**Hands-on Lab**

**XL-320 NXC Programming – Write Position**

Header files that define constants and contain XL-320 functions were created. This lab builds on this to command the XL-320 to rotate to desired angles.

**Preliminary:** 1-DOF Planar Manipulator

**Figure A:** Pictured is an XL-320-based 1-DOF planar manipulator using Lego and M2.5 fastener hardware. The manipulator and XY Cartesian axes are mounted on a 32 stud by 32 stud Lego base plate.



In Dynamixel Wizard, make sure the XL-320 has the following settings:

* Baud Rate: 57,600
* Motor ID: 0x01
* Torque Enable: On
* Velocity is at a slow setting e.g. 200
* Motor position is centered

**Concept 1 Command XL-320 to Rotate Back-and-Forth xl320-helloServo1\_0a.nxc**

**Step 1:** Open previous xl320-defines1\_0a.h file

In a prior lab, the function XL320-setLed was created using [Section 2.2](http://emanual.robotis.com/docs/en/dxl/x/xl320/#control-table) (Control Table) of the Robotis XL-320 E-Manual (shown again below as Figure 1B). Goal Position has the address 30 Decimal (or 0x1E), sized at 2-bytes, and has values from 0 to 1023 Decimal. Viewing xl320-defines1\_0a.h verifies this:

// 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

**Figure 1A:** Addresses (in Decimal) for each Data Name in **RAM**. This table can be found in [Section 2.2](http://emanual.robotis.com/docs/en/dxl/x/xl320/#control-table) (Control Table) of the Robotis XL-320 E-Manual.



**Step 2:** Create Definition Header File (H-File) – RAM Area

Section 2.3 of <http://emanual.robotis.com/docs/en/dxl/x/xl320/> details the RAM Control Table and shown in **Figure 1C**. Following the aforementioned naming convention, **Figure 1D** shows the #defines to be added to the H-file in **Figure 1B**.

**Step 2:** Open xl320-functions1\_0c.h and write XL320\_servo function

The write instruction (0x03) was used to write values (and hence desired colors) to change the XL-320’s LED. Similarly, 0x03 will be used again, but with desired angle position and velocity values. Recall that the status packet has the form:



[Section 3](http://emanual.robotis.com/docs/en/dxl/protocol2/#status-packet) of the Robotis Dynamixel Protocol 2.0 illustrates the packet format

However, there to command the XL-320 to move, 6 parameters will be required: Goal Position; 0x00; Position LO byte, Position HI byte; Velocity LO byte, and Velocity HI byte. Recall that packet length is the number of parameters (6 in this case) plus 3. Thus, the packet length is 9. **Figure 1B** pictorially shows this packet.



**Figure 1B:** Packet to command XL-320 to desired position and/or velocity

The resulting XL320\_servo function is given in **Figure 1C**.

// -------------------------------------------------------------------------

// Servo Function: move XL-320 to desired position and desired speed

void XL320\_servo(unsigned char XL320\_motorId,

 unsigned int XL320\_desiredPosition,

 unsigned int XL320\_desiredSpeed) {

 // Variables to set Length 1 and Length 2

 // unsigned char XL320\_setServoLength\_L;

 // unsigned char XL320\_setServoLength\_H;

 byte XL320\_setServoLength\_L;

 byte XL320\_setServoLength\_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 setServo\_CRC;

 byte CRC\_L, CRC\_H;

 // 1. Calculate lengths

 // Recall that Length 1 and Length 2 = number of parameters + 3

 // Setting Servo requires only 6 parameters: Goal Position, 0x00, Position\_L,

 // Position\_H, Speed\_L, and Speed\_H

 // Hence number of parameters + 3 is 6 + 3 = 9 Dec = 0x09

 XL320\_setServoLength\_L = 0x09;

 XL320\_setServoLength\_H = 0x00;

