January 01, 2008

Lab 3: Motion Control of a Planar Positioner - More Logical Branching & Looping in MATLAB

Bernard M. Gordon Center for Subsurface Sensing and Imaging Systems (Gordon-CenSSIS)

GE U111 HTT&TL, Lab 3

Motion Control of a Planar Positioner,

More Logical Branching & Looping in MATLAB

Contents

1 Overview: Programming & Experiment Goals **2**

2 Homework Assignments **2**

3 MATLAB Primer **3**

3.1 The `fprintf` Command . . . 3

3.2 Nested Loops . . . 4

3.3 The `rem` Function and Its Use in Integer Odd / Even Analysis . . . 5

4 Experiments **5**

4.1 Basics of Planar Motion Control . . . 5

4.2 Etch-a-Sketch . . . 7

4.3 Raster Motion . . . 8

4.4 Extensions and Challenges . . . 10

1 Overview: Programming & Experiment Goals

This session is used to reinforce basic (MATLAB) programming concepts, including

- advanced looping (nested "for" and "while" loops)
- further use of logical conditions
- working with 2D (matrix) arrays
- further use of script M-files
- additional commands for automated messages display

The driving application is programmable motion control of a precision planar positioner. Specific tasks include

- plotting simple shapes (squares) in an Etch-a-Sketch configuration
- plotting more complex shapes (an asymmetric letter)
- planning raster motion in an 11×11 points array

The Components & Equipment used in this experiment include:

- A planar positioner, including two stepper motors & controls, mounted on top of an aquarium. The rotation of each of the two motors is translated to a linear motion of the pointer, using a fine groove screw mount: one motor moves the pointer horizontally, and the other, vertically.
- A glass aquarium.
- A felt point pen holder, connected to the positioner; you will put the pen in place only immediately before use
- A clip board, mounted on top of the aquarium.

Experiments Outline & Goal. Our previous sessions concerned two themes: one was the use of ultrasound transducers to measure the distance to a remote object, and the other, computerized motion control of a stepper motor. Beginning this session, we will begin to combine both these themes: our ultimate goal is to create an underwater ultrasound three-dimensional imaging device. The device is composed of a single underwater ultrasound transducer, serving as both an emitter and a receiver, attached to a planar positioner / pointer. The ultrasound transducer will be used to measure the distance to a remote object. The positioner is composed of two high precision stepper motors that enable a two-dimensional, planar motion of the transducer. By measuring the distance to the remote object from a grid of points, we shall be able to reconstruct and plot its three dimensional shape.

Our first goal is to learn how to use the computer to control the positioner motion, to follow a planned 2D trajectory. With an attached felt-tip pen, you will, in effect, create a simple computerized plotter . Once we master the art of motion control, in subsequent sessions we shall turn our attention to the computerized use of the transducer.

2 Homework Assignments

- Pre-Lab Homework
  - Read this handout, including the MATLAB Primer, carefully
  - Refresh the definitions in Table 1-5, page 15
  - Complete reading Chapter 4 and refresh / reread Chapter 7
  - Do problems 4-10 and 7-8 in the book (include a printout of the MATLAB command window).
  - Do the Pre-Lab Tasks 1-7 listed in this handout. This includes the The preparations of M-Files to perform lab tasks, as listed along this handout.

    Pre-Lab programming tasks include:

  * A printout of your programs in the Pre-Lab Report that is submitted in advance of the lab, and
  * Bringing to the lab a memory stick that includes these programs. You will run the programs in the lab and will **not** have enough time to prepare them in the lab.

- Post-Lab report due one week after finishing lab.

3 MATLAB Primer

3.1 The `fprintf` Command

This section briefly elaborates on the use of the output command `fprintf` that was introduced in the MATLAB textbook. You should be aware that other uses of the `fprintf` command include formatting and storing data. You can find more on that use from MATLAB help (type `help fprintf` in the command window) but we shall not dwell on such uses here.

