Hands-on Lab

Distributed Computing: PC to Master NXT to Slave NXT to Robot

The NXT Brick has only 1 serial port and does not have the computational power to process images. These hardware limitations are common in robotics and hence one often sees a distributed computer approach. To demonstrate this, the visual-servoing of the XL-320 2-link planar manipulator will be explored (**Figure A** left). Here, the PC uses a webcam to capture and process video and yield the robot's desired end-effector location. The PC's serial port connects to an NXT (Master). The PC then serially sends the locations in a string. The Master NXT receives and processes the string. It then wirelessly transmits via Bluetooth, these locations to another NXT (Slave). The Slave NXT is connected to the XL-320 2-link planar manipulator (Robot) via its RS-485 port. The Slave uses the Bluetooth message to calculate robot positions (either forward or inverse kinematics) and commands the XL-320 servos.

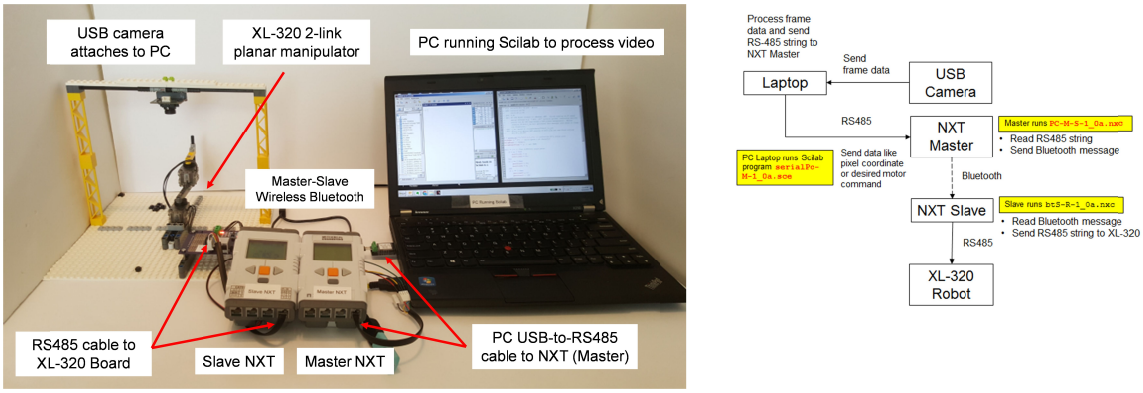

Figure A: Distributed computing setup (left) uses both serial and Bluetooth communications. The flow diagram (right) shows the process of passing information from to USB camera to XL-320 robot. Demo video: https://youtu.be/BDz0-FkbjOM

Concept 1: Establish PC-to-Master-to-Slave-to-Robot communications

Camera data is ignored to begin. Instead, the distributed computing framework first needs to be established. **Figure A** (right) shows that 3 programs need to be created.

Step 1: For the PC, write a Scilab (version 6.1.0) Serial Port Program **`serialPc-M-1_0b.sce`**

A previous lab went thru the installation of the ATOMS Serial Communications toolbox in Scilab. **Figure 1A** demonstrates how the serial port is opened, and messages are written and read.

The program begins with the toolbox function `openserial` to open and set the serial port (e.g. 4800 baud, 8N1). The for-loop goes thru 4 different position settings e.g. 0, 90; 90, 0; 0, -90; and 90, 90. Each setting represents the desired angles of the robot's two XL-320 servos. The setting's first and second numbers are respectively joint angles 1 and 2 of the 2-link planar manipulator. These numbers form part of the string that will be sent to the Master. Each number has 4 characters: the sign followed by 3 digits e.g. "`+102`" (without the quotes). If the desired angle is less than 3 digits, then a white space is used e.g. "`+90 `" (without the quotes). Serial communications often prescribes how strings are defined (e.g. with headers, comma delimiters, and formatting).