 XL320\_position\_L = XL320\_desiredPosition; // Lower byte of 16-bit position

 XL320\_position\_H = XL320\_desiredPosition >> 8; // Upper byte

 XL320\_speed\_L = XL320\_desiredSpeed; // Lower byte of 16-bit speed

 XL320\_speed\_H = XL320\_desiredSpeed >> 8; // Upper byte

**Figure 1C:** XL320\_servo function in xl320-functions1\_0c.h

 // 2. Construct first part of packet

 ArrayBuild(tempPacket, HEADER\_1, HEADER\_2, HEADER\_3, RESERVED, XL320\_motorId,

 XL320\_setServoLength\_L, XL320\_setServoLength\_H, INSTRUCTION\_WRITE,

 RAM\_GOAL\_POSITION, 0x00, XL320\_position\_L, XL320\_position\_H,

 XL320\_speed\_L, XL320\_speed\_H);

 // 3. Perform checksum, see Section 1.2

 // of http://emanual.robotis.com/docs/en/dxl/crc/

 unsigned int packetLength = (XL320\_setServoLength\_H >> 8) + XL320\_setServoLength\_L;

 // See last bullet in Section 1.2 "Packet Analysis and CRC Calculation"

 setServo\_CRC = update\_crc(0, tempPacket, 5 + packetLength);

 CRC\_L = (setServo\_CRC & 0x00FF);

 CRC\_H = (setServo\_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\_servo

/\* ========================================= \*/

**Figure 1C**: 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-helloServo1\_0a.nxc**

**Figure 1D** lists the NXC program that commands the XL-320 to rotate back-and-forth. 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).

The do-while loop first calls XL320\_servo with an angular position of 900 and angular velocity of 200. The XL-320 features on-the-fly changes; once the position and velocity command is issued, the next command is processed. Thus, a Wait(1500) is used to wait until the XL-320 has reached position 900.

The XL-320 then rotates to position 0 at an angular velocity of 200. Again, a Wait(1500) is issued to ensure the servo reaches this position. The loop iterates this back-and-forth rotation until the NXT’s grey button is pushed.

// FILE: xl320-helloServo1\_0a.nxc - Works!

// DATE: 12/08/19 14:03

// AUTH: P.Oh

// DESC: Command servo to rotate back-and-forth by fixed amount

// VERS: 1.0a: based on P.Oh's xl320-defines1\_0a.h and xl320-funtions1\_0a.h

// REFS: xl320-functions1\_0a.h; xl320-defines.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

#define ID\_ALL\_MOTORS 0XFE // 0XFE commands all XL-320 motors

#define ID\_MOTOR01 0X01 // Assumes Motor 1 configured with ID = 01

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

 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

 ClearScreen();

 // Prompt user to begin

 TextOut(0, LCD\_LINE1, "Stop: Press GRAY" );

 do {

 greyButtonPushed = ButtonPressed(BTNEXIT, FALSE);

 XL320\_servo(ID\_ALL\_MOTORS, 900, 200); // rotate to motor position 900, speed 200

 Wait(1500);

 XL320\_servo(ID\_ALL\_MOTORS, 0, 200); // counter-rotate to 0 at speed 200;

 Wait(1500);

 } while(!greyButtonPushed);

 ClearScreen();

} // end main

**Figure 1D:** NXC program xl-320-helloServo1\_0a.nxc

Congratulations! You can command the XL-320 to rotate to desired angles at desired speeds

Exercises

* 1. From the above figure, what is the resolution of the XL-320? Hint: 1024 range yields 300 degrees of motion
	2. Write a NXC program to home the 1-DOF planar manipulator at position 512. This puts the 1-DOF planar manipulator in the 12:00 position. Then command the servo to rotate 45-degrees clockwise. What the XY stud position of the manipulator’s end-effector?
	3. Based on the end-effector’s length, determine a desired XY stud position. Calculate the required angle and command the XL-320 to that stud position

