Hands-on Lab

XL-320 NXC Programming – Read Position

Encoders in the XL-320 allow one to read angular position. This lab builds on the header files that were previously written to read the XL-320's angles.

Preliminary: Dealing with Half-Duplex

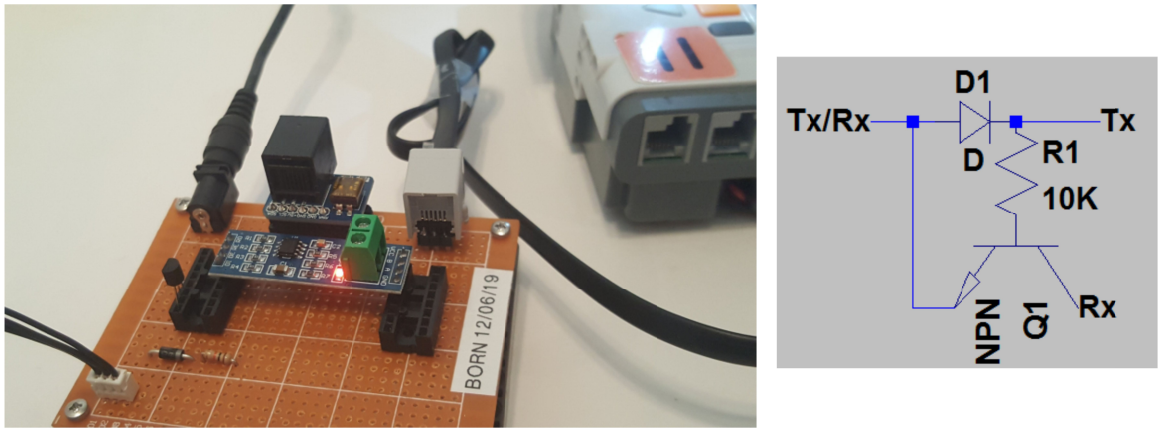

Figure A: The mezzanine board wire-wraps the widely available MAX485 module (left) and implements a transistor-based UART (right) to implement half-duplex communications.

Recall that RS-485 employs voltage differences to represent binary HI and LO states. Also recall that microprocessors often employ TTL (transistor-to-transistor logic) voltage levels, typically 0V and 5V to represent HI and LO states respectively. As such, companies like Maxim offer a chip (e.g. MAX485) to convert RS-485 to TTL voltage levels. The demand for such converters is so high that 3rd party companies offer MAX485 modules (about $2 USD as seen in **Figure A left**).

The XL-320 is that it is a 3-wire device (GND, VCC, and DATA). Having only one line for DATA means that communication is half-duplex; one reads or writes on the DATA line. A 4-wire device (like the MAX485) provides dedicated lines; one for reading and the other for writing. This would enable full-duplex. Thus, one must reconcile this half- versus full-duplex difference between the XL-320 and MAX485 respectively.

Figure A right performs this reconciliation. The XL-320's DATA line attaches to the TX/RX input of the diode. The transmit (i.e. writing) TX and receive (i.e. reading) RX lines attach to the MAX485's inputs. This transistor-based circuit monitors the DATA line state to switch between reading and writing.

Concept 1 Command XL-320 to Read Position `xl320-helloServoRead0_1b.nxc`

Step 1: Open previous `xl320-defines1_0a.h` file

Section 2.2 (Control Table) of the Robotis XL-320 E-Manual (shown again below as **Figure 1B**). `Present Position` has the address 37 Decimal (or 0x25), sized at 2-bytes, and has values from 0 to 1023 Decimal. Viewing the RAM constants defined in `xl320-defines1_0a.h` (**Figure 1A**) verifies this (yellow highlight).

```
// RAM Address related Defines
// See Robotis Section 2.3 http://emanual.robotis.com/docs/en/dxl/x/xl320/

#define RAM_TORQUE_ENABLE          0x18 // 1 byte; turns on/off torque control
#define RAM_LED                    0x19 // 1 byte; changes motor's LED color
#define RAM_D_GAIN                 0x1B // 1 byte; motor's derivative gain
#define RAM_I_GAIN                 0x1C // 1 byte; motor's integral gain
#define RAM_P_GAIN                 0x1D // 1 byte; motor's proportional gain
#define RAM_GOAL_POSITION          0x1E // 2 bytes; destination position value
                                        // from [0, 1023] with 0 most CW and
                                        // 1023 most CCW
#define RAM_MOVING_SPEED           0x20 // 2 bytes; Wheel or Joint dependent
#define RAM_TORQUE_LIMIT           0x23 // 2 bytes; maximum torque limit value
#define RAM_PRESENT_POSITION       0x25 // 2 bytes; motor's present position
                                        // value [0-1023]
#define RAM_PRESENT_SPEED          0x27 // 2 bytes; Wheel or Joint mode
                                        // dependent [0-2047]
#define RAM_PRESENT_LOAD           0x29 // 2 bytes; currently applied load
                                        // value is [0-2047]
#define RAM_PRESENT_VOLTAGE        0x2D // 1 byte; present supply voltage
#define RAM_PRESENT_TEMPERATURE    0x2E // 1 byte; motor's internal temperature
                                        // in Celsius
#define RAM_REGISTERED             0x2F // 1 byte; REG_WRITE instruction
                                        // received or not
#define RAM_MOVING                 0x31 // 1 byte; Goal Position completed
                                        // or in-progress
#define RAM_HARDWARE_ERROR_STATUS  0x32 // 1 byte; present hardware error status
#define RAM_PUNCH                  0x33 // 2 bytes; minimum current to//
```