The Scilab function `strcat` is used to create a specific message. Its form is:

" @position01, position02"

Here, there is a whitespace followed by the @ character. This was assigned by the string variable `strHeader`. This header is non-unique. A header is often read by the receiver (i.e. NXT Master) to ensure the transmitted message is valid. Such validation is important when communication amongst a mix of computers.

```
// FILE: serialPc-M-1_0b.sce - Works!  Fixed 1_0a
// DATE: 04/18/20 07:40
// AUTH: P.Oh
// DESC: PC USB RS485 connect to (Master) NXT.  Scilab running on PC sends
//       serial message e.g. " @90, -90" (without quotes) containing desired
//       angles for XL-320 Lego-based 2-DOF planar manipulator.  Master
//       (running PC-M-S-1_0a.nxc) processes this message (and sends to Slave
//       via Bluetooth).
// VERS: 1_0a: based on scilabPcSerialToNxt0_1h.sce
//       1_0b: Different angles for homework
// REFS: Works with Master NXT running PC-M-S-1_0a.nxc and Slave running
//       btS-R-1_0a.nxc

h = openserial(10,"4800,n,8,1"); // initialize PC's serial port
strHeader = " @"; // white space + at character
stringRoger = "ROGER";
stringRogerFound = 1; // not TRUE

for i = 1:4 // four different angle pairs
    // 1_0a used: (0,90); (90,0); (0,-90); (90,90) - and worked!
    // 1_0b: each coordinate must be sign followed by 3 characters here, created the string
    // directly with specific format i.e. a 4 character string: sign and 3 characters the 3
    // characters are the digits or when value < 3, then substitute with white space
    if i == 1 then
        strPosition01 = "+102";  // NB1: sign and 3 digits
        strPosition02 = "+102";
    end
    if i == 2 then
        strPosition01 = "+90 "; // NB2: sign and 2 digits + white space = 3
        strPosition02 = "-90 "; // NB2: sign and 2 digits + white space = 3
    end
    if i == 3 then
        strPosition01 = "-90 "; // same as above
        strPosition02 = "+90 ";
    end
    if i == 4 then
        strPosition01 = "+45 ";
        strPosition02 = "+45 ";
    end

    strI = strcat([strHeader, strPosition01, ", ", strPosition02]);
    disp(strI);
    writeserial(h, strI); // transmit serially to Master NXT

    buf = readserial(h);
    // Check if Master ready to receive next string
    stringRogerFound = strcmp(stringRoger, buf); // 0: means identical strings
    while (stringRogerFound ~=0) // then NXT -> PC string not ROGER, so wait
       buf = readserial(h);
       stringRogerFound = strcmp(stringRoger, buf);
       sleep(200); // min about 50 ms before reading serial port again
    end; // exit reading serial port when ROGER received
    disp(buf);
    sleep(5000); // just slows down loop so user can see what's happening
end

disp("All done!");
closeserial(h)
```

Figure 1A: Scilab program **serialPc-M-1_0b.sce**

Scilab then transmits the string by calling `writeserial`. The while-loop repeatedly pings the receiver using a `readserial` call and compares any incoming string with "`ROGER`". Here, the Master NXT transmits "`ROGER`" to the PC when it received a valid message (one with a white space, followed by an @ character and numbers). Here, "`ROGER`" is not unique but makes sense as a suitable reply that Master NXT received a valid message from the PC.

Once the PC received "`ROGER`" from the Master NXT, the `for-loop` iterates to send the next set of positions.

