Hands-on Lab

Lego Programming – BricxCC Timers

Timing is important in controller design; data and actions often must be respectively acquired and commanded at prescribed intervals. NxC provides statements for millisecond timers, and when combined with file handling, programs for system identification become possible. Hands-on exercises will culminate in in capturing the step response of an NXT motor.

Preamble: `#include` statement

Previously the file `displaySquareAndSquareRoot2_0.nxc` iterated numbers from 1 to 10, computed their squares and square roots. These computations were saved into a file, which could be opened with programs like Excel. Recall file handling involved 3 functions that we composed:

1. `void InitWriteToFile()`allocates memory to save data into. This function does not take any variables, nor does it return any (i.e. `void`). It creates the filename and writes a header to the file e.g. `x, x^2, sqrt(x)`.
2. `void WriteToFile(string someStringVariable)` writes the string variable `someStringVariable`. This function does not return any variable (i.e. `void`). Recall, `someStringVariable` will contain the alphanumeric representation of numeric data (like integers and floats).
3. `void StopWriteToFile()`gracefully closes the file. This function does not take any variables, nor does it return any (i.e. `void`).

These 3 functions allowed one to save any data into a file. Note, because `WriteToFile` wrote data as a string (i.e. collection of alphanumeric characters), one can open the file with an ASCII editor (i.e. plain-text) like Windows Notepad. Also, the file extension was saved as a CSV (comma separated values) which programs like Excel easily recognize.

File-related functions play an important role in future programs and will be used often. It would save time and be more efficient if one could simple re-use such functions (i.e. write it once and use them whenever needed). This time savings is achieved by the ANSI C statement **`#include`**. One only needs to save the functions in a file with the extension .h (called a header file).

Step 1: Click `File – Open` and load `displaySquareAndSquareRoot2_0.nxc.` Click `File – Save As` with the name "`displaySquareAndSquareRoot3_0.nxc`".

Step 2: Copy all file-saving related functions into a new file called `fileSavingFunctions.h`. Save it the same directory as `displaySquareAndSquareRoot3_0.nxc.`

Step 3: Delete the file-saving functions from `displaySquareAndSquareRoot3_0.nxc` and add the line **`#include "fileSavingFunctions.h"`**.

Step 4: Compile and run from `displaySquareAndSquareRoot3_0.nxc.` This program should now run like `displaySquareAndSquareRoot2_0.nxc.`

```
// File: displaySquareAndSquareRoot3_0.nxc
// Date: 10/14/2012 14:47
// Desc: Display number, its square and square root save to file
// Vers: 3.0 (with file saving functions as include file)
// Refs: displaySquareAndSquareRoot2_0.nxc

#include "fileSavingFunctions.h"

task main ()
{
  int x; // integers from 1 to 10
  int xSquared; // square of x
  float xSquareRoot; // square root of x

  string strX;
  string strXSquared;
  string strXSquareRoot;

  // Create a new file that captures time and motor speed
  InitWriteToFile();

  for (x = 1; x <=10; x++) {
    xSquared = x*x;
    xSquareRoot = sqrt(x);

    // TextOut (xPosition, yPosition, string) put string on LCD's x,y position
    // NB: x = y = 0 is lower left corner of LCD; +x goes rights, +y goes up
    // FormatNum is a string with sprintf syntax

    TextOut (10, LCD_LINE4, FormatNum("x = %d" , x));
    TextOut (10, LCD_LINE5, FormatNum("xSquared = %d" , xSquared));
    TextOut (10, LCD_LINE6, FormatNum("sqrt(x) = %3.3f" , xSquareRoot));
    Wait (SEC_2);

    // Create string version of calculated values
    strX = FormatNum("%d" , x);
    strXSquared = FormatNum("%d" , xSquared);
    strXSquareRoot = FormatNum("%3.3f" , xSquareRoot);

    // Concatenate the 3 strings into a single one.
    // Write resulting string to file. The text will be end with a EOL
    text=StrCat(strX, "," , strXSquared, "," , strXSquareRoot, "," );
    WriteToFile(text);
  } // end of for loop

  // Finished computing square and square root, so clean up and quit
  ClearScreen();
  TextOut(0, LCD_LINE2, "Quitting", false);
  StopWriteToFile();
  PlaySound(SOUND_LOW_BEEP);  // Beep to signal quitting
  Wait(SEC_2);

} // end of main
```

