16x62: Lab1

Due: 1.0, 1.1 Tuesday, Week 3
1.2, 1.3 Tuesday, Week 4

Introduction

The purpose of this lab is to transform you into experienced mobile robot programmers by writing position relative control programs that you will be using throughout the quarter and well-behaved functional programs.

Assignment 1.0 & 1.1: The Position Commands

Write `TurnTo(int <tenths-of-degrees>)`
When called, the robot will turn `<tenths-of-degrees>` relative to its current orientation. Degrees are measure counterclockwise from the current orientation. Furthermore, the robot should turn the shortest way, for example, if given `TurnTo(2700)` the robot should turn clockwise 90 degrees.

Write: `GoTo(int <tenths-of-inches>)`
When called, the robot moves `<tenths-of-inches>` forward if `<tenths-of-inches>` is positive, otherwise backward.

These are low-levels commands that you will be calling, just like `setVelocities()`. The difference is that these commands specify movement to achieve relative positional change instead of specifying motion velocities. You need to attain at least 90% accuracy using these functions. (You probably want much, much better accuracy than this for the sake of the rest of your semester's work!) This means you will probably want to use the encoders. Remember that the encoders' coordinate grid may not correspond to the direction in which the robot must move for `GoTo()`.

For accuracy and increased speed, you need to consider both PID control and delay-compensation control. Remember that the encoders have a time lag so the robot will always be slightly ahead of where its encoders say it is. Final hint: you should seriously consider reconfiguring the acceleration of your robot to a value of your own choosing, so that whatever value the previous team used doesn't affect your code.

Assignment 1.2: The Smarter Wanderer

Write the purely reactive program: `SmartWander`
When called, the robot will explore the environment without hitting anything. The robot should always continue to explore; it should never enter into an endless loop such as repeatedly bouncing between two obstacles.

`SmartWander()` should exhibit both translational and rotational movement. We will test your program not only by placing the robot in a static environment populated with walls, but also by attempting to herd your robot around. This means your robot will be faced with moving human beings. We promise not to wear sonically inert fur.

We will test this behavior by setting the robot loose in an area with an arbitrary scattering of walls and people. The robot should wander smoothly without hitting anything. We'll be looking for the fastest robot that exhibits smooth and safe behavior. Safety is critical; speed is desirable. Don't let your robot get stuck in a silly infinite loop, that will cost you.

This code must be totally functional. This means that you may only use the current sonar readings to determine your new velocity. You WILL receive bonus points if your robot plays any sounds at all.

Assignment 1.3: The Smarterer Wanderer

Improve on your wandering program by adding state. Use state to ensure that the robot will never, ever become stuck in a silly situation forever.

Your challenge now—and this is indeed a challenge—is to actually improve on your stateless wandering program. So find a fault and use state to make it better. This is easy to do, but the real trick is doing this without accidentally ruining all the accidental good behavior of your reactive program at the same time. In class, you will be describing how you used state surgically to improve things rather than horribly denature things. As a special bonus, you will receive extra credit if you choose to do the *Advanced* version of 1.3, in which, rather than adding state to improve your wander, you design your wanderer so that it adjusts its parameters automatically to improve itself! More on this in 1.5 weeks.

Hints

Thursday's lecture will explain much of what you need for 1.0 and 1.1. Don't work too hard before then.

Start with something simple. The KISS (Keep It Simple) principle is very important for this assignment, particularly 1.2. You may want to start out by using only a small number of sonars for the functional programs.