CS 480 Task 4: BDI Blimp Navigation

Description

This task focuses on the BDI (Belief, Desire, Intention) model for intelligent agents. It is basically a reverse version of what will be Task 5: instead of the bat sending out signals to estimate where an insect is, it would be the insect using the signals to estimate its own position from known signal sources. Unlike in the previous tasks, the theme here is not nature-based. Instead, we are modeling a simplification of an existing (obsolescent) navigational system, LORAN (Long Range Navigation), and the agent is a blimp. In simplistic terms, LORAN functions by propagating at least two (usually more) uniquely identifiable waves outward from fixed stations with known positions and times at known speeds. A receiver can determine how far each wave has traveled when it arrives. The intersection and subsequent trilateration from two waves provide a reasonable estimate of position. More waves improve the results, but we will use only two.

The BDI aspects are as follows:

- The belief of the blimp is where it thinks it is, as opposed to where it actually is.
- Its desire is to travel in a known square pattern back to its starting position.
- Its intention is to correct for errors as it travels in the square pattern.

Requirements

You must satisfy the following requirements in a single Java program. Be sure to comment your code so it is clear where each part is handled.

Beware: The jar is called task 4, but the packages inside it are called task 5.

Section 1: Modeling and Simulation

Part 1: LORAN

There are two transmitting stations (with identifiers `station1` and `station2`) positioned in our standard world at (0,-500) and (-500,500), respectively. Randomized positions would be more interesting, but many configurations would not be usable because the propagated waves must intersect within the boundaries of the world.

Both stations propagate their waves as described in lecture. The number of rays is your choice, which remains fixed for all experiments. Each ray propagates at five-meter intervals., Do not record these steps as events in the final track, but you may use the `ray1` and `ray2` objects for debugging. Assume the bounding circle of each ray is five meters in diameter. Induce a small amount of error in the propagation to model interference from mysterious real-world phenomena. This uniformly distributed random error ranges inclusively from -0.6 to +0.7 meter per interval, which is summed along the whole path. (You may have to play with this range if it is too good or bad.) As a result, the wavefront (i.e., the perimeter defined by connecting the end point of each ray) will not be perfectly circular. There is no need to propagate beyond the blimp or the bounds of the world.

How you differentiate and propagate the rays is your choice. You may cheat in your code to pass data internally for convenience, provided that your solution faithfully models the specified limitations.

Part 2: Blimp

The blimp needs to determine when its five-meter bounding circle has been intersected by a ray from each station to estimate its position. There is error inherent in this process due to the bounding circles, hop intervals, and spread of the rays.

The goal of the blimp is to move from one position to another repeatedly, but presume its mechanism for doing so is rudimentary (typically called "dead reckoning"). In other words, moving the blimp is not a simple function of incrementing or decrementing its actual x and y coordinates. Instead, the blimp relies entirely on the navigation signals to estimate where it is (its belief). From this, it needs to determine where it needs to go (its intention) as the next step in fulfilling its mission to trace a square pattern (its desire). Move the blimp forward by five meters per step and orient its yaw with its course.

To model the belief, the blimp has both an actual position (where it really is) and a believed position (where it believes it is based on the navigation signals). You need to log these with the respective identifiers `blimp-actual` and `blimp-believed`, which the viewer will render as two blimps (the actual solid, the believed translucent). Also log the intersection points of the rays with identifiers `ray1` and `ray2`.

The square pattern the blimp needs to follow is the connection of the corners (-200,-200), (200,-200), (200,200), (-200,200), and back to (-200,-200) in five-meter increments.

Here is an example scenario. The blimp is initially at actual position (-200,-200), but it does not know this. It receives the two LORAN signals and estimates its position at believed position (-190,-190). It wants to go to actual position (-190,-200) (because it does know where the lines are). To do so, the appropriate action from its perspective would appear to be to decrement its actual *y* by 10 to correct for the perceived error from the previous move. However, this correction actually puts the blimp at actual position (-200,-210). Repeat this process until the blimp arguably makes a complete circuit of the square or goes completely out of control, then stop the simulation manually.

The blimp has no predefined step distance; the step is determined by the error to correct. Undergraduates do not need intermediate movement from the current point to the next, but grad students do.

Section 2: Summary of Approach

There are relatively few well-defined solutions to AI problems, which means that your implementation can be anything you want. I will evaluate it primarily on its performance and your explanation of how it works.

For each part in Section 1, describe at minimum the following:

- what you did
- how well it worked
- what its pros and cons are
- what you might improve

This summary does not need to be extensive. A few sentences for each bullet should be adequate. Be honest: your summary must correspond to the actual performance of your model.

Be sure to address each of these bullets for each part.

Submission

Submit your source file(s), a text readme file that explains how to compile and run your solution, and the summary document in PDF format through the link on the course web page.

For Part 2 in Section 1, submit three representative track files of your output called `track4_`*n*`.trk`, where *n* is the example number.

Grading

5	Section 1, Part 1
20	Section 1, Part 2
10	Section 2