MISSION #5

High Level Robot Control

Due: Week 10

Introduction:

Last week, you solved the basic problem of navigating down a narrow corridor. In this lab, you will create a more robust program that can operate even in the absence of walls. At the beginning of this course, you programmed using low-level perceptual inputs (infrared sensors) and low-level motion control outputs (*SetVelocity()*). The program you write this week, along with *TurnTo()* and *WhatDoISee()*, implement a higher level set of inputs and outputs.

Assignment 3.0: Go To Next Node

Task: Write a robot function called ***GTNN(int <num-nodes>)***

Description: When called this function will move the robot from the center of a node to the center of the node <num-nodes> in front of the robot in one continuous action. If it is not able to proceed all the way it should stay at or return to the center of the furthest node it was able to reach. This function should return the number of nodes it was able to advance. This function must be immune to cumulative errors in the translation and rotation encoders.

As with *CorridorFollow()* and *WhatDoISee()*, you may assume that during our tests the robot will begin within 10 centimeters of a node's center and within 22 degrees of rotational alignment with the maze. Actually, we'll keep it as well-aligned as is reasonable! In order for *GTNN()* to work indefinitely, you will need to ensure that when *GTNN()* finishes, it is again within 10 centimeters of the center of a node and within 22 degrees of alignment.

GTNN(), *TurnTo(90)*, and *TurnTo(-90)* should provide all the output primitives you will need for robust navigation in arbitrary mazes. The challenge is writing a *GTNN()* that is unaffected by encoder error and therefore can be used indefinitely without performance degradation. Your *GTNN()* faces two tasks in order to meet these requirements. First, it must consistently place the robot near the center of the node – near enough to guarantee that the next *GTNN()* is successful! This eliminates x-y encoder error. Second, *GTNN()* must rotationally realign the robot with the world sufficiently to guarantee that the robot is aligned closely enough to allow both *GTNN()* and *WhatDoISee()* to succeed. This eliminates rotational encoder error.

Note the subtle fact that *GTNN()* cannot correct drift in all directions on every move. If the robot moves to an open node (no walls on any side), no drift correction is possible. If the robot moves to a node with walls on the left and right, no drift correction in the forward/backward direction is possible. Although we will be working in larger environments in the final labs, we will never give you a map that has more than three consecutive nodes in which no drift correction is possible in one of the axis directions. So, you needn't worry about the huge cumulative error that would result if the robot travels a corridor that is twenty nodes long with no forward-backward walls, for example.

We will test your *GTNN()* by running your robot around in a maze using a sequence of *GTNN()*'s and *TurnTo()*'s while observing the robot for gradual performance degradation. We will also test *GTNN()* by asking it to move more nodes than there is room in the maze. Please create an input interface for us so that we can enter a sequence of commands (such as: ***G1 L G2 R G1 L G2*** ((translation: *GTNN(1)*, *TurnTo(90)*, *GTNN(2)*, *TurnTo(-90)*, … etc.))) and you will execute that sequence in a loop. I repeat: IN A LOOP.

Hints:

- KEEP IT SIMPLE. This lab is not easy, and you won't be doing yourselves any favors by jumping right in with a really fancy solution. Start with the basics and add on as time permits.

- Don't hit anything. This includes brushing walls on the side of a corridor as well as smacking right into obstacles that block the robot's path.

- One of the most difficult parts of this assignment is getting *GTNN(1)* to work. Be sure to test this case well.

- You might want to use stepwise refinement for your *GTNN()* program:
  1) Make *GTNN()* work like corridor-follow with walls missing;
  2) Convert units from centimeters to nodes and add features to localize the robot within a node;
  3) Add obstacle avoidance;
  4) Add bells and whistles (e.g. using rotational encoders).

- It may be useful to think of the problem in terms of three phases of action: starting out, moving, and stopping. When you are starting out, you will want to locate the robot's position within a node (using *WhatDoISee()* and some infrared values and some math). As you are moving, you need to be smart about identifying when there are side walls which you can use to navigate. For example, don't use a side wall when the robot is halfway between two nodes. Finally, when you stop, you can use the nearby walls to localize (learn how far off the center of the maze node you are).

- You will, of course, use the encoders to determine the distance the robot travels, but you can also use the rotation encoder to align the robot with the maze when there are no walls around. Note that you first need to determine what direction is "straight", and you will have to update this direction whenever you turn.

- You can improve performance by maintaining information between calls to *GTNN()* (such as what direction is straight).