The command `fprintf` provides for a structured display of messages and data that is impossible with the `disp` command. The command format for that purpose is

```
fprintf('string',variable(s))
```

where `string` stands for a character string (i.e., a text), and `variables` is a list of MATLAB variables (including arrays) separated by commas. If no variables are involved, the effect is essentially the same as of the command `disp('string')`. There is one difference: unless a special ”newline” instruction is specifically included, the screen printout (and in older MATLAB versions, the next MATLAB prompt) will immediately follow the displayed string and the next display will not start in a new line. The following is an example of a command window diary -

```
≫fprintf('this is the first sentence'),fprintf('this is the second sentence')
this is the first sentencethis is the second sentence
≫
```

Table 1 lists commands needed to control text location. These commands must be included within the two apostrophes containing the text string.

`\n`	start a new line
`\r`	beginning of a new line (essentially, same effect as `\n`)
`\b`	back space
`\''`	(`'`, repeated twice) apostrophe
`\\`	\

Table 1: `fprintf` text formatting commands

%e	scientific exponential format with a lowercase ”e”
%E	scientific exponential format with an uppercase ”E”
%f	fixed point decimal format
%i	decimal format
%d	decimal format
%g	either %e or %f, whichever is shorter

Table 2: `fprintf` variable formatting commands

Pre-Lab Task 1. Answer the following

1. What single `fprintf` command produces the display

   ```
   "This is not fun"
   "But it \ could be"
   ```

   Make sure that the displayed text includes the double quotation marks and the backslash and assure that the next display command will be executed in a new line.

2. What format control (from Table 1) should be added to display the word `fun` on top of (overwriting) the word `not`

In order to combine a displayed text and a display of variable values you have to include a format command for each of the desired displays, and include the corresponding value as an entry in the variable list. Format options for decimal numerical display are listed in Table 2. To specify the number of decimal places used to the right of the decimal point, in each of the options listed in Table 2, you have to include the term `.k` between the `%` sign and the letter representing the display type (i.e. `e`, `E`, `f` or `g`). For example

```
≫fprintf('The elephant ate %.2f tons of grass',23.4567)
The elephant ate 23.46 tons of grass
≫fprintf('And her friend also ate %.8f tons of grass',23.4567)
And her friend also ate 23.45670000 tons of grass
```

(For more options, including binary and character variables, consult the MATLAB help `fprintf`.)

Pre-Lab Task 2.

1. Try the following example and include a printout of the relevant section of the command window in your Pre-Lab report.
   `fprintf('\r10Pi is %e, and is %E and %f\n',[10*pi,10*pi,10*pi])`

   Notice that when `variable` is a vector, its entries of are displayed, one by one, each time a format command is included in 'string'.

2. What `fprintf` command produces the following printout, where all numerical values are input variables

   ```
   I ate 37.45 toasts
   with 1.23e+002 pounds of jam,
   and drank 6.54321000E+005 galons of milk,
   and felt sick
   ```

3.2 Nested Loops

Pre-Lab Task 3. Include a printout of the MATLAB response to the following nested loop program in your pre-lab report. Explain the reason for the number of displays of the sentence "the outer counter is", and for the number of displays of "now we are at".

```
for k=1:2
     fprintf('the outer counter is %f\n',k)
     for m=1:2
               fprintf('now we are at (%f,%f)\n',[k,m])
     end
```

```
end
```

3.3 The `rem` Function and Its Use in Integer Odd / Even Analysis

As part of your reading assignment for this lab you reviewed MATLAB rounding using the functions `round`, `floor`, `ceil` and `fix`, as well as the modulus (or, remainder) function `rem`: Recall that the function `rem` yields the remainder after division: Given two numbers `x` and `y`, let k be the **rounding towards zero** `x/y`: `k=fix(x/y)`. In these terms

`rem(x,y) = x - k×y`.

