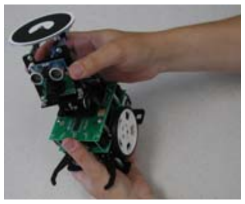
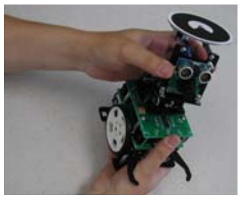
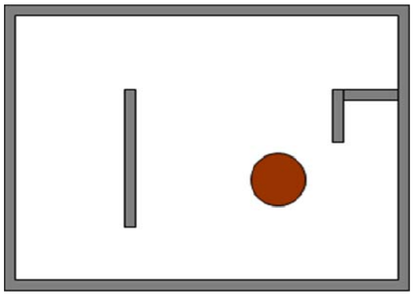


COMP4807 - LAB #4 Mapping (Due Sun. Nov. 11, 2012 @ 11:59pm)



In this assignment you will understand more about how to construct maps from various sensors. You will produce map data from performing edge following and navigation using your code from LAB3. The assignment looks long, but there is not too much coding to do here, provided that you did LAB3. The final part of the assignment can be done outside the lab. Concentrate in the labs on obtaining some useful and accurate maps of the environment.

Environment Setup:



For this lab, you should set up the roaming environment as shown in the picture here. Start up the **RobotTracker** and take a snapshot of this environment once you have it set up. Save the picture as **LAB4_Environment.png**.

Robot Setup:

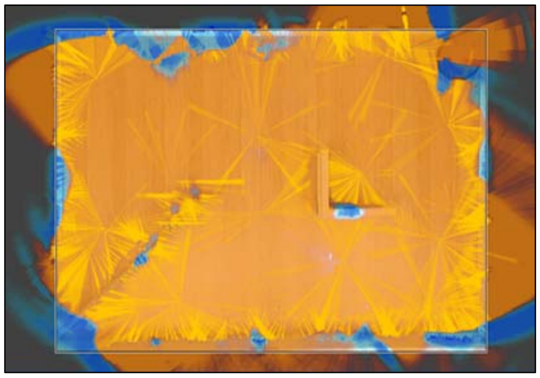
Make sure the robot is turned **off** and then turn **on** (i.e., down position) dip switches **1**(**F_IR**), **2**(**S_IR**), **3**(**BLU**), **6**(**SNR**) and **7**(**DIRS**) from the 8-position switch on the top of the robot and turn **off** (i.e., up) all other switches (i.e., **4, 5** and **8**). At the back of the robot, make sure that the 2-position dipswitches are both off (i.e., up).

The Fun Stuff:

1. Copy your **WallFollow.spin** code from LAB3 into a file called **BorderMap.spin**. Adjust the code to take various sensor range readings (you choose the sensors) and send them back to the PC for storage in a trace file. All sensor range readings should be in units of centimeters (the mapping application will convert them to pixels automatically). Using the **MapDialog**, load the trace file and view the map. Your objective is to create as accurate a map as possible. You will want to try to eliminate "garbage" sensor readings, which are sensor readings that go beyond the sensors range or invalid (i.e., -1 readings). Use a Planner called **BorderMapPlanner.java**.

   Once you are convinced that your code produces decent data sets, use the **RobotTracker** to record a short **.avi** video (called **BorderMap.avi**) of the robot doing one complete wall following cycle. Make sure to also save the trace trace file from the **RobotTracker** (renamed to **BorderMap.trc**) and a screen snapshot (called **BorderMap.png**) showing the actual path that the robot took (no map info shown...just the path using a visible high-contrast color as well as showing the background image so the environmental setup is clearly discernable). Your **BorderMap.trc** should contain ALL of the poses that the robot was in along with the data measurements obtained. Make sure that your robot moves slow enough to get adequate data. Don't worry if your robot does not perform perfect wall following ... just do your best.