Figure 1A: RAM constants in `xl320-defines1_0a.h` file

2. 3. Control Table of RAM Area

Address	Size (Byte)	Data Name	Description	Access	Initial Value	Min	Max
24	1	Torque Enable	Motor Torque On/Off	RW	0	0	1
25	1	LED	Status LED On/Off	RW	0	0	7
27	1	D Gain	Derivative Gain	RW	0	0	254
28	1	I Gain	Integral Gain	RW	0	0	254
29	1	P Gain	Proportional Gain	RW	32	0	254
30	2	Goal Position	Desired Position	RW	-	0	1023
32	2	Moving Speed	Moving Speed(Moving Velocity)	RW	-	0	2047
35	2	Torque Limit	Torque Limit(Goal Torque)	RW	-	0	1023
37	2	Present Position	Present Position	R	-	-	-
39	2	Present Speed	Present Speed	R	-	-	-
41	2	Present Load	Present Load	R	-	-	-
45	1	Present Voltage	Present Voltage	R	-	-	-
46	1	Present Temperature	Present Temperature	R	-	-	-
47	1	Registered	If Instruction is registered	R	0	-	-
49	1	Moving	Movement Status	R	0	-	-
50	1	Hardware Error Status	Hardware Error Status	R	0	-	-
51	2	Punch	Minimum Current Threshold	RW	32	0	1023

Figure 1B: Robotis XL-320 E-manual Section 2.3 shows Present Position (red circle)

Step 2: Open `xl320-functions1_0c.h` and confirm `XL320_servoRead` function

Recall that the first step to create a packet is determining the packet length (i.e. number of parameters). To read the XL320's encoders, one needs 4 parameters: the `RAM_PresentPosition`, 0x00, and number of bytes (which is 2 or 0x02), and position. Recall that packet length is the number of parameters (4 in this case) plus 3. Thus, the packet length is 7. **Figure 1C** pictorially shows this packet.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
H1	H2	H3	RSRV	ID	LEN1	LEN2	INST	P1	P2	P3	P4	CRC1	CRC2
0xFF	0xFF	0xFD	0x00	ID	0x07	0x00	0x02	0x25	0x00	2-bytes	Read Position	?	?

These are set to `read` from XL-320

Read (angle) position

Figure 1C: Packet to command XL-320 to read angle position

The resulting `XL320_servoRead` function is given in **Figure 1D**. Note: Yellow-highlight shows packet length.

```
// ---------------------------------------------------------------------------
// Servo Function: read XL-320 position
void XL320_servoRead(unsigned char XL320_motorId) {

  // Variables to set Length 1 and Length 2
  // unsigned char XL320_setServoLength_L;
  // unsigned char XL320_setServoLength_H;
  byte XL320_setServoReadLength_L;
  byte XL320_setServoReadLength_H;

  // Variables for position and speed
  unsigned char XL320_position_L, XL320_position_H;
  unsigned char XL320_speed_L, XL320_speed_H;
  // byte XL320_position_L, XL320_position_H;
  // byte XL320_speed_L, XL320_speed_H;

  // Variables to set up packet array
  unsigned char tempPacket[];   // temporary packet
  unsigned char finalPacket[];  // final packet to transmit

  // Variables for checksum CRC
  unsigned short setServoRead_CRC;
  byte CRC_L, CRC_H;

  // 1. Calculate lengths
  // Recall that Length 1 and Length 2 = number of parameters + 3
  // Reading servo requires only 4 parameters: RAM_PresentPosition, 0x00,
  // Number of bytes (which is 2), and 0x00
  // Hence number of paramters + 3 is 4 + 3 = 7 Dec = 0x07

  XL320_setServoReadLength_L = 0x07; // Read Instruction
  XL320_setServoReadLength_H = 0x00;

  // 2. Construct first part of packet
  ArrayBuild(tempPacket, HEADER_1, HEADER_2, HEADER_3, RESERVED, XL320_motorId,
    XL320_setServoReadLength_L, XL320_setServoReadLength_H, INSTRUCTION_READ,
    RAM_PRESENT_POSITION, 0x00, 0x02, 0x00);
```