New Program: `displaySquareAndSquareRoot3.0.nxc`

```
// File: fileSavingFunctions.h
// Note: Save this .h file in same directory as the file that includes it
// Date: 10/14/12 14:48
// Auth: P.Oh
// Desc: Header file for 3 functions related to file handling

// Global variables (for file writing)
unsigned int result; // flag returned when handling files
byte fileHandle; // handle to the data file
short bytesWritten; // number of bytes written to the file
string fileHeader; // column header for data in the file
int fileNumber, filePart; // integers to split up data file names
string fileName; // name of the file
string strFileNumber; // file number e.g myDataFile 1, 2, 3
string strFilePart; // file part e.g. myDataFile1-1, 1-2, 1-3
string text; // string to be written to file i.e. data values

// Create and initialize a file
void InitWriteToFile() {
  fileNumber = 0;  // set first data file to be zero
  filePart = 0; // set first part of first data file to zero
  fileName = "squareData.csv" ; // name of data file
  result=CreateFile(fileName, 1024, fileHandle);
  // NXT Guide Section 9.100 pg. 1812 and Section 6.59.2.2 pg. 535
  // returns file handle (unsigned int)

  // check if the file already exists
  while (result==LDR_FILEEXISTS) // LDR_FILEEXISTS returns if file pre-exists
     {
      CloseFile(fileHandle);
      fileNumber = fileNumber + 1; // create new file if already exists
      fileName=NumToStr(fileNumber);
      fileName=StrCat("squareData" , fileName, ".csv");
      result=CreateFile(fileName, 1024, fileHandle);
      } // end while

  // play a tone every time a file is created
  PlayTone(TONE_B7, 5);
  fileHeader = "x, x^2, sqrt(x)" ; // header for myData file
  WriteLnString(fileHandle, fileHeader, bytesWritten);
  // NXT Guide Section 6.59.2.43 pg. 554
  // Write string and new line to a file
  // bytesWritten is an unsigned int.  Its value is # of bytes written

} // end InitWriteToFile

void WriteToFile(string strTempText) {
  // strTempText stores the text (i.e. ticks and motorRpm to be written to file

  // write string to file
  result=WriteLnString(fileHandle, strTempText, bytesWritten);
  // if the end of file is reached, close the file and create a new part
  if (result==LDR_EOFEXPECTED) // LDR_EOFEXPECTED is flagged when end-of-file
    {
    // close the current file
    CloseFile(fileHandle);
    // NXT Guide Section 6.59.2.1 pg. 535
    // Closes file associated with file handle

    // create the next file name
    filePart = filePart + 1;
    strFileNumber = NumToStr(fileNumber);
    strFilePart = NumToStr(filePart);
    fileName = StrCat("squareData" , strFileNumber,"-", strFilePart ,".csv");

    // delete the file if it exists
    DeleteFile(fileName);
    // NXT Guide Section 6.59.2.5 pg. 537
    // Delete the file specified by the string input
```

New Header File: `fileSavingFunctions.h`

```
    // create a new file
    CreateFile(fileName, 1024, fileHandle);
    // play a tone every time a file is created
    PlayTone(TONE_B7, 5);
    WriteLnString(fileHandle, strTempText, bytesWritten);
    } // end if
} // end WriteToFile

// Close the file
void StopWriteToFile() {
  // close the file
  CloseFile(fileHandle);
} // end StopWriteToFile
```

New Header File (continued from previous page): `fileSavingFunctions.h`

Concept 1 – Timers:

In lecture, a DC motor with a voltage input and velocity output is modeled as a 2nd-order system. For motors with a low inductance, like the NXT one, modeling can be simplified into a first-order system. A program will be written that commands the NXT motor with a step input. The motor's velocity is then captured and plotted in Excel. The resulting curve will illustrate a typical first-order response, from which one can determine motor characteristics like rise-time.

