GE U111 HTT&TL, Lab 2

Motion Control of a Rotational Stepper Motor,

Logical Branching & Looping in MATLAB

Contents

1 Overview: Programming & Experiment Goals

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

- looping (”for” and ”while”)
- logical conditions
- further use of script M-files

The driving application is programmable motion control of a stepper motor. Specific tasks include

- estimating the arc covered in a single step from imprecise measurements
- a priori bounds over measurement and estimate uncertainties
- open loop planned motion
- retrieval of sensor data
- feedback: closed loop control of planned motion

The Components & Equipment used in this experiment include:

- A control box
- A stepper motor on a mount
- A flat flag, mounted on the motor shaft,
- A protractor on the face of the mount, indicating the rotation angle of the flag
- Two photo resistor cells, at the 0° and 90° positions
- Two light emitting diodes (LEDs), mounted near the respective photo resistors
- MATLAB software on a PC, including the Instrument Control & Data Acquisition Tool Boxes.

Experiments Outline The experiments involve a computer controlled rotational stepper motor. The motor receives two types of commands from the computer (which are translated to voltage inputs in the control box):

1. A step command results with the shaft turning over a fixed arc length. The step arc length varies from motor to motor, from a few degrees (you will determine how many) in this experiment, to a small fraction of a single degree, in precision positioners.
2. A direction command, determining whether subsequent steps motions are executed in the clockwise or counter-clockwise directions.

The motor is also equipped with two photo-resistors, used as light sensors: when the photo-resistor is exposed to light its resistance is lower than when it is covered. The computer is capable of obtaining sensor readings. Thus the computer can tell whether the motor shaft is at the 0° or the 90° positions, since there the flag is covering the respective sensors.

The experimental tasks in Lab 2 include:

- **Estimating the single step angle:** Since the flag does not provide a precise reading of the shaft’s rotation angle, a single step experiment is insufficient. You will design a multi-step experiment to obtain an accurate estimate, as well as an upper bound over the residual error. This type of experimental estimates of instrument properties and the search for a priori error bounds are prevalent in engineering applications.
- **Open and Closed Loop Motion Planning:** Motion planning is a principal ingredient in engineering applications such as robotics and computer controlled machine tools (e.g., a programmable lathe). In *open loop* motion planning one relies on machine precision and program the number of steps and direction changes that comprise the desired path. In *closed loop* motion planning sensors are used to determine the motor position, to correct for the accumulation of small errors in executed motion. Here you will utilize the available photosensors.

In order to be able to execute these tasks we shall have to employ the important programming tools of *logical branching* and *looping*.

2 Homework Assignments

- Pre-Lab Homework
  - Read this handout carefully.
  - Read Chapter 7 in the textbook
  - Optional: Read Section 4.2-4.3 (pp.87-93)
  - Do Problems 1, 5, 7(a), 10, 11, on pages 227 – 229; include printout of the MATLAB command window with your answers.
  - Prepare an M-Files to perform all the tasks listed as **Pre-Lab Programming**, along this handout.
    Pre-Lab Programming tasks include
    - Including a printout of your programs in the Pre-Lab Report that is submitted in the first session of the lab, and
    - Bringing to the lab a floppy diskette 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 experiment is finished.

3 The Experiment

3.1 Computer-Controlled Stepper Motor Operation & Sensor Thresholds

The computer is connected to various instruments through input and output (I/O) ports. Those are used to receive information from the stepper motor and related devices, and to issue commands to the motor, through its control box. Motion actuation and data acquisition by use of MATLAB require some detailed MATLAB programs that are already stored as M-Files in the lab computers. You do not need to access these detailed programs directly (nor should you attempt to modify them!). For the purpose of our lab, it suffices to accept the name of each program as a MATLAB command. Once you type the name of the program, it is executed to achieve the desired action. The following description concerns motion actuation and related commands.