Figure 1D: `XL320_servoRead` function in `xl320-functions1_0c.h`

```
  // 3. Perform checksum, see Section 1.2
  // of http://emanual.robotis.com/docs/en/dxl/crc/
  unsigned int packetLength = (XL320_setServoReadLength_H >> 8) + XL320_setServoReadLength_L;

  // See last bullet in Section 1.2 "Packet Analysis and CRC Calculation"
  // http://emanual.robotis.com/docs/en/dxl/crc/
  setServoRead_CRC = update_crc(0, tempPacket, 5 + packetLength);
  CRC_L = (setServoRead_CRC & 0x00FF);
  CRC_H = (setServoRead_CRC >> 8) & 0x00FF;

  // 4. Concatenate into final packet and sent thru NXT RS485
  ArrayBuild(finalPacket, tempPacket, CRC_L, CRC_H);
  RS485Write(finalPacket);

  // 5. Call inline function
  waitForMessageToBeSent();

}; // end XL320_servoRead
```

Figure 1D: Continued

The packet is completed by adding the CRC checksum values, returned from the call to `update_crc`.

Make sure the above code is saved into `xl320-functions1_0c.h`. This will ensure XL320_servo can be called when needed.

Step 3: Write NXC Program **`xl320-helloServoRead0_1b.nxc`**

NB: This example may require using Dynamixel Wizard to manually set the XL-320 to Joint Mode

Figure 1F lists the NXC program that commands the XL-320 to read angles. The program begins by including the H-files containing XL-320 constants (`xl320-defines1_0a.h`) and functions (`xl320-functions1_0c.h`).

In `main`, Boolean variables for the NXT Brick's buttons are declared. The Brick's serial port is enabled and configured for 57,600 baud, at 8N1 (8-bits, no parity, 1 stop bit). A call to `XL320_setTorqueEnable` is made to turn off torque enable.

The `do-while` loop first calls `XL320_servoRead` and waits for data to be available. Once available, the NXC function `RS485Read` is called to read from the XL-320. The XL-320 sends a `Status Packet` containing information, including the angle position. This status packet is stored in the `char` array named `data`.

Section 3 of the Dynamixel Protocol 2.0 shows the Status Packet's form (**Figure 1E**).

3. Status Packet

Header1	Header2	Header3	Reserved	Packet ID	Length1	Length2	Instruction	ERR	PARAM	PARAM	PARAM	CRC1	CRC2
0xFF	0xFF	0xFD	0x00	ID	Len_L	Len_H	Instruction	Error	Param 1	...	Param N	CRC_L	CRC_H

3. 1. Instruction

Instruction of the Status Packet is designated to 0x55 (Status)

