# Final Project (for Graduate Students):
*Multi-Robot Predator-Prey*

---

**Assigned: Tuesday, October 28, 2008**
**Due: Tuesday, November 25, 2008, 12:00:00**
**In-class Competition (in the Hydra lab): Tuesday, November 25, 2008, 17:05:00**

---

In this final project assignment, you will develop multi-robot predator-prey software, and your software will be put in a head-to-head contest with other student's software (in the Hydra lab). (However, nearly all of your grade will be based on an evaluation of your code on its own, not how well it performs against other students' code). You will have 2 predator robots and 1 prey robot. The objective for the prey robot is to avoid "capture" as long as possible. The objective for the two predator robots is to "capture" the prey. Here, we define "capture" as meaning that both predator robots are simultaneously within 2 meters of the prey. The prey is successfully escaping as long as it avoids having both predator robots within this range. In this assignment, we assume that the environment is known to the robots, and to you, the designer. Thus, you are allowed to make use of the known environment, by either giving the robot the ability to plan paths in the environment, or even by hardcoding predator search strategies (i.e., to visit waypoints that you pre-specify.) However, your predator robot behavior cannot know anything about how your prey robot behavior works, and vice versa.

In this assignment, you will write code for 2 separate kinds of robots – (1) predator robots, and (2) a prey robot. These will be separately-compiled programs that will run as separate processes. Much of this assignment will make use of code you have written for previous assignments. The new aspects for this assignment are: combining multiple behaviors, multi-robot communication, re-invoking your path planner for multiple path plans, and hunt and evasion behaviors.

As previously noted, your software will be graded as if it were a stand-alone assignment, meaning that you will turn in results like always, this time showing how your own predator robots work to catch your own prey robot. In addition, you will also use your software for head-to-head robot competitions with your classmates in class on Tuesday, November 25th. In these competitions, you will pit your predators against a classmate's prey, and vice versa. If your robots do well in these competitions, then you'll get extra credit over and above your regular grade on this assignment (as well as bragging rights!). [If your robots do not do well in this competition, your grade will not be penalized just because your robot lost in the competition.] More details of how we'll conduct this competition will come later. Because we will be interchanging predator and prey code from different people for these competitions, it is critical that you follow the instructions below exactly, to ensure that the software is interchangeable.

Note that this is not a BattleBot competition, where you dream up subversive ideas for how to crush your enemy. It is expected that you will abide by the spirit of this competition, where predators and prey square off head-to-head in a "fair" match. You are not allowed to make use of any dirty tricks that are against the spirit of a true predator-prey competition.

## .cfg and .world files, plus Predator and Prey robot configurations

To ensure uniformity, I am providing you with the .cfg file and the .world file to use for this assignment; these files are called, creatively, FinalProj.cfg and FinalProj.world. They are available from the class website, here: http://www.cs.utk.edu/~parker/Courses/CS594-fall08/Labs.html. (Look under the Final Project section.) You must use these files for this assignment. Something that can vary in this configuration, however, are the starting positions of the robots. You will want to test your approach with different robot starting positions. For the competition, we will start the predator robots somewhere in the region bounded by (-10.5, 5), (-10.5, -2), (-7, -2), (-7, 5). We will start the prey robots somewhere in the region bounded by (-4.5, 5), (-4.5, -2), (-1, -2), (-1, 5). The starting orientations will be random. Be sure your code works with any starting positions and orientations in these regions.

*Sensors:*
In this assignment, your predators and prey will have fiducials on them that make them distinct from one another. Your predator and prey robots are set up in the .world file to have the following fiducial return values:

Predator robot fiducial_return ← 1
Prey robot fiducial_return ← 2

These fiducial returns allow your software to make distinctions between predator robots and the prey robot. Your predator and prey robots will also have a laser scanner, as usual.