Important: On occasion you will need to rotate the stepper motor shaft manually. To be able to do so you have to first switch the control box off. (Use the single on/off switch on the control box.) Remember to switch the control box "on" again when you finish such manual rotation!
★ The command `setup_rot` initiates the necessary programs for interactions with the motor. You should issue this command in the command window **once** in each session. (Repeated execution may result with memory problems!)

★ The command `onerot` executes the motion actuation signal. In our stepper motor it is a short signal of 5 Volts, received at the control box step port. Each time `onerot` is executed, such an impulse is received and the motor rotates a fixed angle, which we call a "step". Motion over longer angles is achieved via a succession of steps.

★ The direction signal may take two possible voltage values: Following the execution of the command `cwrot`, a constant 10V signal is sent to the direction input port in the motor control box. Future actuation signals will then cause the motor to rotate clockwise. Following the execution of the command `ccrot`, the direction port voltage is set to zero volts and future actuation signals will cause the motor to rotate counterclockwise. The direction voltage is kept constant, until a new command is issued, or until the system is disconnected. The commands `cwrot` and `ccrot` do not cause any motion by themselves.

★ The *function* `readcell(·)` reads the voltage across the photo-resistors: `readcell(1)` reads sensor #1 and `readcell(2)` reads sensor #2. You will have to determine, experimentally, which of the two sensors is #1 and which is #2.

EXAMPLE: To assign the sensor #1 reading to a MATLAB variable named `sensor_value` one issues the MATLAB command

```
sensor_value=readcell(1);
```

Note Again: The commands `setup_rot, onerot, cwrot, ccrot` and `readcell(.)` require special software and are **not** recognized by MATLAB, other than on HTT&TL computers. To test programs using these and other HTT&TL special command on other computers you have to comment these commands (using %) and substitute the by "*dummy*" display commands. For example, to debug a motion control program, temporarily replace "`cwrot`" by "`disp('next motion clockwise')`. However, the programs you prepare in your Pre-Lab assignment must contain the original HTT&TL commands.

Experiment 1: Test the basic motor functions, using the command window.

1. Execute (once!) the command `setup_rot`.
2. Type in the command window the *for* loop

```
for k=1:10
onerot;
end
```

Watch the motor while you hit the `return` key to execute the loop. You or your team mate may lightly touch the shaft of the motor (**NOT** the flag!) whiel the program executes. Record your observation.

3. Type `ccrot` and repeat the previous step. Record your observation.
4. Type `cwrot` and repeat the previous step. Record your observation.
5. Switch off the control box; bring the flag to a position between the two sensors (say, at $45°$). Execute the functions `readcell(1)` and `readcell(2)` and record in your notebook the returned values.
6. Repeat the previous step when the $0°$ sensor is covered.
7. Repeat the previous step when the $90°$ sensor is covered.
8. Based on the previous 3 experiments, determine
   (a) What are the respective positions of sensor #1 and sensor #2
   (b) Determine what you consider as a good threshold such that
   - `readcell(1)>threshold` implies that sensor #1 is covered
   - `readcell(1)<threshold` implies that sensor #1 is exposed

   Is the same threshold good for sensor #2?

Pre-Lab Task 1. Consider the following program and explain in detail what should be the response of the motor

```
cwrot
while readcell(1)<threshold
   onerot
end
```

Save this program in an M-file named `move_a_while.m`

Lab Experiment 2. The purpose of this experiment is to test your threshold and practice the use of *while* loops.

1. Switch the control box off, bring the flag near the $180°$ position and switch the control box on again.
2. Type `cwrot` in the command window
3. Execute `move_a_while.m`
4. Repeat step 1, above, and type `ccrot` in the command window.
5. Execute `move_a_while.m`
6. Record your observations

3.2 Determination of A Single Step Size

The purpose of this experiment is to determine the angle (in degrees) travelled by the flag in a single step.

The Challenge: Since the flag is not pointed, angle reading includes an error of up to $2°$:

```
measured angle = real angle + error
```

Execution of a single step will thus yield

```
estimated step  = measured terminal angle - measured initial angle
                = real terminal angle + error2 - real initial angle - error1
```

and the total error may be up to **twice** the original error bound.