Step 2: Master NXT NXC program for Serial and Bluetooth communications **`PC-M-S-1_0a.nxc`**

In **Figure A** (right), the Master NXT sits between the PC and Slave NXT. As such, the Master NXT must perform serial (with the PC) and Bluetooth (with the Slave) communications. In a previous lab, the NXT performed serial communications with a terminal emulator (Hercules) with the NXC program `nxtReadFromPC1_0b.nxc`. In yet another lab, Master-Slave Bluetooth communications was performed using the NXC program **`btMaster0_2a.nxc`**. The code in **Figure 1B** uses portions of both those NXC programs to serially communicate with the PC running Scilab and Bluetooth communicate with a Slave NXT.

```
// FILE: PC-M-S-1_0a.nxc - Works!
// DATE: 04/15/20 11:31
// AUTH: P.Oh
// DESC: Scilab runs serialPc-M-1_0b.sce on PC to serially send a pair of
//       angles in a string.  Master NXT (running this code) receives and
//       verifies string and extracts angles.  Master NXT then
//       sends Bluetooth message containing these angles, to Slave.  Slave NXT
//       runs btS-R-1_0a.nxc applies these angles to forward kinematics
//       and command the XL-320 servos of the Lego 2-DOF planar manipulator
// VERS: 1_0a: based on btM0_1f.nxc
//       Works with Slave (btS-R-1_0a.nxc) and PC (serialPc-M-1_0b.sce)

#include "protocol0_2a.h"

task main() {

  // Bluetooth related variables
  string stringFromSlave;         // any messages from slave
  int i;                          // dummy index
  string strMaster;               // string to be sent by Master
  string message;                 // string containing message
  string ok = "OK" ;              // OK message for Slave -> Master
  string roger = "ROGER" ;        // ROGER message for Master -> PC

  // Serial port related variables
  byte readBuffer[];              // array to store bytes received from PC
  string charsRead;               // string of ASCII characters read from PC
  int lenCharsRead;               // strlen of charsRead
  byte byteC;                     // ASCII value of character read
  int atPosition;                 // position in string of @ character
  bool atPositionFound;           // @ character found
  int commaPosition;              // position in string of , character
  string strValue01, strValue02;  // extracted numbers as strings
  float value01, value02;         // numeric values of extracted string

  // Set up NXT's serial port
  UseRS485();                              // (1) Configure S4 for RS-485
  RS485Enable();                           // (2) Activate RS-485
  RS485Uart(HS_BAUD_4800, HS_MODE_DEFAULT); // (3) Baud and default parity
  Wait(MS_1);                              // (4) Brief wait for port settings
```

Figure 1B: **`PC-M-S-1_0a.nxc`** implements serial and Bluetooth communications

```
TextOut(0, LCD_LINE1, "Master" );
mastercheck(); // check Master bluetooth connection

while(true) { // read and display strings received from PC until abort
    while(!RS485DataAvailable()) {
       // if no ASCII chars available, then do nothing
    };
    atPosition = 0;
    atPositionFound = FALSE;

    // Some character(s) is on the serial port, so read and check it
    RS485Read(readBuffer);
    // Convert bytes into ASCII string
    charsRead = ByteArrayToStr(readBuffer);
    message = "PC->M:" ;
    strcat(message, charsRead);
    TextOut(0, LCD_LINE2, message);
    lenCharsRead = strlen(charsRead);
    for(i=0; i<=lenCharsRead; i++) {
        byteC = StrIndex(charsRead, i);
        if(byteC == 64) { // 64 DEC is ASCII character for @
           atPosition = i;
           atPositionFound = TRUE;
           ClearLine(LCD_LINE5); // clear @: None message from LCD
        }; // end if
    }; // end for loop to check for @ character
    if(atPositionFound != TRUE) {
           TextOut(0, LCD_LINE5, "@: None" );
    };
    if(atPositionFound == TRUE) { // valid message received
        PlayTone(TONE_A3, 100);
        // (1) find comma position
        for(i=0; i<=lenCharsRead; i++) {
            byteC = StrIndex(charsRead, i); // StrIndex returns ASCII value
            if(byteC == 44) { // 44 DEC is ASCII is comma
                commaPosition = i;
            };
        }; // end for loop checking for comma character
        // (2) Extract first number
        strValue01 = Copy(charsRead, atPosition+1, commaPosition);
        value01 = StrToNum(strValue01);
        // (3) Extract second number.  NB: Format has 1 whitespace after comma
        strValue02 = Copy(charsRead, commaPosition+1, lenCharsRead);
        value02 = StrToNum(strValue02);
        TextOut(0, LCD_LINE3, FormatNum("deg01:%3.2f" , value01) );
        TextOut(0, LCD_LINE4, FormatNum("deg02:%3.2f" , value02) );
        Wait(200);
        // (4) Create proper string to send to Slave
        strMaster = StrCat(strValue01, strValue02);
        message = "M-->S:" ;
        strcat(message, strMaster);
        TextOut(0, LCD_LINE6, message);
        // (5) Send resulting string to Slave
        sendtoslave(strMaster);
        ResetSleepTimer(); // keep Brick awake for Bluetooth connection
        // (6) Wait until Slave says OK
        do {
           stringFromSlave = receivefromslave();
           // keep checking until slave acknowledges with "OK"
           Wait(500);
        } while(strcmp(stringFromSlave, ok) != 0);
        message = "S-->M:" ;
        strcat(message, ok);
        TextOut(0, LCD_LINE7, message);
        // (7) Tell PC ready to receive next message
        RS485Write(roger);
        message = "M->PC:" ;
        strcat(message, roger);
        TextOut(0, LCD_LINE8, message);
    }; // end if atPositionFound
```

