Homework 7 - due 10/29/2003

15-491: CMRoboBits

1 Introduction

This lab will build on your experience at writing behaviors and teach you how to access the local model to access recent visual information about the world around the robot. You will make the robot traverse a twisted path between walls without touching the walls. There will be only a single path through the maze (i.e. no need to back track or try to build a map).

2 Background

You can access the local model by using the LModel pointer in the FeatureSet. (Subscribe to the FeatureSet like you did in previous labs)

Here is the declaration from agent/Behaviors/FeatureSet.h:

- LModel *local_model;

There are two different methods to retrieve information from the local model.
The declarations of the methods and structures needed are in: agent/WorldModel/LocalModel.h

2.1 Method 1

- void calc_occ_grid_cells(int x1,int y1,int x2,int y2);
- const OccGridEntry *get_occ_grid_cell(int x_cell,int y_cell);

The first method is to create an occupancy grid with the robot in the center of the grid. You specify a rectangle centered on the grid. The robot is at (OccGrid::EntryCenterX, OccGrid::EntryCenterY) and the units here are cells. Each cell is a 200mm square). So a call to calc_occ_grid_cells(OccGrid::EntryCenterX, OccGrid::EntryCenterY,
OccGrid::EntryCenterX+5,
OccGrid::EntryCenterY+5) would create a map of the quadrant in front of the robot and to its left extending 1 meter out. (the first quadrant in the coordinate system centered on the robot). You then retrieve information using the get_occ_grid_cell interface. Pass it the (x,y) cell cordinate of the region you are interested in (the robot is at (OccGrid::EntryCenterX,OccGrid::EntryCenterY)). It returns a pointer to an OccGridEntry structure which contains and obs field

- OccGridEntry *cell = get_occ_grid_cell(OccGrid::EntryCenterX+1, OccGrid::EntryCenterY+1); // look in front + left
- cell->obs = a float between 0 and 1. 1 = sure it's an obstacle
- cell->evidence = a float telling how certain we are about this cell
- cell->wall_conf = confidence it's a wall
- cell->unknown_obs_conf = confidence it's an unknown object
- cell->ball_conf = ...
- Look at struct OccGridEntry in the header for more fields.

Be aware that get_occ_grid_cell does *not* bound the cell values that you pass to it. The robot will crash if you pass it out of bounds values. There are OccGrid::EntriesHoriz cells along the X-axis and OccGrid::EntriesVert along the Y-axis. (As a rule of thumb, don't query cells more than 9 cells from the robot)

2.2 Method 2

- void calc_occupancy(OccGridEntry *cell,vector2f ego_basis,vector2f ego_center,vector2f range);

The second way to retrieve information is by querying an arbitrary rectangle. You pass the address of an OccGridEntry (the fields in the structure are explained above) to hold the return value. The first parameter, the basis is a unit vector representing the orientation of the rectangle relative to the robot. For example, to have the long edge of the rectangle oriented 90 degrees from the robot, you would declare basis like this:

- vector2f basis = vector2f(cos(M_PI/2),sin(M_PI/2));

The next parameter specifies the center of the rectangle. To center it on the robot, you would enter vector2f(0,0) here. To move it 50 cm in front of the robot, you would specify vector2f(500,0); Remember that the distances are in millimeters.

The final parameter specifies how large your query rectangle is. The x parameter represents the length of the major axis of the rectangle (the one aligned with basis). The y parameter is the minor axis (perpendicular to basis).

3 Procedure

Write a behavior to follow a path between walls. You must not touch the walls. You must complete the maze within 3 minutes. You can count on the following:

- The path will always be at least 50 cm in width

- There is only one path through the maze
- All turns will be a multiple of 90 degrees
- We will include a hairpin turn where the robot must end up shifted to one side and turned 180 degrees to proceed.
- The robot will start facing the correct direction with walls on either side of it.
- There is a time limit of 3 minutes

See the background section for details on querying the local model.
You must hand in the following:

- Your .cc and .h file for your behavior
- A copy of your memory stick image

See previous assignments for directions on copying the data to your AFS space.

4 Grading

Note that all homeworks are weighted the same when calculating your final grade. The number of points on assignments varies so we can divide the parts up nicely, but we normalize each assignment so they all have equal weight afterwards.

- 20 points - avoiding contact with the walls
- 20 points - distance traveled through the maze
  (3 minute time limit)

Contact is defined as touching the wall for any amount of time. However, we will not penalize you for a contact within five seconds of your previous penalty. So a robot continuously rubbing the wall would receive a penalty ever five seconds. A robot that touched the wall, moved away, and briefly touched again after two seconds would only receive a single penalty. The idea is not to penalize you twice for the same mistake.

Ideally we would like your robot to move quickly through the maze. However, you are not being graded on time, so if you can decrease contact with the walls by slowing down, reduce your speed (as long as you complete the maze before 3 minutes is up)