Software for Intelligent Robotics

Assignment #2
Out: Tuesday, September 3, 2002
DUE: Thursday, September 12, 2002

1. **STRIPS Planner**
   - Complete Exercise 2.7 from the Murphy textbook, page 64, and turn in your results.

2. **Programming Exercise, Part I: Waypoint following using (parts of) a hierarchical controller**
   - In the Nested Hierarchical Controller (Murphy, Chapter 2, section 2.4.1), the PLAN module consists of a Mission Planner, a Navigator, and a Pilot. The Pilot connects with the ACT module to execute the drive and steering commands that cause the robot to move along a path segment. In this exercise, you will build a Pilot to connect with the ACT module to move a simulated Nomad200 robot from its current position to a specified goal point. However, we will modify the interaction between the Pilot and the ACT module by using a servo-loop command structure rather than issuing longer line segment directions, as done in the “pure” hierarchical approach. In this servo-loop approach, vm() commands are used to continually turn the robot in the desired direction, rather than trying to command absolute turning angle and distance commands. You will also build a skeleton Navigator module to feed successive waypoints to the Pilot for the robot to execute. For this exercise, you will not need to include any robot obstacle avoidance (thus, none of the range sensors will be used). We will add the obstacle avoidance capability to the waypoint follower in a future exercise.

To accomplish this, program the following (See also the Helpful Hints at the end of this handout):

a) Create a simple ACT module that accepts as input (from the Pilot) the desired direction of motion from the Pilot and executes appropriate velocity and steering commands (using vm()).

b) Create a Pilot module that accepts as input (from the Navigator) an (x,y) goal point destination and generates the desired direction of motion based upon the robot’s current position and orientation. This motion should be generated using the servo-loop approach with a tight loop between the Pilot and the ACT module that we discussed in class, rather than attempting to command absolute turning angle and distance commands. This desired direction of motion is passed down to the ACT module for execution. The Pilot module also regularly checks to see if the robot has reached the current goal point destination. If it has, it returns a success flag to the Navigator indicating that the goal has been reached.

c) Create a simple Navigator module that passes down to the Pilot successive goal points to be reached (one at a time). Once the Pilot module returns a success flag for the current goal, the Navigator should send down the next goal point. Once all goal points have been reached, the Navigator module should declare success and the robot code should exit. For this exercise, the goal points in the Navigator should be a hardcoded, pre-defined list of (x,y) points to be followed successively.

d) Connect all the modules and run the Nomad simulator robot with a starting point of (0,0), successively moving to the following (x,y) goal points:
   - i. (1700,700)
   - ii. (1800, -1500)
   - iii. (-700,700)
   - iv. (-800, -1300)
   - v. (2700,200)
   - In this exercise, your environment should be obstacle-free.
3. Programming Exercise, Part II: Toward Navigator Path Planning Capabilities

- While we are not yet ready to build an automated path planner, we can begin to see the requirements of a path planner output. In this exercise, you will plan a path (by hand) using a series of waypoints to get the Nomad robot from its starting point of (0,0) to the (x,y) goal point of (-975,1330) in a specific map. Download the map in the file `map3.txt` from our course web site at: [http://www.cs.utk.edu/~parker/Courses/CS594-fall02/Assignments/map3.txt](http://www.cs.utk.edu/~parker/Courses/CS594-fall02/Assignments/map3.txt). By hand, generate two alternative series of (x,y) waypoints that the robot must follow to reach the goal point of (-975,1330). Make your two paths follow qualitatively different routes to the goal. Execute your code twice from Part I to execute the two paths defined by the two series of waypoints.

- Turn in a hard copy of the following:
  - Two screen dumps of the results of your program executing to control the robot through the two series waypoints from (0,0) to (-975,1330). The screen dumps should show the final position of the robot, with the trace of robot motions from the starting point to reach this final position.
  - A list of your selected waypoints for each of the two paths.
  - An answer to the following questions: When a path planner generates a series of waypoints for a robot to follow to reach a goal, what considerations do you think must be taken into account to determine a “preferred” path, when multiple paths are possible? Did you have to fine-tune the parameters of your Part I solution (e.g., velocities, steering rate, etc.) to generate appropriate paths for this exercise? Please explain.
  - (No electronic version needs to be turned in for this part of the exercise.)

Helpful Hints for Programming Exercise, Part I:

- To determine the current position of the robot, use the state variables `State[STATE_CONF_X]` and `State[STATE_CONF_Y]`. You must call `gs()` in order to refresh these state variables. [Note that while an actual indoor robot would not have such easy access to accurate internal x,y values, a localization software procedure onboard the robot could provide these values, within some margin of error. We’ll study this later this term.]

- To determine the current orientation of the robot, you can use the compass, whose current value is found in `State[STATE_COMPASS]`. Note that to use the compass, you must initially configure it (`conf_cp(1)`), and you must call `get_cp()` in order to refresh this state variable. You must also initially
mask the compass “on” using the sensor mask. [Another note: on a physical mobile robot in an indoor environment, the compass is likely to be unreliable, due to interference with the building structure that distorts the compass reading. In practice, orientation would be obtained from the localization procedure mentioned earlier.]

- You will have to do some geometric transformations to make all angle measurements consistent. Note the following:
  - The compass state variable returns values from 0 to 3600 (in tenths of a degree), with 0 in the positive x direction, and the angle value increasing in the clockwise direction.
  - A negative velocity command turns the robot right (e.g., vm(0, -50, -50)). A positive velocity command turns the robot left (e.g., vm(0, 50, 50)).
  - The atan2(y,x) function returns radian values from (-π to π), with 0 in the positive x direction, and values from 0 to π in the counterclockwise direction from 0, while values from 0 to -π are in the clockwise direction from 0.
- You can consider that a robot has reached a goal position when it reaches a distance of less than or equal to 100 (10 inches) from the goal position.