EXAMPLE

```
≫rem(5,3)
2
≫rem(5,-3)
2
rem(-5,3)
-2
rem(-5,-3)
-2
```

Pre-Lab Task 4. Write a program (script m-file) that uses a logical *if* construct and the `rem` function to determine whether a positive integer `N` is even or odd. It should display the statement `N is even` or `N is odd`, as appropriate. The key trick is the fact that if `N` is even then `rem(N,2)=0`, and if `N` is odd then `rem(N,2)=1`. You may assume that the value of the variable `N` is defined before, so that it can be used in your script file. Run your program with `N=8` and `N=9`, and include a print of the relevant part of the command window in your pre-lab report.

4 Experiments

4.1 Basics of Planar Motion Control

The purpose of this part is to learn and practice the MATLAB commands, controlling the positioner. The final task of this lab will be to prepare the motion control program that will actually be used to operate our system as an imaging device.

The positioner control commands are quite similar to the commands used to control the single stepper motor, in Lab 2, in the sense that they include a direction command and a step command. Differences are due to two main changes:

- Now we control two motors: Motor #1 actuates horizontal (“x” axis) motion, and motor #2 actuates vertical (“y” axis) motion. Thus both direction and motion commands will include a reference to the specific motor.
- Flexible step size: The motor controller allows to specify step size. In their original setting, these precision motors allow step size to be specified in single microns (10e-6 m); since our experiments require much larger steps, the current setting accepts step size specified in multiples (including fractions) of 1 cm.

Two additional technical changes are associated with the the way the computer interfaces with the instrument control board, the motor power supply and, later on, the signal generator and the digital oscilloscope

- The instrument initiation program that is called from the MATLAB command window, at the beginning of the session, creates a MATLAB object called `dio` (standing for “digital input-output”). Both the direction and motion commands will include passing the `dio` object.

Specifically, the positioner MATLAB control commands are:

1. Initiation: type `dio=setup_dio;` in the command window.
2. Direction command: `direct(direction,motor #,dio)` where
   (a) `direction` is a number indicating the direction: 1 for motion towards the motor and 0 for motion away from the motor
   (b) `motor #` stands for the number of the motor: 1 for horizontal motion, and 2 for vertical motion
   (c) `dio` is a character string, typed as is.

   Note: The horizontal motor is located at the back of the assembly and the vertical motor #2 is at its top. Thus, in the horizontal motor #1

   - `direction`=1 $\Longleftrightarrow$ motion *toward the motor* $\Longleftrightarrow$ *motion to the back of the assembly*
   - `direction`=0 $\Longleftrightarrow$ motion *away from the motor* $\Longleftrightarrow$ *motion to the front of the assembly*

   Likewise, in the vertical motor #2

   - `direction`=1 $\Longleftrightarrow$ motion *toward the motor* $\Longleftrightarrow$ *upward motion*
   - `direction`=0 $\Longleftrightarrow$ motion *away from the motor* $\Longleftrightarrow$ *downward motion*

   Example: `direct(0,1,dio)` commands the next horizontal motion in Motor #1 to be to the front (direction=20).
3. Motion actuation: `move(stepsize,motor #,dio)` where
   (a) `stepsize` indicates the step size in centimeters (we shall only use integer multiples)
   (b) `motor #` stands for the number of the motor, as above
   (c) `dio`, as above.

   Example: `move(3,2,dio)` commands a 3 cm vertical motion

Important Points:

- The total safe range of motion in each direction (horizontal and vertical) is about 10 cm.
- The positioner is protected in the sense that if one of the motors reaches the end of its range, it is prevented from further motion. However, our experience is that it can be mechanically stuck and we shall therefore do our best to avoid hitting the protectors!
- For ease of reference we shall select a front-most, bottom-most possible position and declare it the as the (0,0) position of the pointer. Facing the positioner, $x = 0$ is also the left-most position. Thus the range of admissible positions will occupy the square of points whose coordinates are $\{(x, y), : 0 \leq x,\ y\ \leq 10\}$ in the positive quadrant of the plane, as depicted in Figure 1.

Lab Experiment 1. Bring the Pointer to the (0,0) Position: To bring the pointer to the (0,0) position you have to send it downwards and then leftwards, as far as possible. Since the initial position is unknown, you will command each motion with estimated increments, to avoid hitting the boundaries of allowed motion. To stop the motor in case of emergency hit the "`control`" and "`c`" keys. Include a brief description of this experiments and a printout of the relevant portion of the command window in your lab report.