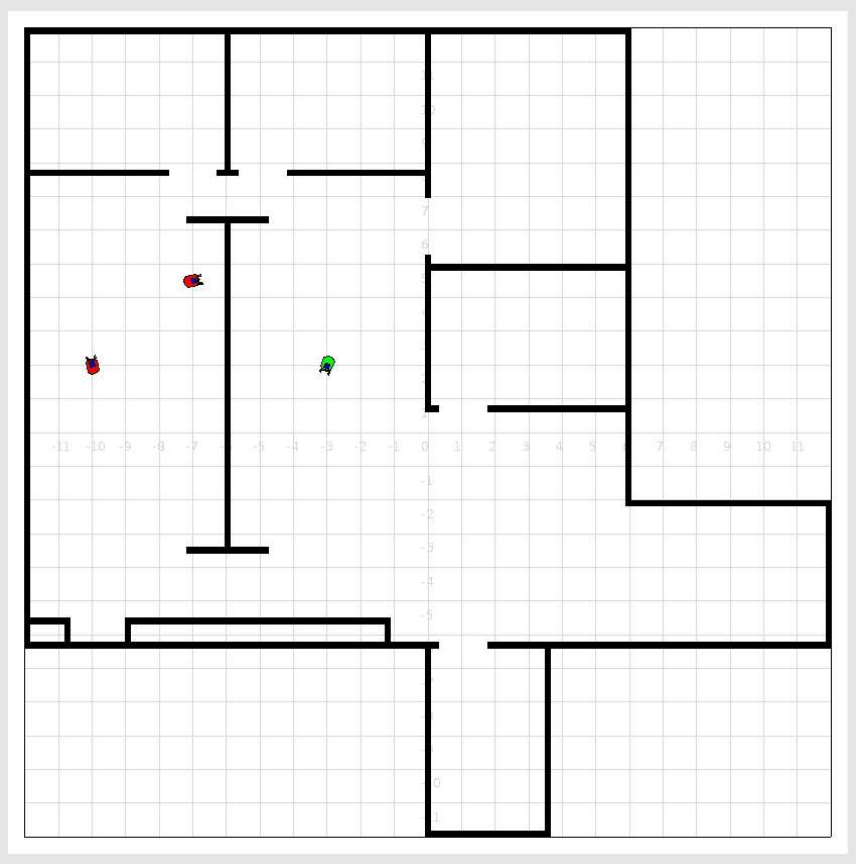
*Max Speeds:*
In this exercise, we are going to allow the prey to move a little bit faster than the predators. To ensure uniformity among all robots, you must abide by the following maximum speed constraints:

Predator robot maximum speed ← 0.4
Prey robot maximum speed ← 0.5

This means, obviously, that the prey can outrun the predator. But, since we'll be operating in a closed environment, it is possible for the two predator robots to try to trap the prey robot if you properly coordinate the predator robots. Also, the prey will be disadvantaged in that it can only see the predator if the predator is in front of it (i.e., in the direction the laser is facing). It can't see in all directions, because its laser only has a $180^o$ field of view.

## The Environment

For this exercise, we will be using the "autolab" bitmap. Below is what the environment looks like, along with possible starting positions of the robots. Here, the two predator robots are in the left room; the prey robot is in the middle room. (In the simulation, the predator robots are red, while the prey robot is green.)



