# CS1112 Fall 2010 Project 5 Part 2 due Thursday 11/11 at 11pm

(Part 1 appears in a separate document. Both parts have the same submission deadline.)

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group.

## Objectives

Completing this project will solidify your understanding of numeric arrays, character arrays, cell arrays, structures, and structure arrays. You will also work with text data files. Part 2 covers 2-d numeric array, structure array and file input/output.

## MATLAB Files and Simulator Toolbox

You will modify the solutions to Project 3 (the robot control functions) and run the modified functions in the Simulator to obtain data that you will then analyze. Download the files `showCoverage.m`, `randomWalkRobot5.m` and `exploreRoom5.m` from the Projects page on the course website; the latter two are our solutions to Project 3 (with the code not necessary for this project removed). We highly recommend that you use *our* Project 3 solutions as the starting point for this project, but you may use your own Project 3 solutions if you wish. You also need the `CreateRobot.m` file from Project 3.

Some important reminders about using the simulator:

- Make sure that the map files from Project 3 (`squareEnclosure.txt` and `labSmall.txt`) as well as the files for this project are all in the Current Directory.
- To run a control program, e.g., `randomWalkRobot5`, in the simulator,
  1. Open the simulator window (type `SimulatorGUI`) in Command Window
  2. Load a map
  3. Set the position of the robot if necessary
  4. Click the *Start* button under `Autonomous` and select the program you wan to run.

## How well does the robot "cover" the floor?

In Project 3 we sent the robot to cover the floor of a secret lab. How well did it do? Did it cover the whole floor? Did it go over some regions more often than others? Did it systematically leave out some regions? Now we will record the robot's location every one-fifth of a second (approximately), divide the floor into subregions, and count how many "hits" the robot has made in each subregion. In essence we are producing a 2-dimensional histogram, but instead of seeing the results as the height of the bars in a histogram we will look at the results via the intensity of color—we will use a color map. So Part 2 is about creating the matrix that would be the input argument to the `show2dData` function that you have completed in Part 1.

The iRobot Create Simulator does not allow us to write control functions that return variable values. In Project 3 we dealt with it by writing the code to produce graphics *inside* the control functions, but that was inflexible and we had to wait for the simulation to run to completion each time we changed the code. This time we will write the simulation data to a data file! Then we can use that data file for post-simulation analysis in any way we wish, anytime we wish. We will start with the `randomWalkRobot` program first since that simulation is shorter and you can develop and debug your code more easily with that than with `exploreRoom`.

# 2 Simulation data of the randomly wandering robot

First read the function `randomWalkRobot5` and note that we include `getRandAngle` as a subfunction. To run this program in the simulator you need to first load the map `squareEnclosure.txt`. (If you wish, use your old `randomWalkRobot` function but change the function name and include the subfunction to match ours). *Read the function specifications (comment) carefully!* Add code to `randomWalkRobot5` so that it creates a data file during simulation:

- The file is a plain text (ASCII) file.
- The name of the file is `trip<hh><mm><ss>.txt`. For example, a simulation that begins at 5:06:40pm corresponds to a file named `trip170640.txt`. (See §2.1 below for useful commands.)
- The first line of the file describes the data file:
  `DESCRIPTION Position data from function randomWalkRobot5`
- The second and last lines of the file should be "time stamps" indicating the start and end time of the simulation, respectively. For example, the time 1:09:03pm should be written on one line of the file as
  `TIME 13:09:03.0`
- Each position recorded appears on one line beginning with the string `'POS'`. The position includes the x-coordinate, y-coordinate, and heading of the robot. For example, if the robot's center is at (.16,.49) with heading -.02, then the corresponding line in the file is:
  `POS 0.16 0.49 -0.02`
  The beginning position should be on the third line of the file.

## 2.1 File, start time, and useful tools

Add code to `randomWalkRobot5.m` to open the file for writing, write the description and data to the file, and close the file.

The statement `t = clock` assigns to `t` a numeric vector of length six where the components indicate the current year, month, day, hour, minute, and second. The first five components are integer values while the last is a real number accurate to several decimal points.

Recall that you can build a string using concatenation and/or with the function `sprintf`. Note these variations in the format string (substitution sequences) that you can use:

- The format string `%04d` uses four character spaces for the substituted integer, allowing for leading zeros. For example, `a=23; s=sprintf('a is %04d!',a);` results in a string `'a is 0023!'` in variable `s`.
- The format string `%08.1f` uses eight character spaces for the substituted number, including the decimal point and 1 decimal place and allowing for leading zeros.

You can easily try out the above and any other format strings in the Command Window!

