation to return to its original location and starting information, click on **Home**, and the starting location will be resumed.



4.2 Advanced Movement

Advanced instructions may be given to the robot formation if the desired distance and angle of rotation are known. Below the grid, which displays the existing robots and their current formation, are fields labeled **Distance** and **Angle**. In the **Distance** field you can enter in the distance in which you want the formation to travel. The **Angle** field will take whole numbers ranging from 0 to 360. Whichever direction your robots are facing will be considered to be the (0,0) coordinate, so give your angle of desired rotation accordingly. Once you have filled in these two fields, click the **GO** button, which is to the right of the **Distance** and **Angle** fields.

Once you have hit **GO** you can watch the screen to see your formation move from the old location to the new location. To move the formation again, reenter the desired distance and angle to be turned before hitting **GO.** If at any time you wish to return to your original starting point hit **Home**, which is located on the right of your screen.

5. Trouble Shooting

Various problems may occur during operation of your robot formation. Some common problems and their quick fixes are listed below.

- **One of the robots experienced power failure.** In this case you have two options. Depending on where the robots are being used, it may be possible to go over to the robot and check the battery pack. If you can switch the battery or in any way power up the robot again, the formation can continue as it had before the power failure. If this does not work, your screen will ask you what you wish to do. You may simply tell the formation to continue on with n-1 robots in the same formation, or you can request a new formation. If after a certain amount of time you do not give the robots a command, a default time out will occur and the robot formation will continue on in the same formation with n-1 robots.

- **The robot formation doesn't seem to be receiving my commands.** In this case there may be a problem with the network cards used by the laptops mounted on the robots. Check to see that all the cards are installed and functioning correctly.

- **The robots are not reaching the specified location no matter how many attempts are made.** It is possible that the desired location is not reachable. In this case, choose another destination for the robots to relocate to, or try choosing a different robot formation. By doing so, the new formation may be able to maneuver around the obstacle that was unavoidable with the previous formation.

- **Using the click and move method, the formation does not go where I want it to.** To choose an exact destination, the completely accurate the advanced movement by specifying the exact distance and angle of rotation is necessary. If you do not know exactly where you want to go, then the click and move is still a better option.

- **After I have chosen the number of robots, I cannot find the desired formation.** It is possible that the formation you are searching for is not predefined for the number of robots you have chosen. You can either choose the number of robots that go with your desired formation or attempt to define your own formation.