

Distributed Intelligence in Autonomous Robotics

Assignment #4

Out: Tuesday, February 25, 2003

DUE: Thursday, March 6, 2003

Formation Changing

In this exercise, you will extend the multi-robot formation-keeping code (provided to you as example code in Assignment #1 – leader.c and follower.c) to enable robots to change formations back and forth between a line formation and a diamond formation. [You may also use your own version of formation-keeping code if you like, as long as it behaves similarly to the example code from Assignment #1.] You will use 4 robots in this exercise. The starting formation will be a line formation, exactly like the example code. The robots will start in a non-moving line formation until the “start” signal is received from the operator. The robots will then move in a straight line toward a distant (x,y) goal position (which will be specified). The human can then send a “change” signal to the robots, which triggers the robots to move into the diamond formation (defined precisely below). Once the robots are successfully transitioned to the diamond formation, they send a signal to inform the operator of this fact, and then continue toward the goal in the diamond formation, switching formations between the diamond and line formations as commanded whenever the human sends the “change” command. The robots will stop when either: (1) they reach their (x,y) goal position, or (2) the human sends the “stop” command.

For this exercise, we assume that the human will not issue a “change” command while robots are transitioning between formations. Your code should result in as smooth a formation change as you can generate, while still having the robots move toward the goal. So, it might be possible to change from the line to the diamond formation without the lead robot slowing down. On the other hand, you may want the lead robot to slow down when transitioning from the diamond to the line formation, in order to enable the other robots to catch up to their new positions. Exactly how you implement the transition is up to your design. Remember that your objective is to enable these changes with (a) smooth robot motions, (b) as little “lost time” as possible in moving the team toward the goal, and (c) as precise a formation as possible throughout the mission.

In this assignment, you will create 3 program files – one for the leader robot, one for the follower robots, and one for the simple user interface. YOU SHOULD NOT have a separate program for each follower robot; this is not good programming practice and does not scale well to larger numbers of robots. The example code from Assignment #1 shows how to use a single follower program for all follower robots.

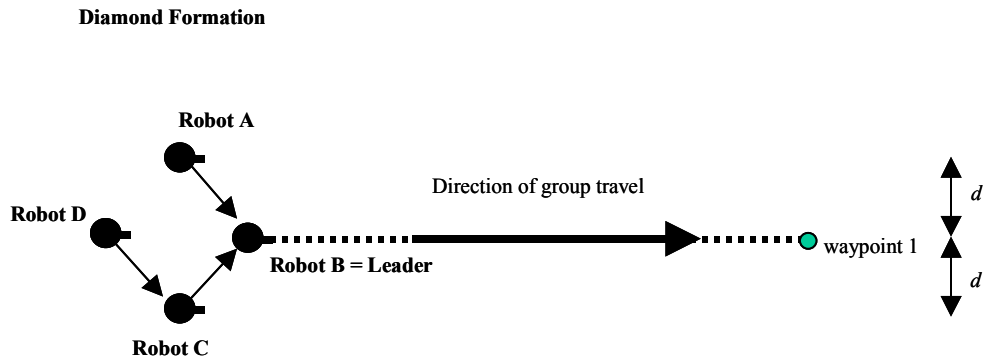
0. Review Multi-Robot Example Code

Re-read pages 10-11 of the handout that was provided to you in Assignment #1. This handout defines the line formation behavior that is implemented in the sample code provided to you. In this approach, there is one leader robot – Robot B – that knows the goal. The other robots follow, always maintaining a desired position from their reference neighbor robot. For the diamond position, the robots will maintain the same reference neighbor robot as defined in the line formation (i.e., A and C reference off of B, and D references off of C). In the line formation, the robots are a distance d from their neighbors. In the diamond formation, the robots are at a distance $\sqrt{2} \cdot d$ from their closest neighbors. (This is defined in more detail below.)

In this assignment, however, your robots will not have to make any turns; they will just have one distant goal that is perpendicular to the initial line formation, thus making the formation-keeping much simpler.

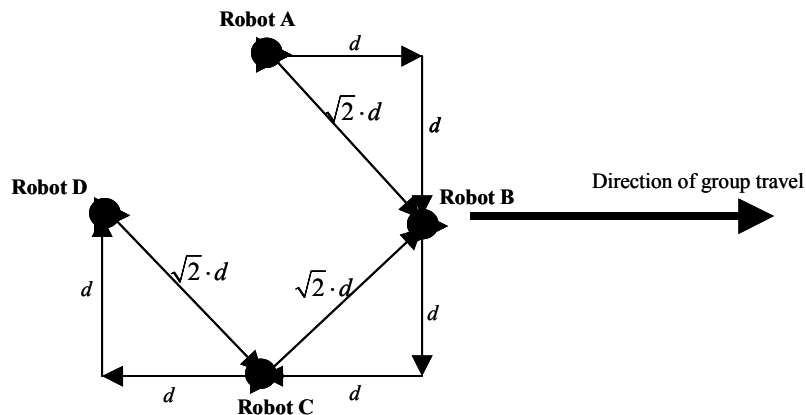
1. Definition of “diamond” formation

For this exercise, we define the diamond formation as follows.