## Predator Robot Design

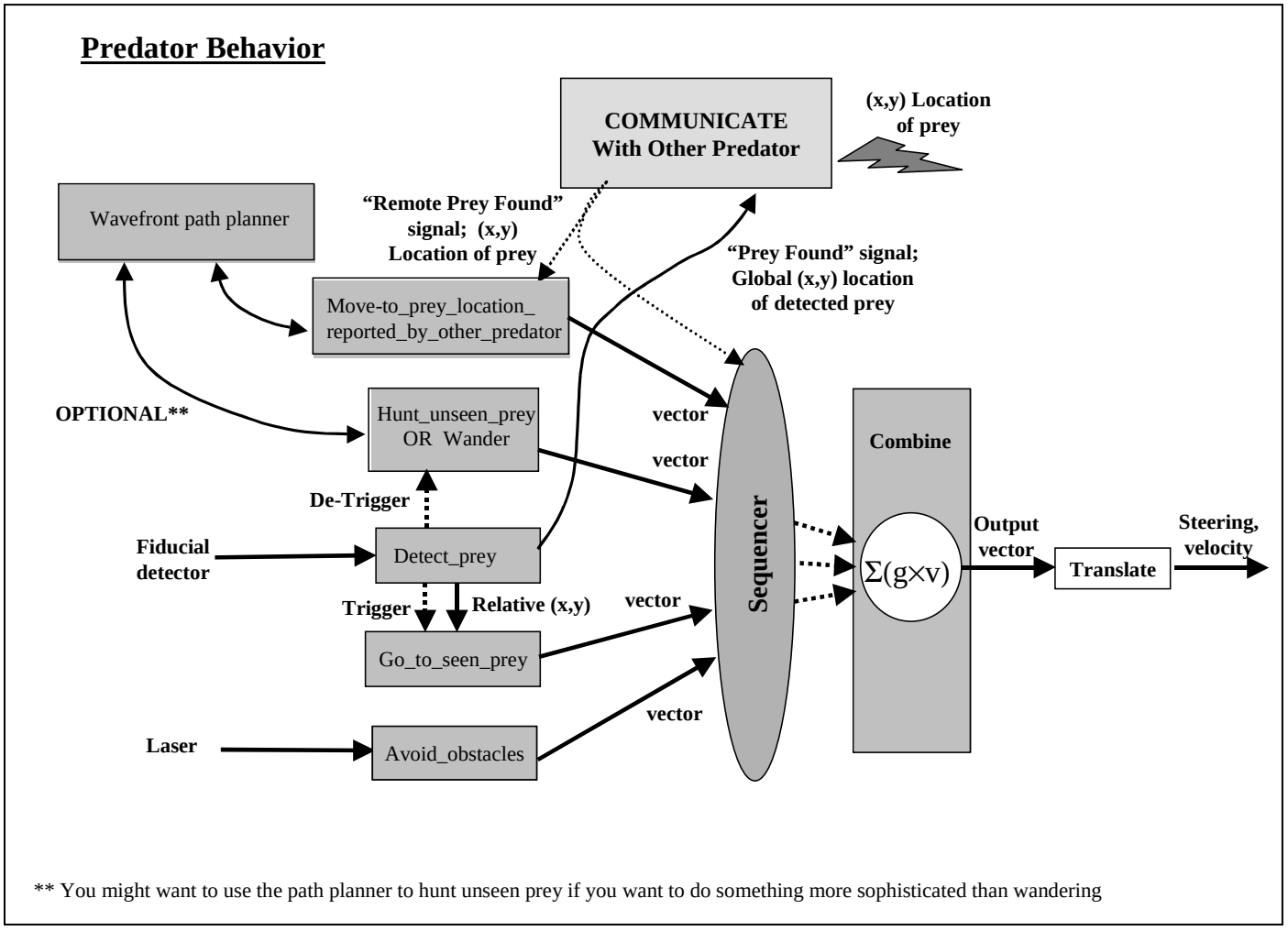
As a suggestion, the overall behavior design of a predator robot is shown in the figure below (you don't have to implement this design exactly, although you may find it useful). The predator robot will combine aspects of your HW #2 robot, which sensed fiducials, with parts of your HW #3 (Part 2), which moved to a waypoint position (although you are free to change this previous code as you like). There are some differences, though:

- First, the "beacon" (which in this assignment is on a prey robot) is moving. So, your predator robot has to be able to navigate toward a moving prey, using the behavior "Go_to_seen_prey".