Figure 1B continued: **PC-M-S-1_0a.nxc** implements serial and Bluetooth communications

```
      readBuffer = 0;
      Wait(5000);  // so that user can read LCD
      ClearLine(LCD_LINE8); // clear M->PC roger from LCD
      ClearLine(LCD_LINE7); // clear S->M ok from LCD
      ClearLine(LCD_LINE6); // clear M->S string from LCD

  }; // end while(true)
} // end main
```

Figure 1B continued: **PC-M-S-1_0a.nxc** implements serial and Bluetooth communications

The previous lab notes can be referenced to understand the underlying serial port and Bluetooth communications. The key and new sections in **Figure 1B** are the checking of the `@` and comma characters (see yellow-highlight). Recall, the PC transmits a set of positions as a string with a header. In **Figure 1B**, `charsRead` is the received string. A for-loop reads each character in that string using the NXC function `StrIndex`. That character is compared to the ASCII value for the `@` character. Once the `@` character is found, its position in the string `charsRead` is stored in the variable `atPosition` and the Boolean variable `atPositionFound` is set to `TRUE`.

A for-loop then goes thru steps (1) to (4) to extract the numerical values of the alphanumeric (ASCII) values in the string. Step (1) first finds the comma's position in the string. Step (2) then uses the NXC function `Copy` to extract the first alphanumeric characters (sandwiched between the `@` and comma) and uses `StrToNum` those characters into a numeric value. Likewise, Step (3) extracts the second numeric value. Step (4) then creates a string called `strMaster` that in Step (5) is transmitted to the Slave NXT via Bluetooth with `sendtoslave(strMaster)`.

The do-while loop in Step (6) implements message checking. Here, the Master waits for the Slave to reply via Bluetooth, with a string saying "`OK`". This is important for synchronization; the Master should not flood the Slave with another position setting message until the Slave is finished using that message. Once the Master NXT receives this "`OK`" message, Step (7) sends a "`ROGER`" message to the PC serially with `RS485Write(roger)`.

Before receiving a new serial string, the Master NXT clears its serial buffer with a `readBuffer = 0` statement, waits 5 seconds (so user has some time to view LCD values) and then clears messages from the LCD.

