ECE250 Project 0: Potential Field Navigation

Due: Monday, September 23, 2024 @11pm to the Dropbox on Learn

Overview

The goal of this project is to become familiar with C++, Object-Oriented Programming, eceubuntu, Valgrind, gdb, make, and the autograder. You will be implementing a dynamically allocated array-based data structure. Use of the STL other than <string> is not permitted – you have to do it all yourself!

Understanding the Problem

Robot navigation is an important area of research in robotics and artificial intelligence. One approach to navigating a robot through an environment is potential field navigation. This method involves creating a 2D grid representing the robot's environment, where each cell in the grid, representing a position in space that the robot can occupy, has a potential value. These values simulate attractive and repulsive forces, guiding the robot towards goals and away from obstacles.

In this project, you will dynamically allocate a 2D array to represent the grid. You will then implement functionality to add goals dynamically. Each goal will exert an attractive force, lowering the potential values in its vicinity. Obstacles, on the other hand, will exert repulsive forces, increasing the potential values around them. A robot using potential field navigation contains this 2D grid as a map. It moves through the potential field by moving in the direction of lowest potential given its current position. In this way, it seeks out goals and avoids obstacles.

Computing the potential values due to goals or obstacles

To simplify the programming, the origin (point (0,0)) is at the top left corner of the map. This looks like this:

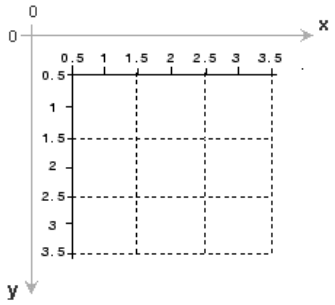

Figure 1: The coordinate system. Why would this simplify programming?

Presume that we are interested in the grid point (x,y) and the goal we are interested in is at point (xG,yG). The potential[1] at point (x,y) due to the goal is given by:

$$\vec{P} = \langle -\frac{K_{\square}}{\sqrt{(x-xG)^2+(y-yG)^2}}, -\frac{K_{\square}}{\sqrt{(x-xG)^2+(y-yG)^2}} \rangle$$

[1] For those of you who remember physics, you will be angry that my potential is a vector field. I *know*. However, the actual robotics algorithm requires the computation of a gradient, and that is not needed for this project. Just go with it.

Where K is a constant that we set in our program. For obstacles the computation is the same, but the potential is positive. The total potential at that point is the sum of all the potentials due to all goals and obstacles in the environment. Assume that if the robot and a goal or obstacle occupy the same square, the potential due to the goal or obstacle is zero there[2].

Program Design and Documentation

You must use proper Object-Oriented design principles to implement your solution.

You will create a design where the array is represented by class(es) which have appropriately private data members and appropriately public services (member functions). It is not acceptable to simply make all data members public and structs are not permitted.

You must submit a brief design document for this project that outlines your class design. Note that in future assignments you will also need to specify the runtime of your algorithms.

Input/Output Requirements

In ECE250, we will be testing your submissions via scripts written to run on Linux. To do this your solution must follow specific formatting requirements.

You must create a test program that contains your main program. This program must read commands from standard input and write to standard output.

The program must respond to the commands shown in Table 1. The outputs listed in the "Output" column must appear as shown. For instance, the first row has "success" as an output. This means that the string "success", written in lowercase, is expected, and if more input cases exist the output will continue on the next line. The program ends (and destructors are called) when standard input is empty.

Table 1: Testing Commands

Command	Parameters	Description	Output
CREATE	N M	Creates a new map array of size N x M. You may assume N > 0 and M > 0, and that both are integers. If a map already exists, it must first be deleted then a new one created. All potential vectors must be set to $\langle 0,0 \rangle$. Set K = 1.	**success**
POINT	T X Y	Adds a new goal (if T == 'G') or obstacle (if T == 'O') at position (X,Y) if that position is within the array. You may assume that X and Y are always given as integers and T will always be one of 'G' or 'O'. Recomputes the potential of all points in the map if insertion is successful. If a goal or obstacle already exists there, overwrite it.	**success** if the insertion was successful **failure** if the location is out of bounds.

[2] If this bothers you, feel free to pretend that there is some executive AI controlling the robot's decisions. Maybe that AI knows how to ignore goals that are already reached. We are not implementing that in this assignment because the point is to program an interesting data structure, not solve all of robot navigation!

Command	Parameters	Description	Output
MOVE	X Y	Outputs the direction the robot should move in if it is at position (X,Y). Assume X and Y are integers.	**Px Py** the value of the potential field at (X,Y) if (X,Y) is within the map **failure** if (X,Y) are outside of the map
CLEAR		Clears the array of all obstacles and goals and resets the potentials in it to ⟨0,0⟩	**success** if the map has been created **failure** if the map has not yet been created
UPDATE	K	Updates the value of K to use in the potential field calculations. Recompute all potentials if K is valid. Note that K can be a decimal number.	**success** if K > 0 **failure** if $K \leq 0$
EXIT		All input files end with EXIT. This command produces no output.	

Test Files, Runtime, and Memory Safety

Learn contains some examples for input files with the corresponding output. The files are named test01.in, test02.in and so on with their corresponding output files named test01.out etc. 5% of this project's autograding grade is allocated to whether your program has memory errors, determined by Valgrind. Prove that your MOVE command has runtime O(NXM).

Submitting your Program

Once you have completed your solution and tested it comprehensively on your own computer or the lab computers, you have to transfer your files to the eceUbuntu server and test there. We perform automated tests on this platform, so if your code works on your own computer but not on eceUbuntu it will be considered incorrect. Once you are done your testing you must create a compressed file in the tar.gz format, that contains:

- A test program containing your main() function.
- Required header files that you created.
- Any additional support files that you created.
- A makefile, named Makefile, with commands to compiler your solution and create an executable named a.out. This makefile will be run using the "make" utility. Do not specify an output file name, the default of a.out is used in the automated testing.
- Your design document (maximum two pages) , which must be submitted as a PDF document. Any other format will not be accepted.

The name of your compressed file should be xxxxxxxx_pn.tar.gz, where xxxxxxxx is your UW ID as above and n is the project number. In my case, the compressed file would be mstachow_p0.tar.gz.