The RobotTracker allows you to view sensor data sets separately or multiple ones at the same time. You can also examine the raw data, the data with the error range applied, the data with the Gaussian angle error applied, the data with the Gaussian distance error applied and finally the full Gaussian data. You can also set the resolution of the Gaussian error from the menu. Lastly, you can view the data as colored or gray scale and also vary the distance error and angular error of the sensor data. All of the "ugliness" involved with producing the map has been implemented for you, including the sensor fusion. The gray colors have also been automatically adjusted for better viewing (i.e., highest white to black contrast). For this part of the assignment, hand in snapshots (**.png** format) of this map window as described below. All snapshots should have the background **Tracker Image** displayed in the background, although it is likely that most of the image will be hidden when your map is displayed. The resolution should be set to 1x 6-sigma. Note that all images are to be color images.



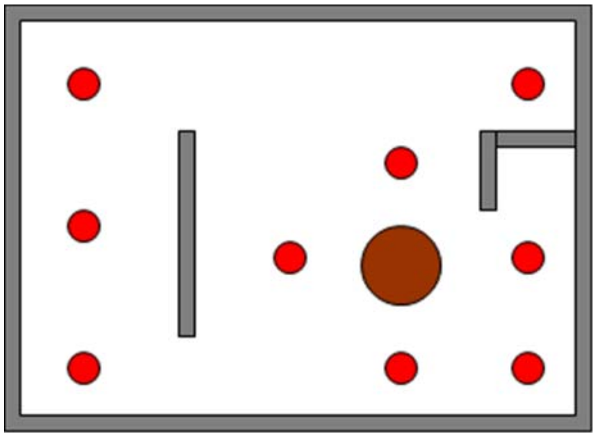
Submit snapshots showing each data set (i.e., each sensor) individually. For each dataset, include a snapshot (names shown below) showing (<u>note</u>: **xxx** is the name of the sensor):

a) **RawData** in color along without the background image (**WallRawXXX.png**)
b) **GaussianDistanceOnly** in grayscale along with the background image(**WallGDXXX.png**)
c) **FullGaussian** data in color along with the background image(**WallFGCXXX.png**)
d) **FullGaussian** data in grayscale without the background image(**WallFGGXXX.png**)

Include 4 more snapshots showing the combined results of all datasets:

a) **RawData** in color along without the background image (**WallRawALL.png**)
b) **GaussianDistanceOnly** in grayscale along with the background image(**WallGDALL.png**)
c) **FullGaussian** data in color along with the background image(**WallFGCALL.png**)
d) **FullGaussian** data in grayscale without the background image(**WallFGGALL.png**)

2. Copy your **Navigate.spin** code from LAB3 into a file called **NavigateMap.spin**. Adjust your code so that it accepts 9 points (shown as red dots here) and does a 360° spin at each of the points. Your robot should traverse any path between these points. Adjust the code so that robot **only** takes sensor readings at the 9 points. That is, between points, the robot should not supply readings. At each point (including the 1st), the robot should slowly spin, while repeatedly sending the sensor readings to your planner which you should call **NavigateMapPlanner.java**. You should base this planner code on your **NavigatePlanner.java** code



from LAB3. Once you are convinced that your code produces decent data sets, use the **RobotTracker** to record a short **.avi** video (called **NavigateMap.avi**) of the robot doing one complete wall following cycle. Make sure to also save the trace from the RobotTracker (renamed to **NavigateMap.trc**) and a screen snapshot (called **NavigateMap.png**) showing the robot's actual path and the 9 user-defined points such that they are clearly visible along with the background obstacles (no map). Make sure that your robot moves slow enough to get adequate data, as you will use this in the next steps.