Step 3: Slave NXT NXC program for Bluetooth and Robot control **btS-R-1_0a.nxc**

Bluetooth on a Slave NXT was performed in a previous lab with the NXC program **btSlave0_2a.nxc**. Also, NXC code that implemented forward kinematics with an XL-320 based 2-link planar manipulator was done in a previous lab with **xl320-2dof-fk-1_0.nxc**. These are incorporated into code shown in **Figure 1C**.

```
// FILE: btS-R-1_0a.nxc - Works!
// DATE: 04/15/20 11:49
// AUTH: P.Oh
// DESC: Slave receives Bluetooth string from Master (running PC-M-S-1_0a.nxc).
//       Slave extracts numerical values from string.  The values are angles
//       which are fed into forward kinematics.  The result is XL-320 joint
//       commands.  The Lego-based 2-DOF planar manipulator moves to those
//       joint commands, briefly waits, and then goes back to HOME position.
//       Slave then sends OK message via Bluetooth, back to Master, and
//       waits for the next angle command from Master.
// VERS: 1_0a: based on btS0_2a.nxc
// REFS: Works with Master running PC-M-S-1_0a.nxc and PC running
//       serialPc-M-1_0b.sce.
//       extract0_1e.nxc: used to detect comma and extract numbers from string
//       xl320-2dof-fk-1_0.nxc forward kinematics
```

Figure 1C: **btS-R-1_0a.nxc** for Slave NXT connected to XL-320 2-link planar manipulator

```
#include "protocol0_2a.h"
#include "xl320-defines1_0a.h"   // XL-320 defines from Control Table
#include "xl320-functions1_0d.h" // P.Oh functions written for XL-320

#define ID_ALL_MOTORS 0XFE // 0XFE commands all XL-320 motors
#define ID_MOTOR01    0X03 // Assumes Motor 1 configured with ID = 3
#define ID_MOTOR02    0X07 // Assumes Motor 2 configured with ID = 7
#define mmPerStud     8    // 8 millimeters per LEGO stud

// Global variables
  bool  orangeButtonPushed;      // Detect Brick Center button state
  bool  rightArrowButtonPushed;  // Detect Brick right arrow button state
  bool  leftArrowButtonPushed;   // Detect Brick left arrow button state
  bool  greyButtonPushed;        // Detect Brick Grey/Abort button state

void rotateMotorAbsolutely(float angle01, float angle02) { //------------------
 // Rotates desired the two Dynamixel XL-320 motors to their desired angles
 // Assumes motor count of 512 denotes 0 degrees.  Uses right-hand rule for
 // rotational direction

  float desiredAngle01InDegrees;   // Angle Motor 1 to move to [deg]
  float desiredAngle02InDegrees;   // Angle Motor 2 to move to [deg]
  float degreesPerCount;           // Conversion 0.29 [degrees/count]
  float calculatedCount;           // Count equivalent of desired angle [count]
  int   motor01Offset;             // Motor 1's offset [count]
  float theta01InDegrees;          // Motor 1 angle [counts]
  int   theta01InCounts;           // Motor 1 angle [deg]
  int   motor02Offset;             // Motor 2's offset [count]
  float theta02InDegrees;          // Motor 2 angle [counts]
  int   theta02InCounts;           // Motor 2 angle [deg]
  string msg01, msg02;             // dummy strings to print values to screen

  motor01Offset = 512; // Set Link 1 at 0 deg (i.e. 512 counts)
  motor02Offset = 512; // Set Link 2 at 0 deg (i.e. 512 counts)

  // Note 1: Looking into horn from Top, count > 512 is CCW (i.e. +Z axis)
  // and count < 512 is CW (i.e. -Z axis)
  degreesPerCount = 0.29; // [deg/count] found from XL-320 data sheet

  ClearScreen();
  desiredAngle01InDegrees = angle01;
  theta01InCounts = motor01Offset + desiredAngle01InDegrees/degreesPerCount;
  desiredAngle02InDegrees = angle02;
  theta02InCounts = motor02Offset + desiredAngle02InDegrees/degreesPerCount;

  // Format string so displays nicely on Brick screen
  sprintf(msg01, "Goto [%3.1f, " ,desiredAngle01InDegrees);
  sprintf(msg02, "%3.1f]" , desiredAngle02InDegrees);
  TextOut(0, LCD_LINE2, strcat(msg01, msg02));

  XL320_servo(ID_MOTOR01, theta01InCounts, 200); // motor position at speed 200
  Wait(2000); // wait about 2 seconds before issuing another command
  XL320_servo(ID_MOTOR02, theta02InCounts, 200); // motor position at speed 200
  Wait(2000); // wait about 2 seconds before issuing another command
  PlayTone(TONE_B3,50);

}; // end rotateMotorAbsolutely function ---------------------------------
```