Step 1: Open BricxCC, click `File – New` and save your program as `nxtMotorOlsr1_0.nxc`. Write the following code.

```
// File: nxtMotorOlsr1_0.nxc
// Auth: P.Oh
// Date: 10/14/12 15:33 - works!
// Vers: 1.0: Uses MotorRotationCount which reports encoder count in degrees
//            and program calculates difference over delta tic counts
// Desc: NXT motor on Port A, save open-loop step response data to file

#include "fileSavingFunctions.h"

#define MOTOR OUT_A // set constant MOTOR for Port A
#define FULL_SPEED 75 // 75 percent of possible motor speed
#define DEG2RPM 166.667 // deg/msec to RPM
#define RPM2RADPERSEC 0.105; // RPM to rad/sec

task main() {

  // Motor related variables
  long prevAngleInDegrees; // placeholder for degree read by motor encoder
  long curAngleInDegrees; // current motor angle [DEG]
  long deltaAngleInDegrees; // change in motor angle [DEG]
  string strDeltaAngleInDegrees; // string form of deltaAngleInDegrees
  float motorRpm; // motor speed [RPM]
  string strMotorRpm; // store integer value of motorRpm as string
  float motorRadPerSec; // motor speed [rad/s]
  string strMotorRadPerSec; // string form of motorRadPerSec

// Timing related variables
  long prevTick;
  long curTick; // current timer value
  long deltaT; // For calculating time between ticks
  string strDeltaT; // string form of deltaT
  float elapsedTimeInSeconds; // time in seconds
  string strElapsedTimeInSeconds; // string form of elapsed time
```

Program: `nxtMotorOlsr1_0.nxc`

Code description: `task main` begins by declaring motor related variables. As one will see later, the NxC function `MotorRotationCount(MOTOR)` will be used. This function reports the position (not the velocity) of the motor connected to the Brick's `MOTOR` port (which happens to be Port A). Also, time related variables are declared. As one will encounter later, the NxC function `CurrentTick()` will be used to poll the Brick's current clock (called a tick counter). Like a stopwatch, variables `curTick` and `prevTick` are used to calculate the time that has elapsed and store the resulting difference in the variable `elapsedTimeInSeconds`.

Step 2: Continue adding code to your program `nxtMotorOlsr1_0.nxc`

```
// Button related variables
  bool orangeButtonPushed, rightArrowButtonPushed;

  // Create a new file that captures time and motor speed
  InitWriteToFile();

  // Initialize variables
  elapsedTimeInSeconds = 0.0; // set elapsed time to zero
  prevAngleInDegrees = 0; // motor initially motionless so set angle to zero

  // Prompt user to begin step input
  TextOut (0, LCD_LINE1, "Right Btn starts" );
  do { // wait until user hits right button
   rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
  } while(!rightArrowButtonPushed);

// Begin step response
  prevTick = CurrentTick();
  TextOut (0, LCD_LINE1, "Orange Btn quits" );
  // Command motor to move i.e. step input
  OnFwd(MOTOR, FULL_SPEED); // turn on motor at 100% full speed
```

Program (continued from previous page): `nxtMotorOlsr1_0.nxc`

Code description: The program declares two button related Boolean variables: `rightArrowButtonPushed` and `orangeButtonPushed`. These variables are used to detect which buttons the user pushes on the Brick. The code above then initializes a new file by calling the `InitWriteToFile()` function where data (like time and motor velocity) will be saved.

We will perform a step response on the NXT motor. As such, we initialize the motor's position at 0 degrees (`prevAngleInDegrees = 0;`) and say that 0 seconds has elapsed (`elapsedTimeInSeconds = 0.0;`).

The program then displays a message on the Brick's LCD and enters a do-while loop that simply polls for the right button to be pressed with NxC's `ButtonPressed` function.