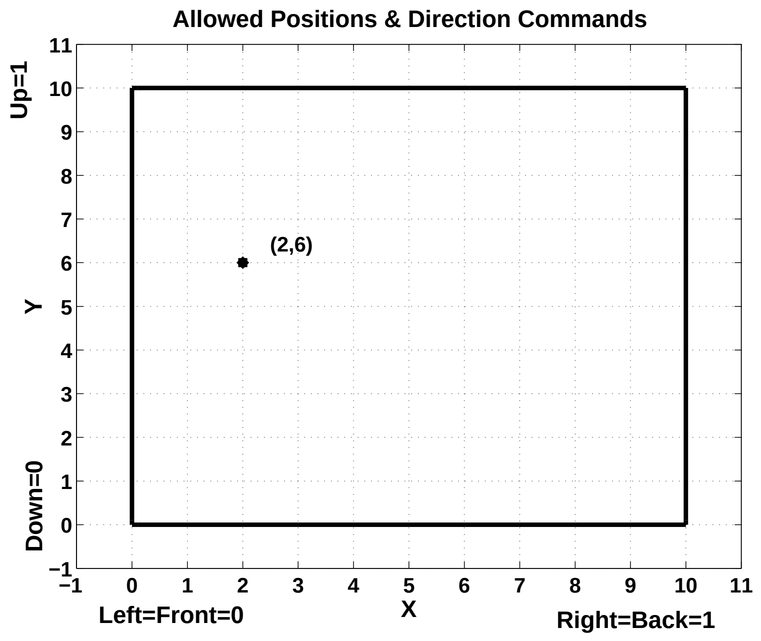

Figure 1: The square of safe positioner location is identified with the square $0 \leq x, y \leq 10$ in the positive quadrant. As an example, the black dot (●) indicates the position $x = 2$, $y = 6$. The values of `direction` for horizontal and vertical motion are indicated along the two axes.

4.2 Etch-a-Sketch

This part includes two planar motion planning tasks.

Pre-Lab Task 5. Here you will use the felt-tip pen attached to the positioner to draw the boundaries of the square of possible positions of the pen's tip, as depicted in Figure 1. In this Pre-Lab you have to write the program, save it on your memory stick as an M-file named `square.m` and include a copy of the program in your Pre-Lab report. It is assumed that at the time the program is executed the pan is located at the (0,0) position (i.e., $x = 0$ and $y = 0$). Your program should therefore command the motor to traverse in the clockwise direction the entire perimeter of the square whose vertices are the points (0,0), (0,10), (10,10) and (10,0). Recall: traversing each side of the square requires two commands:

1. A `direct` command determining the next direction of motion (towards the motor or away from the motor) in the relevant motor (#1 horizontal and #2 vertical). Since you cannot see the actual system when you prepare this program, you may consult Figure 1 for the appropriate values of the `direction` parameter.
2. A `move(10,motor #,dio)` command, executing a 10 cm move in the appropriate motor.

Lab Experiment 2. Execute and debug the program `square.m`. Include the debugged program, a printout of the relevant portion of the command window and your actual felt-tip pen plot, signed by the TA and marked by your name, in your lab report. Each team should prepare one plot and members may use a photo copy, indicating who has the original. Make sure that both the pen holder, the pen, and the clip-board assembly with several sheets of papers, are connected properly. For good performance, the pen should touch the paper but not overly-press against it.

Pre-Lab Task 6. The purpose of this task is to write a program that draws the letter F, using the positioner system with the attached felt-tip pen. The pre-lab task is the preparation of the necessary motion control program, named `drawF.m`. In preparing this program, note

1. Your starting point must be at the top-left (10,0) position.
2. The letter should be 6 cm tall and its top should be 3 cm wide.

3. Plan first the points where the plotter starts / change direction / ends
4. In your plan, take into consideration the fact that the actual plot, on the paper, is the mirror image of the path followed by the positioner.