The leader of the diamond formation is the same as the leader for the line formation – Robot B. As in the line formation, only the leader in the diamond formation knows the waypoint goal to be reached. In making transitions between formations, the robot’s labeled position in the line formation should match its labeled position in the diamond formation (i.e., Robot A in the line formation should move to the position of Robot A in the diamond formation, Robot C in the line formation should move to the position of Robot C in the diamond formation, etc.).

In this exercise, your code will be using the same form of neighbor-referenced formation that was used in the example code, in that each robot is assigned a position to maintain relative to a specific neighbor. In this exercise, the leader does not have a neighbor-referenced position to maintain. For the diamond formation, the reference position of each robot is as shown in the figure below.



These positions are explicitly given in the table below:

Robot	Desired position relative to referenced neighbor
A	$\sqrt{2} \cdot d$ distance behind Robot B, at a 45° angle to the left
C	$\sqrt{2} \cdot d$ distance behind Robot B, at a 45° angle to the right
D	$\sqrt{2} \cdot d$ distance behind Robot C, at a 45° angle to the left

2. Operator Interface

First, write a module (similar to the data collection module of Assignment #3, except this time there is no data collection) for the operator interface. This very simple interface should enable the operator to (1) start the robots at the beginning, (2) issue a “change” command to have robots toggle their formation, and (3) stop the robots. You should implement this interface using communications between the operator interface code and the robot code. The operator should be given simple prompts for the valid commands (e.g., “Enter S to start the robots, T to toggle the robot position, and X to stop the robots, then press return: ”. Or, whatever you like that accomplishes the objective). Be sure your code checks for invalid responses and doesn’t crash if the operator enters an invalid command.

3. Formation transitions

Write a subroutine called `formation_transition()` that enables the robots to change between their current formation (either line or diamond) to the other formation (either diamond or line). In this transition, you may want to have the robots communicate with the leader to ensure that the leader does not get too far ahead during the transition. For instance, when transitioning from the diamond to the line formation, you might want the leader to slow down, and then speed back up to the original pace once it receives messages from the other robots that they are now successfully transitioned into formation. Of course, you may not need to do this – it is just an option to consider in your design. Your objective is to minimize the amount of time that robots are in transition, while still generating smooth robot motions. Your robots should not spend an inordinate amount of time making the transitions.

Note that the existing code for formation-keeping (in `follower.c`) causes a robot to stop if its desired goal position is behind it. It is recommended to keep this behavior in this exercise. Thus, when Robot D moves from its current line position to its target position in the diamond formation, it could stop until the other robots pass by, and then move to its desired position in the diamond formation.

Once your robots have successfully reached the diamond formation, they should communicate this to the operator, with a message printed to the screen, so that the operator knows when it is OK to issue another “change” command.

4. Keep formation

Revise the “`figure_goal()`” subroutine in `follow.c` so that the calculation of the desired robot position depends on whether the team is in a line formation or a diamond formation.

5. Complete program

Put together all the pieces above to enable your system to operate as outlined in the discussion above. Be sure to use the script provided for Assignment #3 or a forking procedure so that all of your robot processes can be started from one xterm window. Run your code for 4 robots, ensuring that they are able to successfully change formations back and forth from line to diamond and back while moving toward the goal. **The single waypoint goal position for this Assignment is $(x,y) = (20000, 0)$.**

6. Document and clean up code

Write documentation at the top of your code (for each file that you create that includes a “main” function) that includes the following:

- Your name
- File name of your code
- “CS594 Assignment #4”

- A short description of what the code does
- A description of the compile command needed to compile this file
- A description of the command needed to execute your code (which should make use of the script from Assignment #3 if your code does not make use of process forking)

Document each function with a short description of what the function does.

Before you submit your code, please change it so that it uses Nserver port # 7019.

Remove (or comment out) all debugging print statements from your code that are not meaningful for running your code.

TURN IN THE FOLLOWING:

For this assignment, please turn in the following:

- a) A screen dump of your robots at the beginning of the task (i.e., in line formation prior to the operator starting the team in motion).
- b) A screen dump of your robots approximately half-way through a transition between line and diamond formation (i.e., after the operator has issued a “change” command).
- c) A screen dump of your robots successfully moving in the diamond formation.
- d) A screen dump of your robots approximately half-way through a transition between diamond and line formation (i.e., after the operator has issued another “change” command).
- e) A screen dump of you robots after they have successfully transitioned back to the line formation.
- f) A discussion of any issues you had to deal with in order to successfully generate the robot control code.
- g) A hardcopy of your code, fully documented according to the instructions given above.
- h) Email a tar or zip file of your code, named *yourlastname-4.tar* or *yourlastname-4.zip* to parker@cs.utk.edu. This tar/zip file should contain the following:
 - All files needed to compile your code from an empty directory.
 - A makefile that compiles all your code with a “make all” command.
 - A script that enables the user to execute your code from one xterm window.

(Be sure to confirm the correctness of your tar or zip package by compiling and executing your code from the tar/zip file from scratch in an empty test directory. As before, points will be deducted if your compilation or execution instructions are missing, incomplete, unclear, or incorrect, or if your code has compiler errors, or if files are missing.)