Once the user pushed the right arrow button, the Brick's internal counter is polled (i.e. the stopwatch is pushed to start) with the `prevTick = CurrentTick();` statement. The LCD displays a message to prompt the user to hit the orange button to terminate the program. The program then commands the NXT motor with the NxC function `OnFwd(MOTOR, FULL_SPEED);`

Step 3: Continue adding code to calculate motor velocity and elapsed time

```
do {
   // Read change in motor angle
   curAngleInDegrees = MotorRotationCount(MOTOR); // get relative position
   deltaAngleInDegrees = curAngleInDegrees - prevAngleInDegrees;
   strDeltaAngleInDegrees = FormatNum("deltaAngle = %ld", deltaAngleInDegrees);

   // Measure elapsed time and hence motor RPM
   curTick = CurrentTick(); // read timer value
   deltaT = curTick - prevTick; // measure time elapsed between angle reads
   motorRpm = deltaAngleInDegrees * DEG2RPM / deltaT;
   strMotorRpm = FormatNum("%5.2f" , motorRpm);
   motorRadPerSec = motorRpm * RPM2RADPERSEC;
   strMotorRadPerSec = FormatNum("%5.3f" , motorRadPerSec);
   elapsedTimeInSeconds = elapsedTimeInSeconds + (deltaT/1000.0); // in sec
   strElapsedTimeInSeconds = FormatNum("%5.3f" , elapsedTimeInSeconds);

   // Display motor actual speed and elapsed time
   TextOut(0, LCD_LINE4, FormatNum("RPM = %5.2f" , motorRpm));
   TextOut(0, LCD_LINE6, FormatNum("Time = %5.3f s" , elapsedTimeInSeconds));

   // Write text data to file. The text will be end with a EOL
   text=StrCat(strElapsedTimeInSeconds, "," , strMotorRadPerSec, "," , strMotorRpm,"," );
   WriteToFile(text);

   // Update current tic value and angle
   prevTick = curTick;
   prevAngleInDegrees = curAngleInDegrees;
   Wait(100); // update loop every 100 milliseconds

   // Check if user wants to quit
   orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
  } while( !orangeButtonPushed && (FreeMemory()>=2000) );
```

Program (continued from previous page): `nxtMotorOlsr1_0.nxc`

Code description: The program begins a do-while loop which exits when the user pushed the Brick's orange button or when the Brick's memory storage runs out.

The statement `curAngleInDegrees = MotorRotationCount(MOTOR);` polls the motor's internal encoder hardware. The encoder reports values from 0 to 360 degrees and saves the value in the variable `curAngleInDegrees.` Because the encoder reports relative changes, the change in angle needs to be subtracted from the previously polled value. This is achieved through the `deltaAngleInDegrees = curAngleInDegrees - prevAngleInDegrees;` statement. For file saving purposes, the numeric data is converted into a string of alphanumeric characters using NxC's `FormatNum` function.

Unfortunately NxC does not have a function that reports motor velocities directly. As such, velocity is calculated by polling sequential motor angles and dividing by the elapsed time. The motor velocity in both radians per second and RPM are calculated and saved to file.

Before the loop iterates, the Brick's "stopwatch" and angle encoder are updated with their last polled values i.e. `prevTick = curTick; prevAngleInDegrees = curAngleInDegrees;` Finally, the program invokes the `Wait(100);` statement. This essentially sets the sampling time of the program to 100 milliseconds. The program loops back unless it detected the user pushing the orange button (or the Brick's free memory exceeds 2 KB).

Step 4: Continue adding code to complete the program

```
// Orange button pressed, so command 0 speed to motor and quit
        ClearScreen();
        TextOut(0, LCD_LINE2, "Quitting", false);
        // Stop motor
  OnFwd(MOTOR, 0);
  StopWriteToFile();
  PlaySound(SOUND_LOW_BEEP);  // Beep to signal quitting
        Wait(SEC_2);
}  // end of main
```

