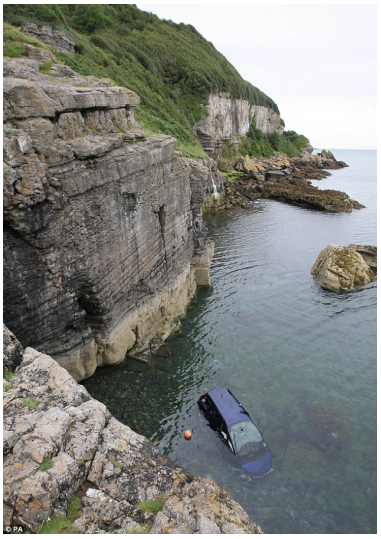
CS 4758/6758: Robot Learning: Homework 2

Due: Feb 23 (Tuesday), 5pm

1 Cost Functions

You are trying to train an autonomous vehicle to navigate mountain roads. One particularly treacherous stretch winds along the side of a cliff, with its face to the left and a steep drop-off to the right. Denoting the lateral position of the car by x, design a function $J(x) \geq 0$ describing how (sub)optimal its course is. [E.g., x is zero in the center of the road, and very high to the right.] Draw a graph of J and explain your choices. (5 points, plus bonus)

2 Learning in your Project

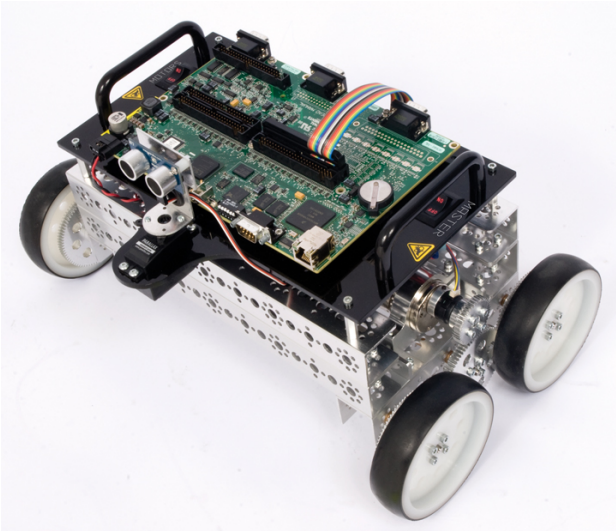
Please describe briefly how supervised learning can be applied to your project. Or you can talk about other learning methods (unsupervised learning, semi-supervised learning or reinforcement learning) and how are you planning to apply them in your term project. (5 points, plus bonus)

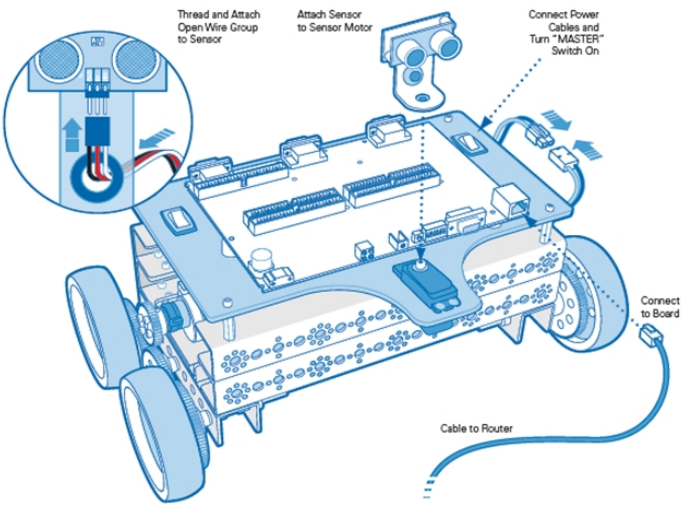
3 Sensor Noise

A robot (Figure 3) is equipped with three types of ranging sensors: laser, ultrasound, and infrared. To test the sensors, you place them in front of obstacles at known distances and measure their output. The file *sensors.train* contains the resulting data.