- Second, you need to incorporate a behavior ("Hunt_unseen_prey") for seeking out the prey. This could just be a random wander algorithm. Or, maybe this behavior would make use of your wavefront path planner from HW #4. Or, you may hardcode a series of waypoints to visit, if you want to give the predator robot a fixed path to take for searching. This will likely result in finding the prey faster than through a simple wander behavior. However, you are not required to make a "smart" behavior for hunting unseen prey. You may just have the robot randomly wander. In either case, the "Detect_prey" triggering function will inform this behavior when the prey is found

(i.e., based on fiducial detection), which will act to de-trigger the hunt, and instead just make use of the "Go_to_seen_prey" behavior to head toward the seen prey.

- Third, you are required to have your two predator robots communicate with each other when one of them finds the prey. The predator robot that finds the prey reports the prey's position to the other predator periodically, as long as that prey is being detected. The predator robot that receives this report must then move methodically toward the reported position of the prey, using the behavior "Move_to_prey_location_reported_by_other_predator" (which you may definitely rename to something shorter; I'm just using this here so that it is descriptive). This behavior will make use of your path planner from HW #4 to move methodically toward the current prey position. Since the prey is still probably moving, the location being reported will also change, meaning that you'll have to occasionally re-plan paths to different prey locations.




** You might want to use the path planner to hunt unseen prey if you want to do something more sophisticated than wandering

For this assignment, we will just the automated referee tool, referee.cc (on the course website) to determine when the predators have capture the prey, along with the elapsed time.

NOTE: Your two predator robots MUST RUN THE EXACT SAME CODE! If you want the two predator robots to perform different activities, then you must have them communicate with each other to coordinate who does what.