To write to an opened file with the ID `fileID`, use the function `fprintf`, e.g., `fprintf(fileID, 'a string\n');` Put a semicolon at the end of the statement; otherwise a value (the number of characters written) will be printed to the Command Window.

You can see the data text file in MATLAB: just double click the filename in the Current Directory Pane and the text file will be shown in an Editor window. Some versions of the software *Notepad*, available on most PCs, do not display new lines (line breaks) correctly. Therefore it is best to view the text file within the MATLAB environment.

## 2.2 `TIME` and `POS`

Implement two functions, `timeStamp` and `writeLocation2file`, to do the actual reading and writing of the time and location. Then in `randomWalkRobot5` you will call these functions whenever one `TIME` or `POS` line is to be written. (*Hints*: (1) these functions are very short—don't be alarmed! (2) In developing your code, first simply print the lines to the screen instead of writing them to file. This way you can see in the Command Window whether there are mistakes in the line/format *during* the simulation instead of having to

wait until the simulation ends to check the output file. Once you've got the line/format correct *then* write each line to file as specified.)

In the random walk the robot travels at .5 m/s for 1 second in each step of the random walk (a `pause` of 1 second in the code), yet you need to take a location reading approximately every .2 seconds when the robot travels forward. This means you need to break down the 1 second of travel to accommodate the required location readings.

Implement this function to write a "time stamp" to file:

```
function timeStamp(fileID)
% Write current time to the opened file with the ID fileID:  put on one
% line the string 'TIME ' followed by the a string with the format
% hh:mm:ssss  where hh represents the hours string in 2-digit 24-hour-clock
% format, mm represents the minutes string in 2-digit format, and ssss
% represents the seconds string in decimal format (2 digits, the decimal
% point, followed by one digit).  E.g., the time 1:09:03pm should be
% written on one line of the file as
% TIME 13:09:03.0
```

Implement this function to write the robot's current position to file:

```
function writeLocation2file(fileID,serPort)
% Write the current position to the opened file with the ID fileID:  put on
% one line the string 'POS' followed by the x and y coordinates and the
% heading of the robot.  Each value is written to 2 decimal places in a
% field width of 10.  For example, if the robot is centered at (-14,.1)
% with a heading of -0.2 radians, the line to be written is
% POS    -14.00      0.10     -0.20
% Note that in the example above there are 4 spaces left of '-14.00', 6
% spaces left of '0.10', and 5 spaces left of '-0.20', i.e., each value is
% written in a "text field" that is exactly wide enough for 10 characters.
```

Get the robot's location by calling the simulator's `OverheadLocalizationCreate` function:

```
[xc,yc,theta]= OverheadLocalizationCreate(serPort)
```

`xc` and `yc` are the x- and y-coordinates (in meters) and `theta` is the angle (in radians).

Run `randomWalkRobot5` in the simulator to get your data file. Now you can move on to the next stage: read the data from the file, compute the matrix containing the "2-d histogram" data, and finally produce the color map.

# 3 Floor coverage by the randomly wandering robot

Download the function `showCoverage` and read the comments and code in `showCoverage.m`. The main function and subfunctions contain mostly "dummy statements" that assign bogus values to the function parameters so that the functions can execute. These dummy statements are often called *stubs*. We stub functions so that we can run them throughout code development for testing, even before the implementation is complete! Try this (dummy) function call now:

```
[M, C]= showCoverage('trip111111.txt',2,3,[.5 .5 .2])
```

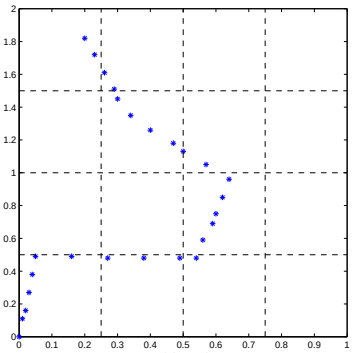
It doesn't matter that you don't have a file `trip111111.txt` in your directory. Assuming that you have correctly implemented function `show2dData` from Part 1, a color map will be produced. As you implement the (sub)functions for real you will remove the stubs. However, *do not change the function headers and specifications in any way.*

First, implement the specified subfunctions `makePosition` and `getData`. `makePosition` defines a Position structure and you must call it in creating the structure array specified in `getData`.

`getData` is in charge of reading the data file and creating the array of Position structures. Additionally it calculates some statistics from the data. The link *Reading text data from a file* on the Projects page summarizes the commands that you will need and gives an example.

The main function `showCoverage` starts with calling `getData` and then displaying in the Command Window the start time of the trip stored in the data file. A simple message such as '`Trip started at 17:06:40`' suffices.