Figure 1C continued: **btS-R-1_0a.nxc**

```
task main() {

  // Bluetooth related variables
  string stringFromMaster;          // store string from Master
  int lenStringFromMaster;          // store length value of received string

  byte byteC;                       // ASCII value of character read in strData
  int i;                            // dummy counter variable
  int commaPosition;                // Position in strData of comma
  string message;                   // dummy string to display message
  string strValue01, strValue02;    // extracted numbers as strings
  float floatValue01, floatValue02; // floats of extracted string
  string strOkFromSlave = "OK" ;    // OK from slave

  // planar manipulator variables
  float l1, l2; // length of link 1 and link 2 [mm]
  float theta1, theta2; // angle of joint 1 and joint 2 [rad]
  float theta1InDegrees, theta2InDegrees; // angle of joint 1 and 2 [deg]
  float xP0, yP0; // end-effector absolute position i.e. wrt x0y0 frame [mm]
  int xP0InStuds, yP0InStuds; // [studs]

  // calculation and dummy variables
  float C, k1, k2, num, den;

  // initializations
  l1 = 7 * mmPerStud; // [mm] link 1 is 7 studs long
  l2 = 5 * mmPerStud; // [mm] link 2 is 5 studs long

  UseRS485();
  RS485Enable();
  RS485Uart(HS_BAUD_57600, HS_MODE_8N1); //57600 baud, 8bit, 1stop, no parity

  ClearScreen();
  // Prompt user to begin
  TextOut(0, LCD_LINE1, "Start: hit ->");
  do {
     rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
  } while(!rightArrowButtonPushed);
  ClearScreen();

  // First go to home position
  ClearScreen();
  TextOut(0, LCD_LINE2, "Homing..." );
  Wait(2000);
  theta1InDegrees = theta2InDegrees = 0.0;
  rotateMotorAbsolutely(theta1InDegrees, theta2InDegrees);
  Wait(2000);
  PlayTone(TONE_E4, 500);

  ClearScreen();
  slavecheck();  // initialize NXT running this program as the Slave
  TextOut(0, LCD_LINE1, "Slave" );
```

