# 16x62: Lab2

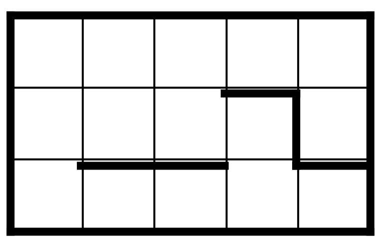
**Due: Tuesday, Week 5**

## Introduction:

Now that you are expert robot programmers, it is time to put the concepts you have learned in the past two weeks together. In the last lab, you wrote low-level control units to move the robot to a desired position. You also wrote sonar-based, functional programs that exhibited interesting and dangerous behaviors, sometimes even avoiding collisions. In this lab you will be creatively combining sonar-based collision avoidance with a control system into a single reactive program. This is actually EASIER than last week's assignment!

In the first assignment, you will write code that uses static sonar readings to provide more abstract information about the robot's state. In the second assignment, you will make the robot move safely down the center of a cardboard corridor.

Beginning with this lab, we will be using a more perfect world in which the robots will play. The following figure illustrates our new node-based world (which does not have a fixed size). A node is 29 in. by 29 in. Interior walls are always rotationally aligned, parallel to either the y-axis or the x-axis.



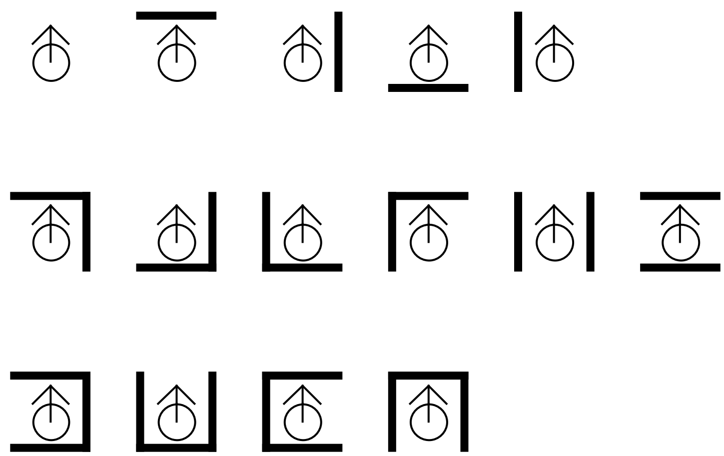
Example robot world

## Assignment 2.0: Sensor Interpretation

Write `WhatDoISee`
When called this program returns information about the existence of walls in all four cardinal directions around it, according to the chart below.

There are sixteen perceptually distinct nodes and robot orientations. It is important to realize that these percepts are relative to the robot's current direction. The same node may have a different `WhatDoISee` result if the robot simply turns 90 degrees.

The result of `WhatDoISee` is a description of whether or not there is a wall in each of the four directions relative to the robot.



**IMPORTANT**: You may assume that during our tests the robot will be within **5** inches of the node center and within 22 degrees of rotational alignment with the environment. You are writing this function is to use it throughout the rest of the course, so make it good!

When we test this code, we want to push a button. You should then update the sonars and then use `WhatDoISee()` and communicate the wall structure back to us on the screen. **The robot should not move.**

## Assignment 2.1: Corridor Following

Write the program: `CorridorFollow(int <distance>)`
When called, this function moves the robot <distance> **tenths of inches** along the center of a 29-inch wide corridor. If the robot is able to travel the entire distance, the function should return 0; otherwise, if it is blocked, the robot should stop before impact and return the distance remaining (in tenths of inches).

`CorridorFollow()` should be accurate to within **5 inches** of the requested distance, provided that the robot is not blocked by an obstacle along the way. The obstacles we place in the corridor will be walls across the hall. Your robot should stop before it hits an obstacle and should avoid hitting the sides of the corridor, too.

Your program has two goals: 1) stay along the centerline of the corridor as you travel down the corridor; and 2) travel the specified distance if the path is obstacle-free and, otherwise, stop and exit gracefully.

As with `WhatDoISee`, you may assume that during our tests the robot will start within 5 inches of the corridor center and within 22 degrees of rotational alignment with the corridor.

For our testing purposes, make an interface where we type the distance to be traveled and, after `CorridorFollow` returns, we see the distance remaining (or 0).

## Hints

- Don't depend on just a single sonar in each cardinal direction for `WhatDoISee`. If you use multiple sonars and some type of `min()` function, you can filter away the possibility of being fooled by a single specular reflection when the neighboring sonar is working. **This `WhatDoISee()` is actually an extremely easy assignment-- no trickery required**.
- Make sure that `WhatDoISee()` isn't fooled by the configurations of neighboring nodes. That is, `WhatDoISee()` is detecting only walls on the border of the robot's current node. If the node the robot is in is open, for instance, then the return value should be that there are no surrounding walls, even if the adjacent nodes have walls in place (you may be able to see the edges of such walls with a sonar).
- While moving down a corridor, there are two aspects of control that you need to worry about: 1) staying parallel to the walls, and 2) staying in the center of the corridor. Use the <u>sonars</u> for both of these purposes. You can look at the current sonars, decide if you're angled toward or away from the close wall, and whether you're to the left or right of the hallway centerline. Including being all lined up, this makes for four cases, and your reactive system could simply assign different `wheel speeds` for each of these four cases. That's a good, simple way to start this assignment. Then you can experiment with slightly less reactive systems—such as systems that actually do PID control to maintain hallway centerline, with the hidden variable being your angle to the hallway.
- When trying to write corridor-follow, you need to have a good visual on how the robot thinks it's misaligned in the hall. Consider doing this with the addcontinuousrobot function in MazeGraphics, so that you can really create a sense of its predicted angle relative to the walls. This is great diagnostic feedback for you, and can result in a cool animation on-screen.