Submit snapshots showing each data set (i.e., each sensor) individually. For each dataset, include a snapshot (names shown below) showing (note: **xxx** is the name of the sensor):

a) **RawData** in color along without the background image (**NavRawXXX.png**)
b) **GaussianDistanceOnly** in grayscale along with the background image(**NavGDXXX.png**)
c) **FullGaussian** data in color along with the background image(**NavFGCXXX.png**)
d) **FullGaussian** data in grayscale without the background image(**NavFGGXXX.png**)

Include 4 more snapshots showing the combined results of all datasets:

a) **RawData** in color along without the background image (**NavRawALL.png**)
b) **GaussianDistanceOnly** in grayscale along with the background image(**NavGDALL.png**)
c) **FullGaussian** data in color along with the background image(**NavFGCALL.png**)
d) **FullGaussian** data in grayscale without the background image(**NavFGGALL.png**)

3. You can do this at home after your lab tests have been completed. Write a stand-alone JAVA program called **TraceFixer.java** that reads in Trace files and produces a new trace file which can then be loaded into the **MapApplication** program and displayed. The objective is to get the most accurate map possible given your two sets of trace data, manipulating the data and then producing a new trace file that is readable with the mapper. The program should load in both the **BorderMap.trc** and **NavigateMap.trc** files, combining them into one final trace file called **BestMap.trc**. In addition to simply merging the files, you should look at ways of adjusting the data. One idea is to consider the fact that multiple readings are obtained between poses and perhaps distribute range data between two adjacent poses by creating additional pose estimates. For example, the three lines of poses below could be altered to spread the data along the line between poses (257,323) and (274,300).

```
257,323,46,4,13,4,12
-1,-1,-1,4,12,3,12,5,11
274,300,49,5,10
```

That is, you can replace the data above to something like this:

```
257,323,46,4,13
260,318,46,4,12
263,313,47,4,12
266,308,47,3,12
269,303,48,5,11
274,300,49,5,10
```

Basically, you can examine 2 consecutive valid poses and spread all data readings between them. Do what you can. You will be marked for improved map quality.

For this part, simply hand in your **TraceFixer.java** file and also a **TraceFixer.txt** file describing in simple text roughly the idea behind your strategy for improving the map. You should then submit two more images using your newly created **BestMap.trc** file:

a) **BestFull.png** - FullGaussian, grayscale with or without **Incorprate Robot's Shape** selected.
b) **BestFullColor.png** - FullGaussian, color with or without **Incorprate Robot's Shape** selected.

Submission:

Create a webpage for this course that contains a simple report on your lab (i.e., a description of what you did and what each part of the assignment was trying to do). Include downloadable links for all of your code as well as any snapshots, trace files and videos. There is no need to make the webpage beautiful … but that's up to you. Login to Carleton's WebCT system and submit a link to your website. Make sure that the website is up and running at least until your code is marked. If you have never created an HTML page before, you can use Mozilla's **Sea Monkey** browser which has a built in editor that allows you to make simple pages. Always keep a backup of all your work (perhaps on a USB key or burn a CD). Here is a summary of what to hand in:

JAVA CODE:
- `BorderMapPlanner.java`
- `NavigateMapPlanner.java`
- `TraceFixer.java`
- `TraceFiled.txt (explanation file)`

SPIN CODE:
- `BorderMap.spin`
- `NavigateMap.spin`

VIDEO CLIPS:
- `BorderMap.avi`
- `NavigateMap.avi`

TRACE FILES:
- `BorderMap.trc`
- `NavigateMap.trc`
- `BestMap.trc`

SCREEN SNAPSHOTS:
- `LAB4_Environment.png`
- `BorderMap.png`
- `NavigateMap.png`
- `WallRawXX1.png ... WallRawXXN.png`
- `WallGDXX1.png ... WallGDXXN.png`
- `WallFGCXX1.png ... WallFGCXXN.png`
- `WallFGGXX1.png ... WallFGGXXN.png`
- `WallRawALL.png`
- `WallGDALL.png`
- `WallFGCALL.png`
- `WallFGGALL.png`
- `NavRawXX1.png ... NavRawXXN.png`
- `NavGDXX1.png ... NavGDXXN.png`
- `NavFGCXX1.png ... NavFGCXXN.png`
- `NavFGGXX1.png ... NavFGGXXN.png`
- `NavRawALL.png`
- `NavGDALL.png`
- `NavFGCALL.png`
- `NavFGGALL.png`
- `BestFull.png`
- `BestFullColor.png`