Figure 1C continued: **btS-R-1_0a.nxc**

```
  for(;;) {
    do { // keep checking of Master sent a message
      stringFromMaster = receivefrommaster();
      lenStringFromMaster = StrLen(stringFromMaster);
    } while(lenStringFromMaster == 0);

    // Now Master's message received
    message = "Rec'd: " ;
    strcat(message, stringFromMaster);
    ClearLine(LCD_LINE2); // clear any old Master's string message from LCD
    TextOut(0, LCD_LINE2, message); // display newly received message

    // (1) Find position of comma
    for(i=0; i <= lenStringFromMaster; i++) {
       byteC = StrIndex(stringFromMaster, i); // StrIndex returns ASCII value in DEC
       if(byteC == 44) { // 44 ASCII is comma
          commaPosition = i;
       }; // end if
     }; // end (1)

    // (2) Extract first number
    strValue01 = Copy(stringFromMaster, 0, commaPosition);
    // ---- message = "str1: " ;
    // ---- strcat(message, strValue01);
    theta1InDegrees = StrToNum(strValue01);
    theta1 = theta1InDegrees * PI/180; // [rad]

    // (3) Extract second number.  NB: Format has 1 whitespace after comma
    strValue02 = Copy(stringFromMaster, commaPosition+2, lenStringFromMaster);
    // --- message = "str2: " ;
    // --- strcat(message, strValue02);
    theta2InDegrees = StrToNum(strValue02);
    theta2 = theta2InDegrees * PI/180; // [rad]

    // Forward Kinematics equations yield end-effector position (xP0, yP0)
    xP0 = l1*cos(theta1) + l2*cos(theta1 + theta2); // [mm]
    yP0 = l1*sin(theta1) + l2*sin(theta1 + theta2); // [mm]
    // End-effector position in LEGO studs
    xP0InStuds = ceil(xP0 / mmPerStud); // round up [stud]
    yP0InStuds = ceil(yP0 / mmPerStud); // round up [stud]

    TextOut(0, LCD_LINE3, "Will go to:" );
    TextOut(0, LCD_LINE4, FormatNum("xP0 = %3d studs" , xP0InStuds) );
    TextOut(0, LCD_LINE5, FormatNum("xP0 = %3.3f mm", xP0) );
    TextOut(0, LCD_LINE6, FormatNum("yP0 = %3d studs" , yP0InStuds) );
    TextOut(0, LCD_LINE7, FormatNum("yP0 = %3.3f mm", yP0) );
    // Prompt user to begin motion
    TextOut(0, LCD_LINE8, "Yes: hit ->");
    do {
       rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
    } while(!rightArrowButtonPushed);
    ClearScreen();

    rotateMotorAbsolutely(theta1InDegrees, theta2InDegrees);
    Wait(2000);
    TextOut(0, LCD_LINE2, "Back to Home" );
    theta1InDegrees = theta2InDegrees = 0.0;
    rotateMotorAbsolutely(theta1InDegrees, theta2InDegrees);
    Wait(2000);
    PlaySound(SOUND_DOUBLE_BEEP);

     // (4) Tell master ready for new message
    sendtomaster(strOkFromSlave);
    ResetSleepTimer(); // don't time out and shut off Brick

  } // end for
} // end main
```

Figure 1C continued: **btS-R-1_0a.nxc**

The Slave NXT program `btS-R-1_0a.nxc` looks long. However it should also look familiar. The function `rotateMotorAbsolutely` is identical to the one used in `xl320-2dof-fk-1_0.nxc` which implemented forward kinematics on the XL-320 2-link planar manipulator and displayed the stud positions on the Brick's LCD.

The key section of code begins with the endless for-loop. After receiving the string `stringFromMaster` from the Master NXT via Bluetooth, the process of extracting angles begins. Step (1) uses `StrIndex` to search for the comma character in that string. Steps (2) and (3) extract the alphanumeric characters and converts them to numerical values with calls to `StrToNum`. These numerical values are respectively assigned to `theta1InDegrees` and `theta2InDegrees`. These values are then applied to the forward kinematics equations to calculate the robot's end-effector (x_{p0}, y_{p0}) position as well as to `rotateMotorAbsolutely` to command the XL-320 smart servos. After the 2-link planar manipulator rotates to the desired angles, it waits for 2 seconds and then rotates to the HOME position.

Before looping back, Step (4) sends an "`OK`" message to the Master NXT via Bluetooth using the call `sendtomaster(strOkFromSlave)`. Recall for synchronization, the Master NXT will not send more Bluetooth strings until it receives this "`OK`" from the Slave NXT.

Step 4: Run `loader.sce`

Scilab version 6.1.0 was a major update and not all build-in ATOM modules have been updated. The Serial Communications toolbox is one example. Thus, one must load the updated toolbox before attempting to execute any Scilab serial functions (see **Figure 1D**). A link to the loader.sce program and related binaries can be found in Scilab's site and downloaded from GitHub. Alternatively it can be downloaded from the course website.