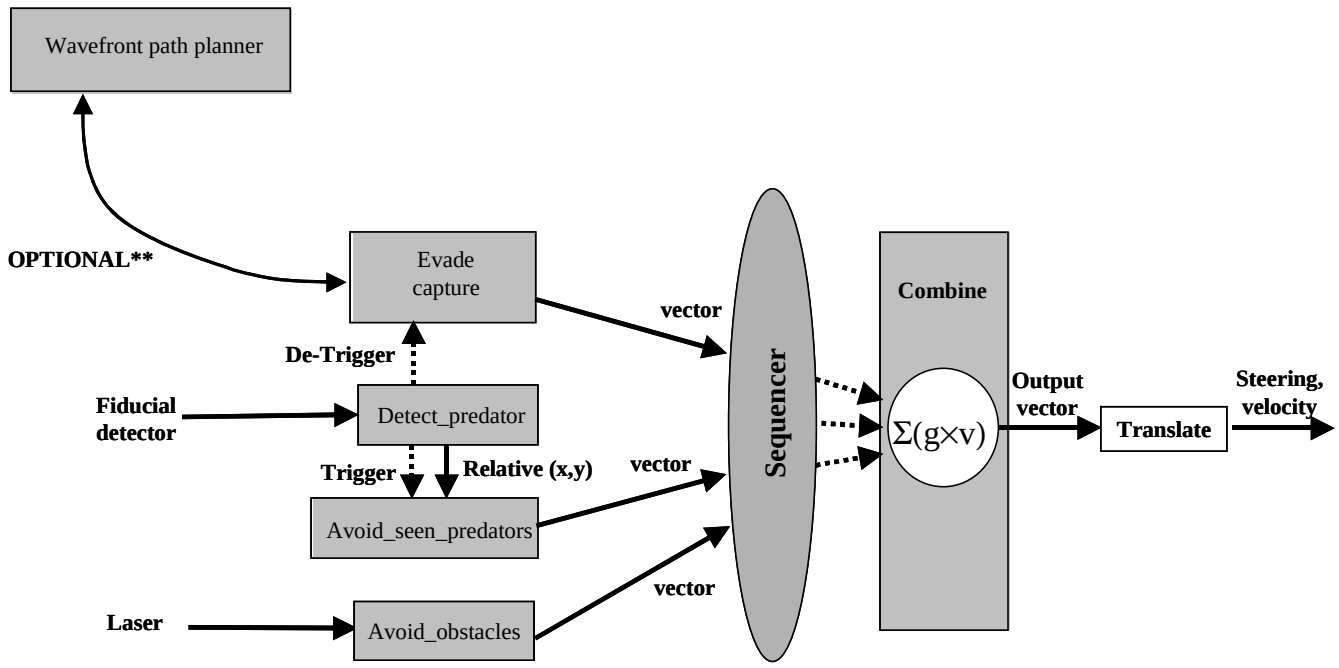
## Inter-robot communication

In the appendix, we give instructions on how to have robots communicate with each other in these simulations. The idea is simply to use sockets to communicate between processes. We're giving you code for this communication (on the course webpage), so that you don't have to spend time hacking sockets. If you don't want to use this code, you don't have to. It's your option.

## Prey Robot Design

At a minimum, your prey robot must avoid obstacles and move away from the sensed predators. Note that the prey robot has a problem, in that if it turns to run from predators, it can no longer see the predators, because the laser that detects the predator fiducial is pointing forward. You'll have to come up with a behavioral strategy for dealing with this. Perhaps you'll implement something like rabbit behavior: when the predator is first seen, you run like heck for a while away from the predator, then stop and look behind you to see if you have escaped. Keep in mind that there are 2 predators, so your prey robot will want to take both into account in deciding which way to escape (think vector summation). You may also give your prey robot strategies for avoiding capture as long as possible, such as trying to find a good pace to hide. You may use your own knowledge of the environment to design specific strategies. However, as already stated, you may not use any knowledge of the predator's strategy in designing the prey's strategy (or vice versa). Your prey robot will not communicate with any other robot. The figure below gives the general outline of the prey robot behavior construction. (As with the predator behavior construction, this design is given to you as a suggestion that you should find helpful; however, you are not required to implement this design exactly.)

**Prey Behavior**




** You might want to use the path planner to plan paths for evading capture, although it isn't required

## Running your code with multiple robots

(You've already seen most of the information in this section in HW #2, but I'll repeat it here for completeness.) When you run your experiments, each of the 3 robots will be running a separate process. Both predators will run copies of the same program (called "yourlastname-predator-FinalProj"); the prey robot will run its own program (called "yourlastname-prey-FinalProj"). To control multiple robots in the same simulation, do the following. Open up 3 separate windows, one for each robot. Connect each window to the directory where you have your compiled robot control codes. Start up Player/Stage as always (i.e., "robot-player FinalProj.cfg"). Then, enter the following commands, each in its own separate window:

- For controlling predator robot #1:
  linux> ./yourlastname-predator-FinalProj –p 6666

- For controlling predator robot #2:
  linux> ./yourlastname-predator-FinalProj –p 6667

- For controlling prey robot:
  linux> ./yourlastname-prey-FinalProj –p 6668

Note that the "-p" option specifies the port number being used by that robot. These port numbers are defined in the FinalProj.cfg file. These commands will then connect to each of the 3 separate robots, and your 3 robots will execute their respective control codes.

**Example binaries for predator and prey**

To help you test and debug your own code, example binaries of the predator and prey are available on the course website.

**Design notes**

This project is open for you to achieve the robot behaviors as you like. As discussed, the software design ideas given in the figures above are suggestions, not requirements. Some specific design points are:

- You are free to use the vfh and wavefront drivers that are provided for you in Player/Stage for the purposes of obstacle avoidance, going to goals, and path planning. You do not have to write your own. Just be sure to abide by the maximum velocity requirements.
- No noise needs to be added to any of the sensors.
- You are not allowed to change the hardware configuration.
- Crashing into another robot constitutes a dirty trick.
- Essentially, you are free to use whatever software techniques you like to achieve the predator and prey behaviors that are specified, as long as the code is yours, or is provided to you through Player/Stage. If you want to use any other publicly available software, ask Dr. Parker for permission first.