The Remedy: If instead of a single step, a succession of `n` steps is executed, the single step estimate becomes

$$\begin{aligned} \texttt{estimated step} &= \frac{\texttt{measured terminal angle - measured initial angle}}{\texttt{n}} \\ &= \frac{\texttt{real terminal angle - real initial angle + error2 - error1}}{\texttt{n}} \\ &= \texttt{real step size} + \frac{\texttt{error2 - error1}}{\texttt{n}} \end{aligned} \tag{1}$$

The advantage is the here the total measurement error (`error1 - error2`) does not increase, but its effect on the `estimated step` is reduced by a factor of `n`.

Pre-Lab Task 2. Create a program named step_angle_estimate.m. This program should

1. Uses the `input` command (with a screen display of the request, see page 28-29 in the book) to request input values for `n` and `initial_angle`.
2. Executes a *for* loop that commands `n` counterclockwise steps (i.e., in the angle growth direction).
3. Uses the `input` command to request a value for `terminal_angle`.
4. Uses Equation (1) to estimate the step angle, and assign it to a variable named `step_angle`.
5. Uses either `disp` or `fprintf` command to display the estimated step angle.

Lab Experiment 3.

1. Visually estimate an upper bound (in degrees) over the `measured angle` error.
2. Using Equation (1), compute how many steps are needed (i.e. how large should `n` be) in order to guarantee that the step estimation error will be less than $0.01°$.
3. Estimate the step angle:
   (a) Estimate the current angle of the flag.
   (b) Execute step_angle_estimate.m with `n` as determined in part 2.
   (c) Input the values for `n` and `initial_angle`.
   (d) Watch the motor as your program executes, and count how many full rotations it completes before it stops.
   (e) Estimate the terminal angle of the flag once the motor stops. Remember to add $360°$ for each complete circle (i.e., each time the flag passes the original angle)! For example, if the starting angle estimate is $36°$ and the flag completes two full circles and stops at $275°$, the ending angle is $2 \times 360° + 275° = 975°$.
   (f) Input the value of `terminal_angle`.

3.3 Open Loop Motion Planning

Here the goal is to make the motor go through a succession of pre-assigned angle motions, using *for* loops.

Pre-Lab Task 3. Create a program named open_loop.m to perform the following tasks

1. Rotate clockwise an arc of[1] $70°$.
2. Display the statement `Arc No.  1 Completed`

[1] If the assigned arc length is not an integer multiple of a single step angle, round the necessary number of steps to the nearest integer. You may use MATLAB's `round` commnd (see MATLAB help).

3. Rotate counterclockwise $1435°$.
4. Display the statement `Arc No.  2 Completed`
5. Rotate clockwise $25°$.
6. Display the statement `Mission Completed`

Lab Experiment 4. Debug and execute the program `open_loop.m`. Include a printout of the relevant portion of the command window in your lab report.

3.4 Closed Loop Motion Planning

Here the goal is to make the motor go through a succession of pre-assigned angle motions, using sensor measurement feedback and both *for* and *while* loops.

Pre-Lab Task 3. Create a program named closed_loop.m to perform the following tasks

1. Uses the `input` command to obtain by user input the values of `threshold`, `num0sensor` = the number (1 or 2) of the photo resistor at the $0°$ position, and of `num90sensor` = the number (1 or 2) of the photo resistor at the $90°$ position. These variables will be used in using `readcell(num...)` and in logical conditions in what follows.
2. Rotate clockwise till the $0°$ position is reached.
3. Display the statement `Arc No.  1 Completed`
4. Rotate counterclockwise till the $90°$ position is reached.
5. Display the statement `Arc No.  2 Completed`
6. Continue an additional counterclockwise $425°$.
7. Display the statement `Arc No.  3 Completed`
8. Rotate clockwise till he $0°$ position is reached.
9. Display the statement `Mission Completed`

Lab Experiment 5. Debug and execute the program `closed_loop.m`. Include a printout of the relevant portion of the command window in your lab report.