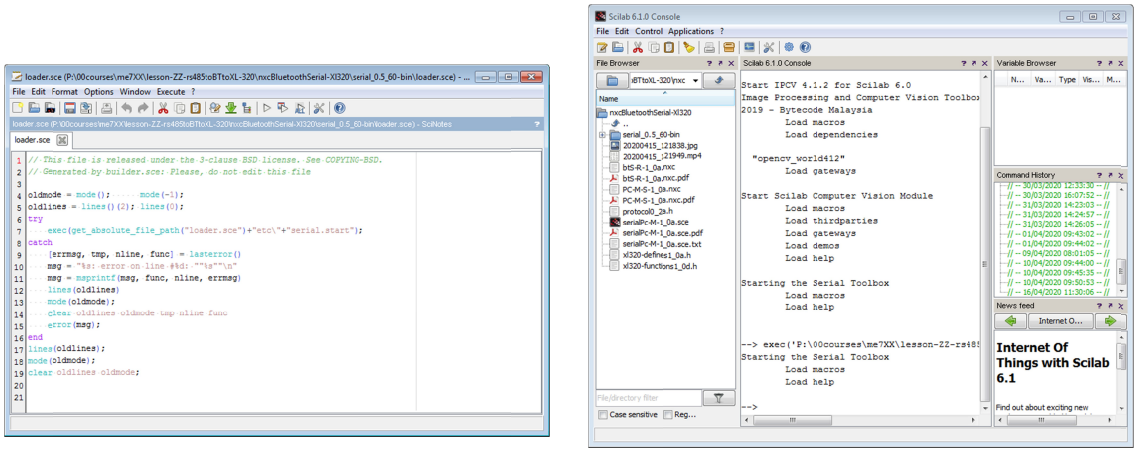

Figure 1D: Updated Serial Communications toolbox loader.sce program (left) is executed. The Scilab console (right) shows that the updated toolbox has been loaded.

Step 5: Execute the programs. See demo video https://youtu.be/BDz0-FkbjOM

A. First turn on Bluetooth on the Master and Slave. The programs assume Bluetooth channel 1 is used.
B. Ensure the XL-320 motor controller board is powered and connected to Port 4 on the Slave NXT.

C. On the Slave NXT execute **btS-R-1_0a.nxc**. The program will move the 2-link planar manipulator to the HOME position. Once done, the LCD will display "Slave" and is ready to accept Bluetooth messages.
D. On the Master NXT execute **PC-M-S-1_0a.nxc**. The LCD will display "Master" and is ready to accept serial messages.
E. On the PC execute **serialPc-M-1_0b.sce**. This will send four different strings. Each string contains a pair of numbers that represent the desired angle settings.
F. After the robot moves to the four different angle settings, Scilab will exit. One can now abort both the Master and Slave NXT programs.

Congratulations! You have implemented distributed computing with a PC and a pair of NXT Bricks using serial and Bluetooth communications!

Exercises

Recalling Concept 1, the PC sends a serial string with the form " @number01, number02" without quotes. An example is " @90, 0". There is a white space followed by the @ character which is immediately followed by the first number. The first number is immediately followed by a comma and another white space and then the second number.

1. Change the format such that the PC sends a serial string with the form " >number01, number02" to the Master NXT. Have the Master NXT confirm that this is a valid string by checking for the ">" character. Hint: What is the ASCII value for the ">" character?

Recalling Concept 1, when the Master NXT receives a valid string from the PC, it replies "ROGER".

2. Change both the NXC and Scilab code accordingly so that "RECEIVED" is used rather than "ROGER".

Recalling Concept 2, when the Slave NXT receives a valid string from the Master, it replies with "OK"

3. Change the NXC code in both the Master and Slave NXTs, so that "confirmed" is used rather than "OK"

NOTE: Serial communications programming can be tricky. One can get Scilab error messages related to Serial Communications when the port does not close properly. One option is to close Scilab and re-open. Then, run the loader.sce program before another attempt at executing SCE code that performs serial communications. Another option is to type `closeserial(h)` in the Scilab console.