Figure 1E: Protocol 2.0 Status Packet format

```
// FILE: xl320-helloServoRead0_1b.nxc - Works!
// DATE: 12/10/19 08:41
// AUTH: P.Oh
// DESC: Command servo to rotate back-and-forth by fixed amount
// VERS: 0.1a: based on P.Oh's xl320-defines1_0b.h and xl320-funtions1_0b.h
//             xl320-helloServo1_0a.nxc
//       0.1b: troubleshotting 0.1a
// REFS: xl320-functions1_0b.h; xl320-defines1_0a.h, xl320-helloLed1_0a.nxc
//       09/10/19 ppt-paulOhDynamixelXl320HeaderFile-1.0a.pptx
// NOTE: If factory default XL-320 used, then ID is 0x01
//       ID of 0xFE commands any and all XL-320 motors

#include "xl320-defines1_0a.h"   // XL-320 defines from Control Table
#include "xl320-functions1_0c.h" // P.Oh functions written for XL-320
                                 // 1.0b.h contains XL320_servoRead
                                 // 1.0c.h updated the XL320_servoRead

#define ID_ALL_MOTORS 0XFE       // 0XFE commands all XL-320 motors
#define ID_MOTOR01    0X01       // Assumes Motor 1 configured with ID = 1

task main() {

  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
  unsigned char data[13];        // 13-byte status packet from RS485
  int   Position;

  UseRS485();
  RS485Enable();
  // Note: First, use Dynamixel Wizard to set XL-320 to desired baud rate
  // Then, use RS485Uart to match this baud rate e.g. 57600
  RS485Uart(HS_BAUD_57600, HS_MODE_8N1); // 57600 baud, 8bit, 1stop, no parity
  Wait(100);

  // Turn off Torque enable so that one can freely turn XL320 axle by hand
  XL320_setTorqueEnable(ID_MOTOR01, 0); // 0 = turn OFF torque enable
  Wait(100);

  ClearScreen();
  // Prompt user to begin
  TextOut(0, LCD_LINE1, "Stop: Press GRAY" );
  while(true) {
     XL320_servoRead(ID_MOTOR01);
     Wait(20);
     until(RS485DataAvailable());
     RS485Read(data);
     // data[9] = LO and data[10] HI byte contain XL-320 position
     // Thus formulate the position and display as integer
     Position = data[9] + (data[10] << 8);
     ClearScreen();
     TextOut(10, LCD_LINE3, FormatNum("Pos = %5.5d" , Position));

     if(ButtonPressed(BTNRIGHT, FALSE)) {
         while(ButtonPressed(BTNRIGHT, FALSE)) {
           // Do nothing, but this check flushes any key presses
         };
         XL320_servo(ID_MOTOR01, 700, 200); // rotate to motor position 700, speed 200
         Wait(200);
     }
     else if(ButtonPressed(BTNLEFT, FALSE)) {
         while(ButtonPressed(BTNLEFT, FALSE)) {
           // Do nothing, but this check flushes any key presses
         };
         XL320_servo(ID_MOTOR01, 200, 200);   // counter-rotate to 200 at speed 200;
         Wait(200);
     }; // end if

  }; // end while;

} // end main
```

Figure 1F: Listing for `xl320-helloServoRead0_1b.nxc`

When the Status Packet is used to read angle position, **Figure 1E** will take the form of **Figure 1G**.

Header				ID	Length		Inst	Err	Param		CRC	
FF	FF	FD	00	01	06	00	55	00	E8	03	C3	2B
0	1	2	3	4	5	6	7	8	9	10	11	12

Figure 1G: Example Status Packet send from the XL-320 when reading angle position

The angle position is stored in `data[9]` and `data[10]`. In the example shown in **Figure 1G**, `data[9] = 0xE8` and `data[10] = 0x03`. It takes 2-bytes to store the XL-320 angle position because angle values can range from 0 to 1023. Recall that packets are composed of bytes. A byte is 8-bits, and thus can only hold values from 0 to 255.

Thus, the angle position needs to be reconstructed from the 2-bytes. Robotis Protocol 2.0 says that packets apply Little Endian. Thus, data[9] is the LO byte and data[10] is the HI byte. In `xl320-helloServoRead0_1b.nxc` one sees (Yellow Highlight) the bit-wise line:

```
Position = data[9] + (data[10] << 8);
```

Figure 1H shows this example pictorially. The columns 0 to 15 represent the 16-bit variable named `Position`. In the example (**Figure 1G**) `data[9] = 0xE8 = 1110 1000` binary (2nd row) and is the LO byte and `data[10] = 0x03 = 0000 0011` (binary) is the HI byte. The 3rd row of Figure 1H shows the result of a left bit-wise shift of 8 positions. Adding the 2nd and 3rd rows results in the 4th row: 0000 0011 1110 1000 binary which is 0x03E8 or 1000 Decimal. Thus, **Figure 1G** shows a Status Packet that reports the XL-320 being at angle position 1000.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Data[9]									1	1	1	0	1	0	0	0	0xE8
Data[10]<<8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0x0300
Data[9]+Data[10]<<8	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	0x03E8

Figure 1H: Pictorial representation of Little Endian bit-wise operation to reconstruct angle position

The program displays the resulting `Position` using `TextOut`. Pushing the right or left arrow buttons on the NXT Brick will rotate the XL-320 to Position 700 or 200 respectively, at speed 200.

Congratulations! You can command the XL-320 to read angle positions

Exercises

1.1 Write an NXC program that slowly rotates the XL-320 from 0 to 1000. Every time the angle reaches a multiple of 100 units (recall that XL-320 resolution is 0.29 degrees/unit), stop the XL-320, play a tone, and then continue moving to position 1000.

1.2 Write a function for reading the XL-320's angular velocity called `XL320_servoReadVelocity`. Test your function, with an NXC program that rotates the XL-320 (e.g. Wheel Mode) at a known velocity.

1.3 Test your `XL320_servoReadVelocity` function by writing an NXC program that begins rotating in Wheel Mode, at speed 200. Increment or decrement the velocity by 100 each time the right or left arrow buttons are pushed respectively. Keep the minimum and maximum speed values at 200 and 700 respectively.