Lab Experiment 3. Execute and debug the program `drawF.m`. Include the debugged program, a printout of the relevant portion of the command window and your actual felt-tip pen plot, signed by the TA and marked by your name, in your lab report. Each team should prepare one plot and members may use a photo copy, indicating who has the original. Make sure that both the pen holder, the pen, and the clip-board assembly with several sheets of papers, are connected properly. Again, for good performance, the pen should touch the paper but not overly-press against it.

4.3 Raster Motion

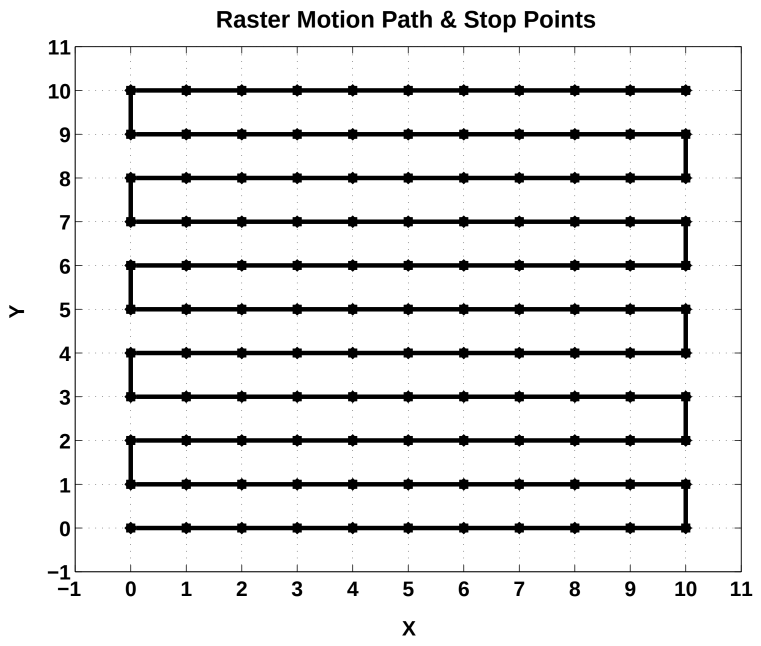

Figure 2: The raster motion path begins at the (0,0) position and visits the marked points (●) along the marked path.

The main and most sophisticated part of this Lab is the preparation of a program to control raster motion of the transducer. Following is the necessary background. As mentioned earlier, the eventual use of the positioner will be as part of a scanning imaging device. A grid of equally spaced points will cover the 10cm x 10cm square of admissible positions of the underwater, ultrasound transducer. The transducer will be moved from one grid point to the next. At each point it will be used to measure the straight-line distance to a target object, in front of it. As a first, preparatory step, the purpose of this task is to create the program needed to control the transducer's motion between the assigned grid points.

The grid is depicted in Figure 2. It comprises of 11 horizontal rows and 11 vertical columns, 1cm apart (121 grid points, in all): counting the bottom row and the left column, respectively, as first, the grid point at the intersection of the k^{th} row and the m^{th} column will have the coordinates x=(m-1)cm, and y=(k-1)cm. (Thus, for example, the x coordinate of all grid points in the 1st column is 0.)

An efficient way to move through all grid points is to scan through the rows at alternative directions, along the path depicted in Figure 2: Starting at the bottom left (the point (0,0)), the transducer will move to the right, from one point to the next, to the end of the 1st (bottom) row. It will then move 1cm up to the right-most point in the 2nd row, then continue all the way to the left, in that row, then move up to the 3rd row, and so forth. This motion is known as *raster motion*, and is used in numerous engineering application, where scanning a rectangular domain by a single point is needed. An example is the motion of the electron beam in a standard television or computer screen.