Program (last part): `nxtMotorOlsr1_0.nxc`

Step 5: Lastly, modify your header file `fileSavingFunctions.h`

In your `InitWriteToFile()` function define the file name to store data as well as the first row that labels your data:

```
void InitWriteToFile() {
  fileNumber = 0;  // set first data file to be zero
  filePart = 0; // set first part of first data file to zero
  fileName = "nxtMotorData.csv" ; // name of data file
  result=CreateFile(fileName, 1024, fileHandle);
  // NXT Guide Section 9.100 pg. 1812 and Section 6.59.2.2 pg. 535
  // returns file handle (unsigned int)

// check if the file already exists
  while (result==LDR_FILEEXISTS) // LDR_FILEEXISTS returns if file pre-exists
     {
      CloseFile(fileHandle);
      fileNumber = fileNumber + 1; // create new file if already exists
      fileName=NumToStr(fileNumber);
      fileName=StrCat("nxtMotorData" , fileName, ".csv");
      result=CreateFile(fileName, 1024, fileHandle);
      } // end while

  // play a tone every time a file is created
  PlayTone(TONE_B7, 5);
  fileHeader = "Time [s], Motor Speed [rad/s], Motor Speed [RPM]" ; // header
  WriteLnString(fileHandle, fileHeader, bytesWritten);
  // NXT Guide Section 6.59.2.43 pg. 554
  // Write string and new line to a file
  // bytesWritten is an unsigned int.  Its value is # of bytes written

} // end InitWriteToFile
```

In your `WriteToFile()` function, edit code so that data is saved to the desired file

```
void WriteToFile(string strTempText) {
  // strTempText stores the text (i.e. ticks and motorRpm to be written to file
:
:
:
// create the next file name
    filePart = filePart + 1;
    strFileNumber = NumToStr(fileNumber);
    strFilePart = NumToStr(filePart);
    fileName = StrCat("nxtMotorData" , strFileNumber,"-", strFilePart ,".csv");
:
   } // end if
} // end WriteToFile
```

Step 6: Save both `nxtMotorOlsr1_0.nxc` and `fileSavingFunctions.h` files. Compile and execute. Once the program runs, hit the right arrow button to start the step response. After a few seconds (say 5 seconds), hit the orange button to terminate. Open NXT Explorer to retrieve your data file and plot in Excel.

In Excel, you should have a plot that resembles **Figure 1A**.

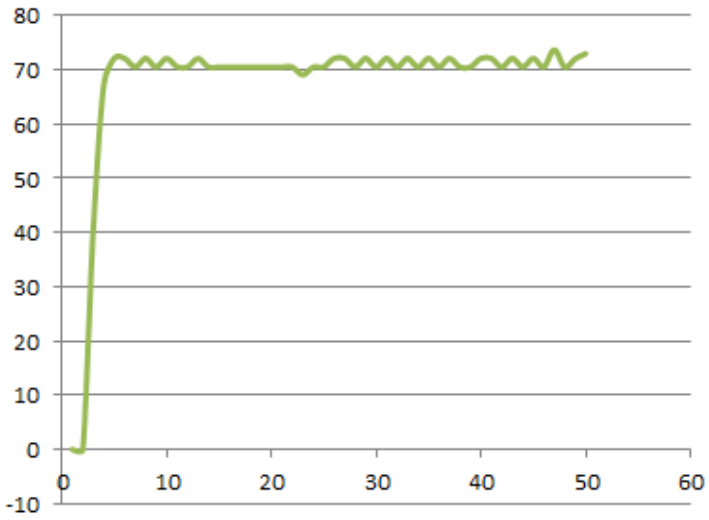

Figure 1A: Excel plot of `nxtMotorData.csv`

Exercise 1: In NxC create programs for the following:

1-1 Figure 1A is an example of an open-loop step (velocity) response to a 75% motor command. From the plot, calculate the rise time Recall that rise-time (also called time constant) is defined as 63.3% of steady-state.

1-2 Theory states for a first-order system, that at 3 time constants, the response will be within 1% of steady-state. Calculate 3 x rise time and find the velocity at the time. Verify that this velocity is within 1% of the steady-state velocity.

1-3 Write a new program that performs an open-loop step response but acquires the NXT motor's position from 0 to 5 seconds (use a 100 millisecond sample time). Plot the curve. Why does this curve loop like a ramp?