In `showCoverage` next you compute the 2-d histogram data `M`. Let's look at an example:




| | | | |
|---|---|---|---|
| 2 | 2 | 0 | 0 |
| 0 | 4 | 2 | 0 |
| 0 | 0 | 5 | 0 |
| 7 | 3 | 1 | 0 |

Histogram data

The diagram on the left shows the position data (x-y coordinates only; no heading). If `nr`=4 and `nc`=4, then there are 16 subregions as shown in the diagram. The histogram data `M` that corresponds to the position data is shown on the right. I.e., `M(r,c)` is the number of hits in the subregion (r,c).

**Setting the limits of the region**

The smallest and biggest x values in the position data are `minX` and `maxX`, respectively; `minY` and `maxY` are defined similarly. We will set the region's limits to be integer values, so the region's x limits are `floor(minX)` and `floor(maxX)+1`. Notice that the upper limit is not `ceil(maxX)`. Why? Let's look at the example data in the diagrams above again and focus on the x dimension. The boundaries of the bins are at 0, .25, .5, .75, and 1. We must count in such a way that each asterisk falls in one bin (subregion) only:

$$\begin{array}{ccccccc} 0 & \leq & x & < & .25 & \quad & \text{count as column 1} \\ .25 & \leq & x & < & .5 & & \text{count as column 2} \\ .5 & \leq & x & < & .75 & & \text{count as column 3} \\ .75 & \leq & x & < & 1 & & \text{count as column 4} \end{array}$$

For example, there is a point at "exactly" $x$=.5 near the middle of the position data diagram; that point is counted only in bin (2,3), not the neighboring bin (2,2). In other words, an x-coordinate that falls exactly on a boundary is counted in the bin to the right of that boundary (direction of higher column number). Therefore the region's upper limit must be strictly greater than the max value. So if `maxX` happens to be an integer value, then the upper x limit for the region needs to be `maxX+1` to allow `maxX` to be counted. For both real and integer values then the upper limit is `floor(maxX)+1`, not `ceil(maxX)`. Similarly, the y limits are `floor(minY)` and `floor(maxY)+1` and a y-coordinate that falls on a boundary is counted in the bin *above* that boundary (direction of *lower* row number). Remember that *y-coordinate values increase with decreasing row numbers*.

To help you with testing, the data file that corresponds to the above diagram is given on the Projects page. See the file `trip170640.txt`.

Finally you can run function `showCoverage` on your data file to see the "coverage" of the random walk.

# 4 Coverage of our floor covering algorithm in `exploreRoom`

Three of the functions you wrote earlier, `showCoverage`, `timeStamp`, and `writeLocation2file`, are general—any data file following the format described earlier can work with these functions. Now you will modify `exploreRoom5` to open a data file, write to the file, and close the file.

First read the function `exploreRoom5`; this is the solution from Project 3 minus the code related to the cliff sensors for reading the reflectivity of the floor. To run this control program in the simulator you need to first load the map `labSmall.txt` or `lab.txt`.

Add code to `exploreRoom5` so that it creates a data file during simulation:

- The file is a plain text (ASCII) file.
- The name of the file is `trip<hh><mm><ss>.txt`.
- **Different from** §2: The first line of the file describes the data file:
  `DESCRIPTION Position data from function exploreRoom5`
- The second and last lines of the file should be time stamps indicating the start and end time of the simulation, respectively. For example, the time 1:09:03pm should be written on one line of the file as
  `TIME 13:09:03.0`
- **New**: Write a time stamp line in the file when the bumper (bump sensor) activates.
- Each position recorded appears on one line beginning with the string `'POS'`. For example, if the robot's center is at (.16,.49) with heading -.02, then the corresponding line in the file is:
  `POS 0.16 0.49 -0.02`
  The beginning position should be on the third line of the file.

As specified in the function comment, the robot location should be recorded about every .2 seconds when the robot is traveling straight (not rotating). The timing is approximate, based on the `pause` used. (Don't worry about the .01 seconds used to back up.) Your code in `exploreRoom5` should call the `timeStamp` and `writeLocation2file` functions to do the writing of the data.

Finally, you can run the simulation with the map `labSmall.txt` (or `lab.txt`) to produce a data file. Then use your `showCoverage` function to read the data from the file and see the floor coverage by the robot. Start with just a few bins (4 to 6 in each dimension) and then try around 20 bins in each dimension. Are there any gaps in the coverage? Are there subregions that are visited much more frequently? (These are just thought questions; you don't have to submit answers...)

Do submit your files `randomWalkRobot5.m`, `timeStamp.m`, `writeLocation2file.m`, `showCoverage.m`, and `exploreRoom5.m` for Part 2 on CMS.