**Automated referee for determining capture: referee.cc**
**and**
**Inputting the autolab map into your code**

Thanks to two of your enthusiastic classmates (Bobby Coop and Richard Edwards), we now have an automated referee tool for determining when capture is made. This tool is called referee.cc, and is available on the homework website for the course, along with some supporting files (args.h, inputMap_v2, and a makefile called MakeReferee). You'll also need to download the map file "autolab.pnm", which is converted for you to a p6 format of pnm, which is now readable by the updated utility inputMap_v2.cc. To compile this code, simply enter "make –f MakeReferee".

The referee tool will automatically set the positions of the two predators and prey within the pre-defined starting boundaries outlined in this write-up. To use this tool, simply run it in a separate window on the same machine as the predators and prey. (You'll also need to be sure that the "autolab.pnm" file we provide you (i.e,. pnm p6 format) is in the same directory as your referee executable. The tool will prompt you to start the simulation (by pressing enter), at which point the robots will be randomly positioned. Then, you should start your robot's predator and prey code; whenever the predators are both within the specified distance of the prey, the referee will

declare capture, along with the elapsed time. Use this tool while you are developing your code, so that you can take advantage of the random starting positions.

This referee tool needs to access the map in order to ensure that the predators and prey are on the same side of the wall as the others (i.e., it doesn't count to capture the prey if it is on the other side of the wall!). So, the referee.cc code makes use of the inputMap_v2 utility, which reads in the pnm p6 file "autolab.pnm" (which is provided for you). You can also make use of this updated inputMap_v2 utility (with the p6 version of autolab.pnm) to input the map into your own robot control code.

(If you are curious, you can find more reading on pnm files here: https://people.scs.fsu.edu/~burkardt/data/pnm/pnm.html.)

***How we'll grade your code***

Because you are writing both the predator and the prey code, it may be hard to judge how good your code is. For example, if your predators are always able to catch the prey quickly, then does this mean you have really good predator code, or does it instead mean that you have really lousy prey code? It's hard to say. This is why we'll have the in-class competition (in the Hydra lab) to pit your predator robots against someone else's prey robots (and vice versa). Over a series of head-to-head competitions, we'll be able to see what the best strategies are for predators and prey. In the competition, the predator robots will get more points the more quickly they capture the prey, while the prey robot will get more points the longer it avoids capture. Specifically, the score of both the predator and the prey will be the number of elapsed seconds until the prey is captured, within a maximum time period (such as 4 minutes; the specific time TBA). However, in the case of the predator, lower scores are better, and in the case of the prey, higher scores are better. We will rank order the winning predators and preys to decide winners in each category. *You are required to participate in the competition (-10 points on your final project grade if you don't participate).* However, otherwise, this competition will be for extra credit points for you (and bragging rights!). Details on the competition will be announced later.

For your individual code grading, we'll look to make sure that your predators and prey are designed as outlined in this assignment. We want to see the ability for the prey to escape for some period of time, but for the predators to eventually be able to capture the prey. We want to see that you've integrated your path planner with your predator hunting behavior, and that you have implemented and use inter-robot communication between the predator robots.

***WRITE UP THE FOLLOWING (written up in a single pdf file called yourlastname-FinalProj.pdf):***

**a)** A brief discussion of your predator behavior strategy. (Just point out the aspects of the behavior where you had a choice; don't re-iterate the required parts of the design.)

**b)** A brief discussion of your prey behavior strategy. (Just point out the aspects of the behavior where you had a choice; don't re-iterate the required parts of the design.)

