

Homework 5: *Multi-Robot Predator-Prey*

Assigned: Thursday, March 29, 2007
Due: Thursday, April 19, 2007 at 06:00:00

In this assignment, you will develop multi-robot predator-prey software. 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 1.5 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: multiple robots, inter-robot communication, re-invoking your path planner for multiple path plans, and hunt and evasion behaviors.

Your software will be graded as usual, 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 Thursday, April 19. 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, HW5.cfg and HW5.world. They are available from the class website, here: <http://www.cs.utk.edu/~parker/Courses/CS594-spring07/Labs.html>. (Look under the HW#5 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:

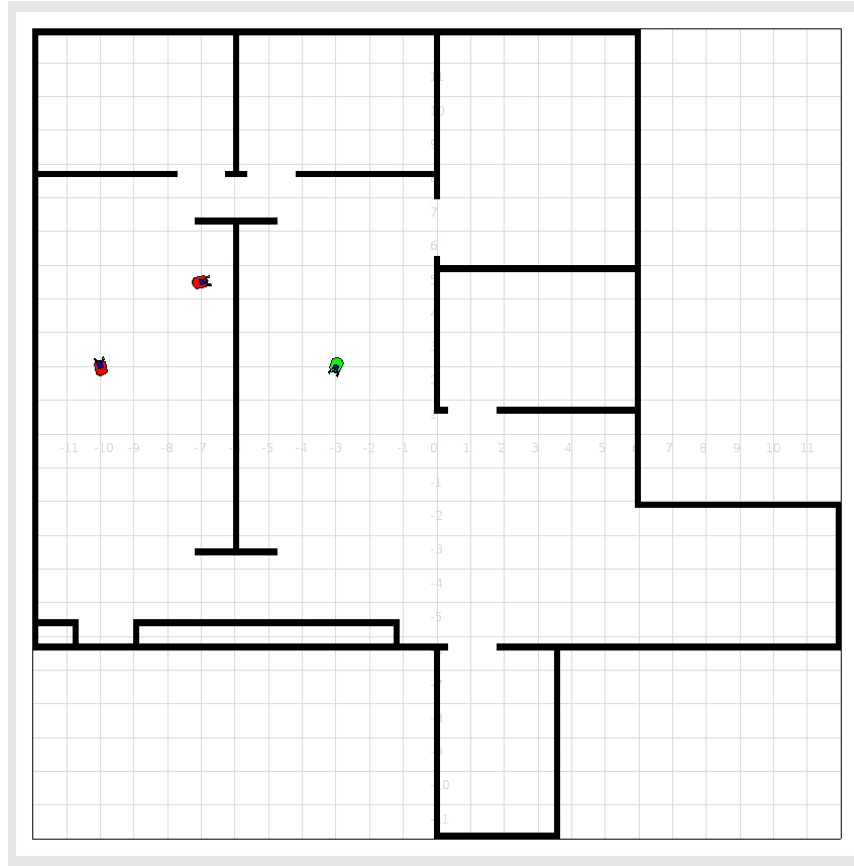
In this exercise, we are going to allow the prey to move a little bit faster than the predators. To ensure uniformity among all robots, you must abide by the following maximum speed constraints:

```
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° 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.)

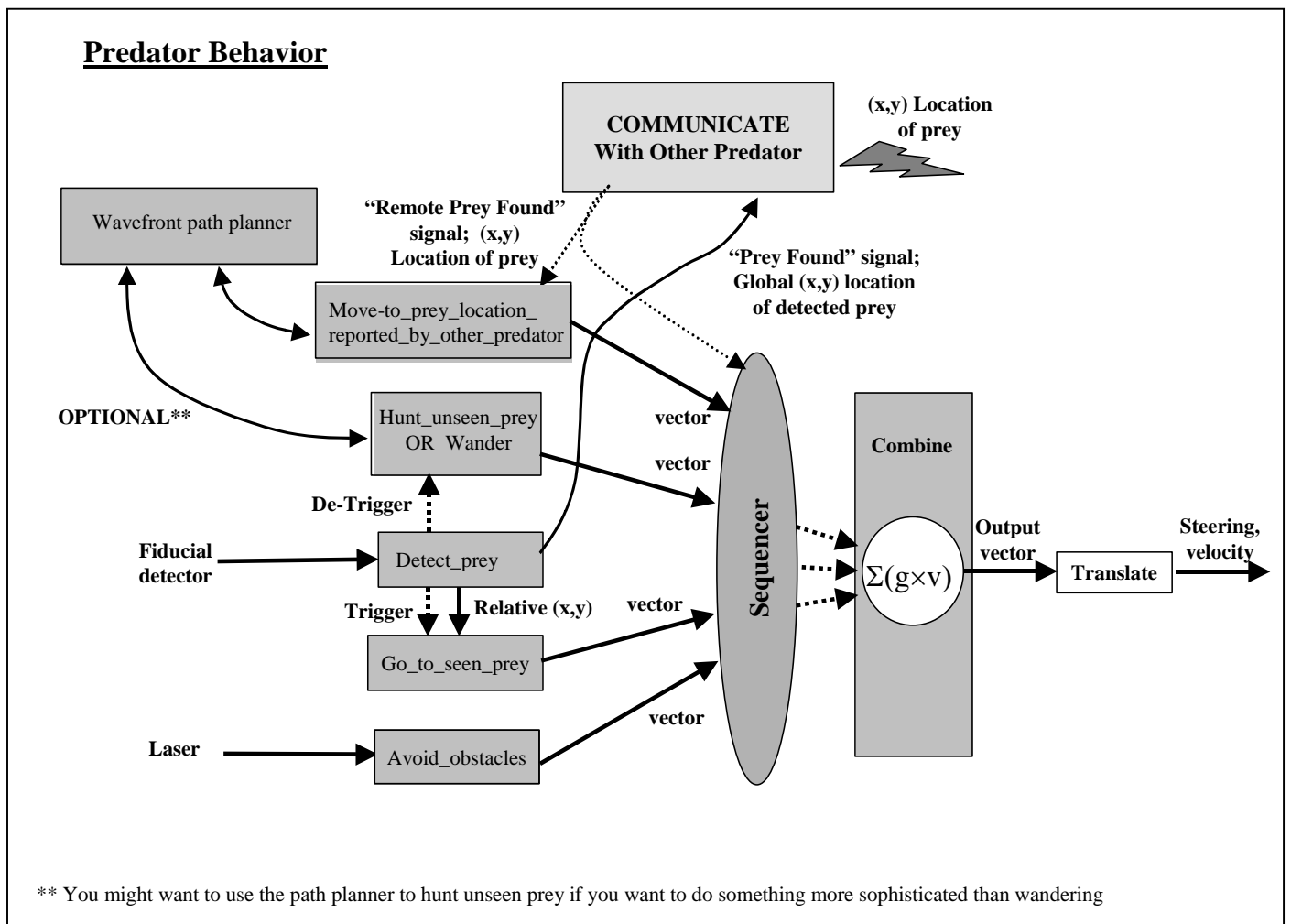


Predator Robot Design.

The overall behavior design of a predator robot is shown in the figure below. The predator robot will act a lot like your HW #3 robot that sought out beacons and moved toward them. 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 may want to incorporate a smarter behavior (“Hunt_unseen_prey”) for seeking out the prey, rather than just using a wander behavior. 