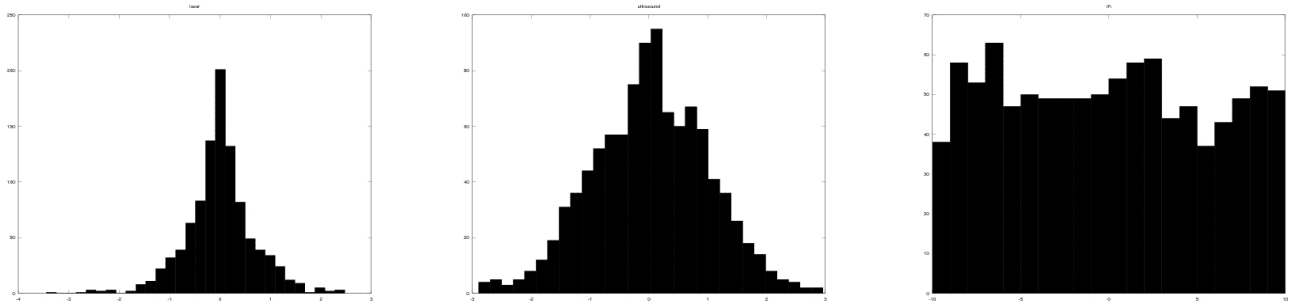
Call the vector of true distances y, and the matrix of sensor readings X (each row corresponds to a single reading, each column to a sensor). You want to figure out how to best combine the sensor data to accurately estimate y. You could compute a weighted average: using a vector of weights β, $\hat{y} = X\theta$ gives the estimated position. But what weights to use?

A) Find θ that minimizes $\|y - \hat{y}\|_2^2$. [Note: $\|v\|_2^2 = v^T v$] Report it.





Now lets take a different approach. Suppose for each sensor i ($i = \{1, 2, 3\}$), its readings x_i are related to the correct distance by $x_i = y + \epsilon_i$. Then ϵ_i represents the sensor's error. Plotting histograms of ϵ_i yields:



B) For each error signal, decide whether it is best modelled by a Gaussian, Laplacian or uniform distribution. For Laplace and Gaussian distributions, compute maximum likelihood estimates of their parameters ((μ_g, σ) or (μ_l, b)) from the data in *sensors.train*, and report them.

C) Assuming the probability distributions of **B)** to be correct, and also assuming that each sensor makes independent errors. You take a new triplet of measurements–one from each sensor (x_1, x_2, x_3). Write an expression for the likelihood of a position y; that is $\Pr(y|x_1, x_2, x_3; \mu_g, \mu_l, \sigma, b)$.

D) Assign y its most likely value. That is, let the estimated position $\hat{y} = \arg\max_y \Pr(y|x_1, x_2, x_3; \mu_g, \mu_l, \sigma, b)$. If possible, give a closed form expression for $\hat{y}$. Otherwise, please specify how the maximization problem could be solved. Intuitively, how does this filter work? Is it linear?

E) Which algorithm is better? The first algorithm (linear regression) or the second (statistical modeling)? To decide, measure their performance on a test set. The file *sensors.test* contains different sample of identically distributed data. Report the RMS error of each algorithm. [Note : RMS error or root mean square error is $\frac{\|\hat{Y}-Y\|_2}{\sqrt{n}}$ where n is the number of observations.] Explain the results.

(40 points)