**c)** 1 screenshot of your predator and prey robots moving from their starting positions to the final position where the prey is captured. Be sure your screenshot includes the robot traces. The starting robot positions for this screenshot must be as follows (these are the same as in the FinalProj.cfg file provided to you):

- Predator robot #1: starts at (-7, 4.5, 10)
- Predator robot #2: starts at (-10, -2, 100)
- Prey robot: starts at (-2, 5, 250).

NOTE once again: Your predator code may in no way make use of the knowledge of the prey's starting location, or vice versa.

***SUBMITTING YOUR HOMEWORK:***

Place all your files in a single directory. These files should include:

- Your pdf file as described above, called "yourlastname-FinalProj.pdf"
- Your makefile, called "makefile" or "Makefile"
- Your predator robot control code, called "yourlastname-predator-FinalProj.cc".
- Your prey robot control code, called "yourlastname-prey-FinalProj.cc".
- Any additional include files or other code you created (called whatever you want them to be called).

Remove all other unnecessary files. Use the submit script **594mr_submit** to submit your files. (These will be emailed to Dr. Parker.)

# Appendix: Communication Between Robots

*Note: You are not required to use these utilities. Or, if you want to change them, you may. They are just provided for your convenience, in case you want to use them.*

---

On the course website are two files provided for you: communicate.h and commsExample.cc:

http://www.cs.utk.edu/~parker/Courses/CS594-fall08/Homeworks/communicate.h

http://www.cs.utk.edu/~parker/Courses/CS594-fall08/Homeworks/commsExample.cc

These files give you the basic routines needed to communicate between multiple processes, such as robots in Player/Stage. These routines are set up to send UDP datagrams between two robot processes. (The reason why this uses UDP rather than TCP is that, in general, UDP offers better performance than TCP on real robots, which can break. We can have problems with hung processes on real robots if, for example, one robot fails during the application. We won't go into the details here.)

The commsExample.cc file shows you how to use these communications messages. The gist is this: You enter (as command line parameters) the ID and PORT number of the current robot's process, as well as the ID and PORT number of the 2nd ("friend") robot's process. Then, the code sets up the socket for communication. When you want the current robot to send a message to the other robot, your code must format that message (using the "send_cmd" process). When you want the current robot to receive a message from the other robot, your code must use the "recv_cmd" process to read and decode the message.

**About message formats**

The provided code *does not* provide the complete message format that you'll need for this homework. Instead, it shows you how you can create messages according to your own format. The basic message format is as follows:

```
TS$R!
```

where:

```
T:   1 character message type (currently 'F' means target found, and 'C'
     means application is complete (stop the program))
S:   ID of sender (converted to characters)
$:   special delimiter symbol separating fields
R:   ID of recipient (converted to characters)
!:   special delimiter symbol denoting end of message
```

So, for example, the actual message sent/received might be:

```
F1$2!
```

However, for this assignment, you'll need to beef up this message format to add in the x,y position of the target position. So, ultimately, your sent/received messages would look something like:

```
TS$R$xxx.xx$yy.y!
```

A specific message might be:

```
F1$2$10.5$-1.4!
```

where `xxx.xx` is the `x` position of the target and `yy.y` is the `y` position of the target. It is your job to add this additional information to your messages. You may want to make use of functions such as `atof` for this purpose. You'll need to make these changes in 2 places: in send_cmd to format the message to be sent, and in recv_cmd to decode the received message. You can add in any message types as you like and find useful.

Note that in processing the messages, the code is set up to only accept messages from the "friend" robot, and only if the message has the current robot's ID in the recipient field. This is a good safeguard, to ensure that the robot only responds to valid messages intended for itself.

**Command line parameters**

To run this code, you'll need to enter in the current robot's ID and port number, and the "friend" robot's ID and port number as command line arguments. So, for one process, you'll have something like:

```
linux> ./commsExample 1 6000 2 6001
```

and for the second process, you'll have something like:

```
linux> ./commsExample 2 6001 1 6000
```

These command lines will allow these two processes to send and receive messages from each other.