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 smarter behavior for hunting unseen prey. You may just use wander. In either case, the “Detect_preay” 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 they 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_preferred_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.



For this assignment, we will just manually determine (through visual inspection) when both predator robots are within the required range of the prey (i.e., 1.5m), and declare the prey captured. However, if you like, you may also automate this, by having the two predators talk

with each other and confirm when both are simultaneously within 1.5 meters of the prey. Again, this is optional.

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.

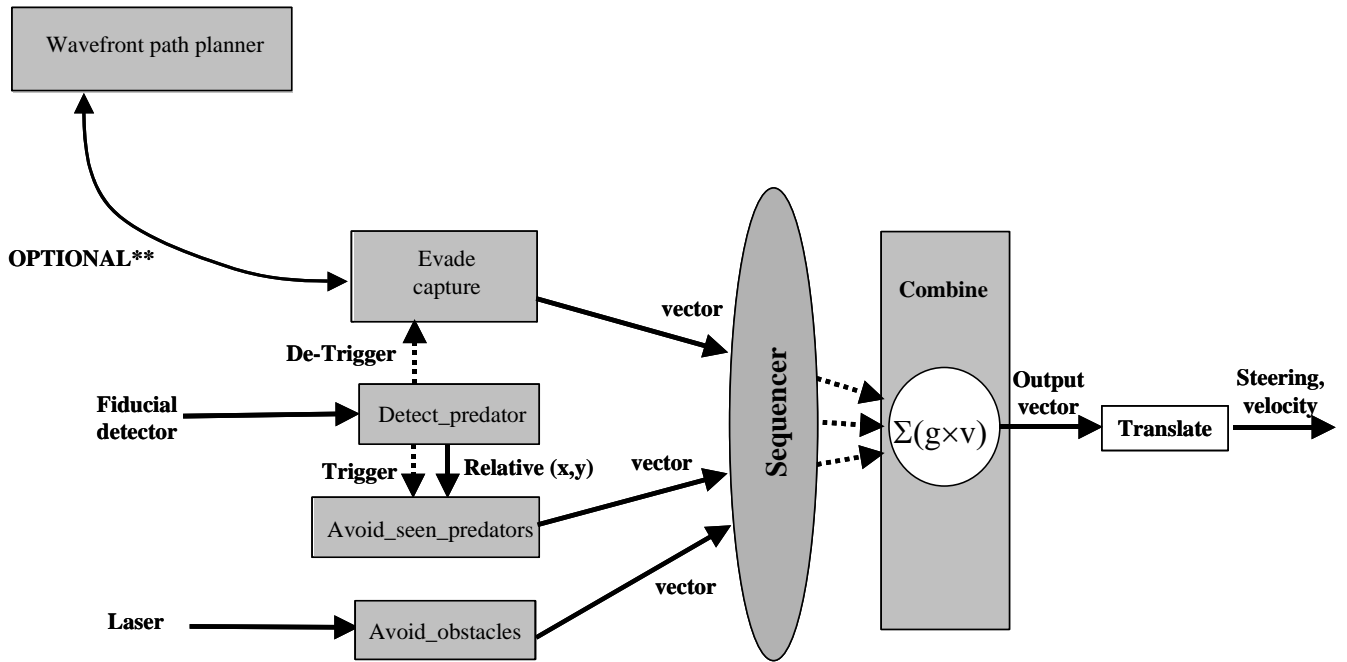
Inter-robot communication.

In a separate writeup, 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'll give you code for this communication, so that you don't have to spend time hacking sockets.

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 place 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.

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.

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-HW5"); the prey robot will run its own program (called "yourlastname-prey-HW5"). 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., "player HW5.cfg"). Then, enter the following commands, each in its own separate window:

- For controlling predator robot #1:
linux> ./yourlastname-predator-HW5 -p 6666
- For controlling predator robot #2:
linux> ./yourlastname-predator-HW5 -p 6667
- For controlling prey robot:
linux> ./yourlastname-prey-HW5 -p 6668

Note that the “-p” option specifies the port number being used by that robot. These port numbers are defined in the HW5.cfg file. These commands will then connect to each of the 3 separate robots, and your 3 robots will execute their respective control codes.

How we'll grade your code

Because you are writing both the predator and the prey code, then 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 instead does it mean that you have really lousy prey code? It's hard to say. This is why we'll have the in-class competition 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. But, this competition will be for extra credit points for you (and bragging rights!).

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. You should make use of the usual motor schema vector combination approach. The use of the “Act”, “Pilot” and “Navigate” functions are optional; use them if it is helpful, but it isn't necessary.

WRITE UP THE FOLLOWING (written up in a single pdf file called yourlastname-HW5.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 HW5.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-HW5.pdf”
- Your makefile, called “makefile” or “Makefile”
- Your predator robot control code, called “yourlastname-predator-HW5.cc”.
- Your prey robot control code, called “yourlastname-prey-HW5.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 **594sir_submit** to submit your files. (These will be emailed to Dr. Parker.)

[When I unpack your files, I should be able to say “make yourlastname-HW5”, and then “player HW5.cfg”, and then “./yourlastname” to run your code. Be sure you have everything set up properly so that this works as expected.]