Pre-Lab Task 7. Your task here is to create a program that executes the planned raster motion. As a substitute to acoustic distance measurement it will stop and issue a screen display of the coordinates of each visited grid point. The program should be constructed as follows:

1. It should comprise of two, nested for loops:
   - The inner loop will be responsible for horizontal motion, through grid points in each row. The inner loop counter will be m=1:11 thus going through all 11 columns, in each row).
   - The outer loop will be responsible for vertical motion, from one row to the next. The outer loop counter will be k=1:11 thus going through all 11 rows).
2. Each cycle of the outer loop will execute the following series of tasks:
   (a) The centimeter value of the y coordinate of the k^{th} row is computed: `y=k-1`; In the eventual imaging application, this information will be used to properly store and process the distance measurements to the target object.
   (b) The program should determine the current value of k is even or odd: assign a variable value `odd=1` in odd rows (1st, 3rd, 5th rows, etc.), and `odd=0` in even rows. The method to determine whether a number is even or odd was developed in the Pre-Lab Task 4.
   (c) Issue the direction of horizontal motion in the k^{th} row. That direction would be to the right (i.e., towards the motor) in rows with an odd index, and to the left, in rows with an even index. Using the assigned value of the variable `odd`, the command will be `direct(odd,1,dio))`.
   (d) The inner loop will be executed, as detailed below.
   (e) If k<11, (meaning that the current row is not the top row) the direct and move commands for an upwards vertical motion of 1cm will be issued. Thus, at the end of each cycle of the outer loop, the positioner will move up one row. The upward motion command is not needed at the end of the 11th cycle, because the 11th row of grid points is the highest for the planned motion.
3. Each cycle of the inner loop will include:
   (a) The value in centimeters of the `x` coordinate of the visited point will be computed: it is `x=m-1`, if k is odd (since then m=1 at the left end of the row, where `x=0`, and `m=11` at the right end of the row, where `x=10`), and it is `x=11-m` if k is even (since then `m=1` at the right end of the row, where `x=10`, and `m=11` at the left end of the row, where `x=0`). You can use an *if* construct and the variable `odd` to determine the appropriate value of `x`.
   (b) The value in centimeters of the `y` coordinate is `y=k+1`.
   (c) Instead of the actual signal analysis commands that will be used in imaging, you will now include the following message display command

   ```
   fprintf('grid point (%d,%d) is visited\n',[x,y])
   ```

   where `x` and `y` are the computed coordinates of the visited grid point. The floating point decimal format `%d` was selected for efficient display of the integer coordinate values, without the decimal point. The same effect would have been achieved with the `%.0f` format.
   (d) If m<11, the inner loop cycle will conclude with the move command needed for a 1cm horizontal motion. (Notice that the horizontal motion command is not needed at the 11th point, which is the last in each row.)

You have to prepare an M-file named `raster.m`, containing the nested loop described above. A copy of the program should be included in your pre-lab report and in your memory stick.

Lab Experiment 4. Debug and execute the program `raster.m` in the Etch-a-Sketch setting. Your Lab report should include the drawing made by the felt-tip pen, while the program was executed, signed by the TA (one drawing per-team:

the original should be include in one lab report and other team members will use a Xerox copy, indicating who submitted the original. Your report should also include a printout of the command window output from a successful run.

4.4 Extensions and Challenges

Extension. Write an interface program that, with the pen starting at (0,0), will do the following steps:

1. Prompts the user for which figure (”F”, ”H”, ”E”, or raster) is to be drawn
2. Moves the pen to draw an ”F”, ”H”, ”E”, or raster scan.
3. Stops and prompts the user to remove the pen
4. Moves the pen back to (0,0)
5. Prompts the user for the next figure to be drawn as in Step 1. The program should end if the user enters ”9”.

Challenges. Do the following Challenge Problems as time allows:

1. Write and demonstrate for the instructor or TA an m-file that will draw a letter ”A” or ”N” (with a diagonal line). Add this m-file to your interface program above.
2. Write and demonstrate for the instructor or TA an m-file that will draw a circle with a user-defined radius. Add this m-file to your interface program.
3. Write and demonstrate for the instructor or TA an m-file that will draw a spriral with a user-defined radius and pitch. Pitch is defined as the percent that the spiral radius changes each revolution. Add this m-file to your interface program.