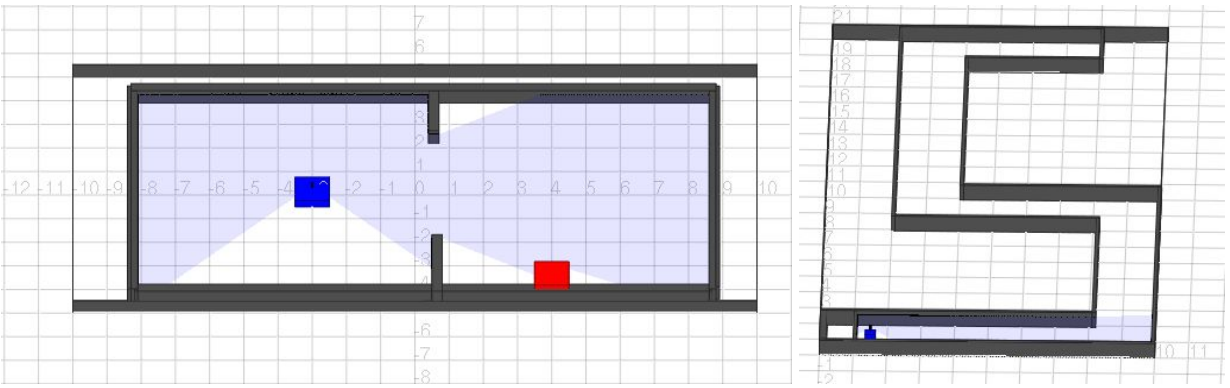


Figure 1: (a) Robot trying to get to the charging station, (b) Robot trying to navigate a maze.

4 Controling a Simulated Robot

In this problem you will write a ROS node to control a simulated robot (Figure 1). You need to write a controller that uses laser data and the odometry to navigate in the environment. The robot can only move forward and backward in its x direction and rotate around its z axis.[1]

Part 1: You are given a piece of code (python and C++) that drives the robot from one side of the lab to the other.

Now, the bot's batteries are low and you want it to go recharge by using odometry. The starting location (the blue spot, x=0, y=3, angle=0) and the location of charging spot (red spot, x=-3, y=-4) are known. (See Figure 1a.)

Load the included simple.world file into stage (ie. Run stageros with simple.world as its argument). Modify either controller.cpp or controller.py and include it in your project so running it makes the simulated robot park at the charging station (the red block). Write x and y location of the robot to a file. Submit the graph of bot's progress (x and y location w.r.t. time) along with your code.

Part 2: You want to modify the controller to guide your robot through an *unknown* maze.

Description of the maze: The maze is simple and can have only one possible path. The walls are less than 1 unit away from the robot except in the direction it needs to go and the direction it came from. All turns will be at 90 degree angles and the path is always parallel to the x or y axis. I.e., it is a pretty simple maze. (See Figure 1b.)

Load maze.world and modify your controller to use laser data (`wallDistForward,wallDistLeft,wallDistRight`) to solve the maze. Do not hardcode the order of the turns into your controller; use the available sensor data to decide when the robot should turn and in what direction. Write x and y location of the robot to a file and submit a graph of the bot's progress along with your code.

(50 points)

ROS Installation:

This requires a ROS installation. If you do not have access to ROS you can follow the instructions on http://www.cs.cornell.edu/courses/CS4758/2010sp/materials/ROS_Install.pdf

Get started on this as soon as possible, the software has a long installation period (about 1-2 hours of computer time). If you did not come to the ROS lecture or are having trouble understanding the problem, be sure to look at http://www.ros.org/wiki/ROS/Tutorials.

Read through http://www.ros.org/wiki/simulator_stage/Tutorials/SimulatingOneRobot. This page outlines how to run stage to simulate a robot. You may need to run `rosdep install <package name>` to install

[1]The rotation in the z axis is described in radians. The units of all these measurements are the same so setting the velocity to 20 for .1 seconds will move the robot 2. It is important to note that by default the robot will stop if more than .2 seconds passes between commands.

the required dependencies (required helper programs) before running rosmake.[2] Since some of the dependencies to install rviz are very large (900MB) you may want to skip that part of the tutorial.

In this problem you will need to write a node to use the simulated sensor input to navigated the simulated robot through a simple obstacle course.

First you will need to create a package. Instructions for this are given at
http://www.ros.org/wiki/ROS/Tutorials/CreatingPackage
You will need to have the following dependencies for your package: `tf`, `nav_msgs`, and either `rospy` (pythin) or `roscpp` (C++) depending on the language you want to use.

Once your package is ready to learn to use ROS with one of these languages see
http://www.ros.org/wiki/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29
or
http://www.ros.org/wiki/ROS/Tutorials/WritingPublisherSubscriber%28python%29

Download `controller.py` or `controller.cpp` and put it in the source directory of your package (specified in the tutorials above). This is the file you would be editing for the controller algorithm. *Modify the controller sample code between the areas marked only.*

[2]Driving the robot around with the keyboard is a little problematic since sometimes key presses won't register because of the stage screen. If this is a problem minimize stage, try driving the robot, then restore stage to see if it moved.