**Hands-on Lab**

**NXC Programming – RS-485 Communications**

The Lego NXT Brick can communicate with other peripherals via RS-485 serial communications. Port 4 on the Brick provides this high-speed full-duplex capability. Many peripherals like scanners, joysticks, keypads, and the XL-320 servo use this serial communication protocol. RS-485 can also connect the NXT Brick to other micro-processors and computers that have this port. This lab explores RS-485 NXC programming to send and receive packets.

**Preliminary:** NXT and Laptop Setup

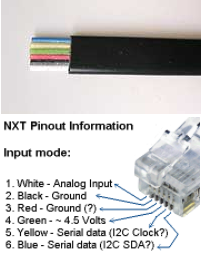
**Figure A:** NXT communicating to a Windows-based laptop via RS-485 (left). A USB-to-RS485 (right) connects into the laptop.





Most laptops today feature USB ports for serial communications. As such, a USB-to-RS485 dongle can plug into the laptop. **Figure A** (right) shows one purchased from [Amazon.com](https://www.amazon.com/gp/product/B07SD65BVF/ref=ppx_yo_dt_b_asin_title_o02_s00?ie=UTF8&psc=1); it is labeled to indicate the function of each screw terminal line. A spliced NXT cable (**Figure B left**) was then inserted into the dongle (**Figure B right**).

**Figure B:** NXC Cable pinout (left). NXC cabled spliced and inserted into USB-to-RS485 dongle (right)





Blah: RJ-45 connectors?

Lastly, a terminal emulator is needed to establish serial communications. [Hercules](https://www.hw-group.com/software/hercules-setup-utility) is an example of a free Windows-based emulator. Installing this executable allows one to choose settings like baud rate and parity.

**Concept 1 ASCII Character Transmission**

RS-232 and its descendent RS-485 employ HI and LO voltages to represent ON and OFF binary states. In RS-232, the voltages are -12V to +12V values. RS-485 uses lower values, but more importantly, use voltage differences in order to offer more robust transmission.

Most micro-processor based systems employ lower voltages (typically 0 to +5V). As such, a converter is employed to transform RS-232 or RS-485 states to

**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

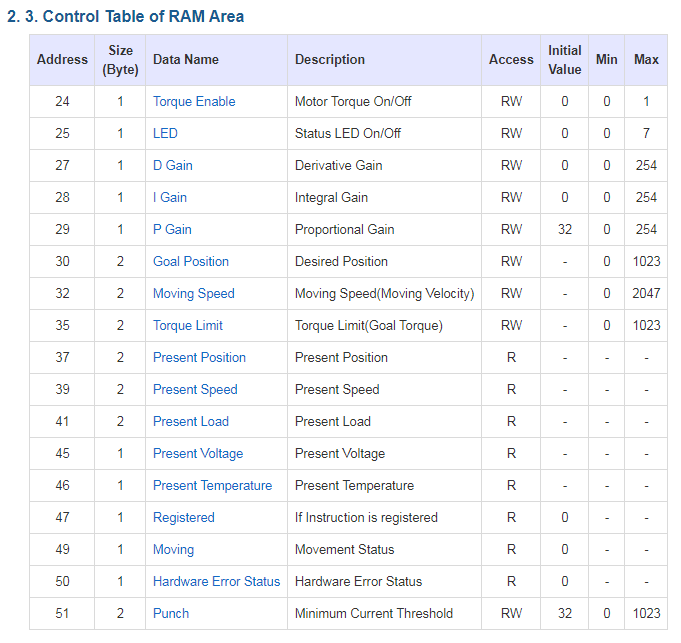
#define RAM\_P\_GAIN 0x1D // 1 byte; motor's proportional gain

#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.

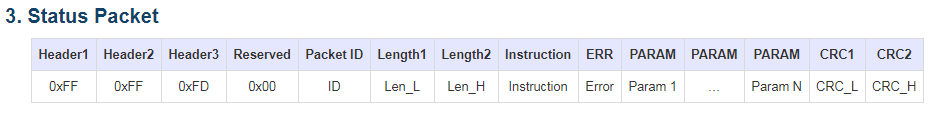


**Step 2:** Create Definition Header File (H-File) – RAM Area

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**.

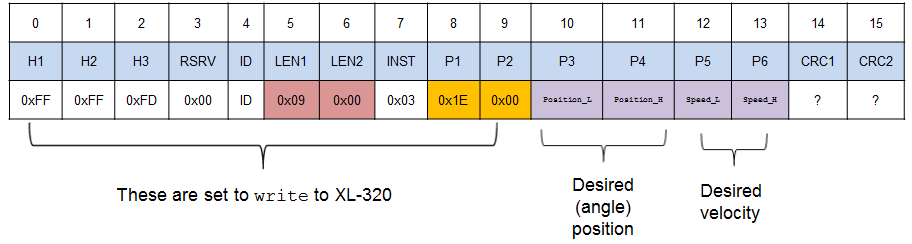
**Step 2:** Open xl320-functions1\_0c.h and write XL320\_servo function

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();

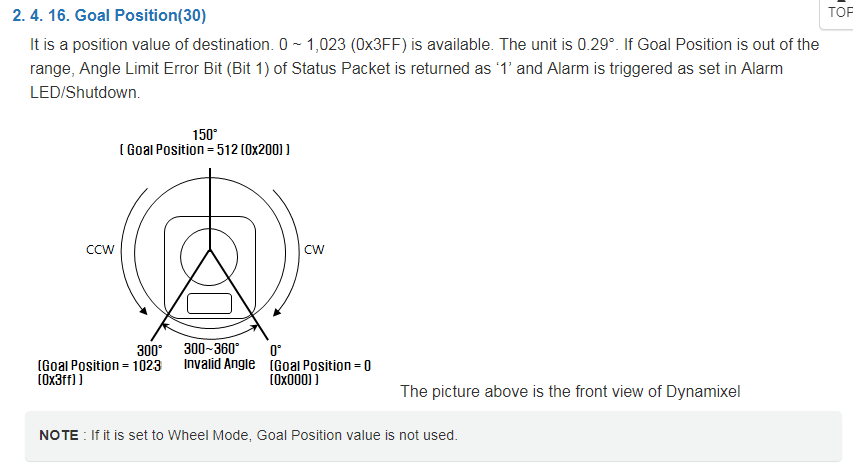
} // end main

**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



**Concept 2 Command XL-320 to Wheel Mode xl320-helloWheelMode1\_0a.nxc**

Wheel mode allows the XL-320 to rotate continuously. In the Dynamixel Wizard mode, one might recall that Torque Enable must first be turned off. Then, one can select Wheel Mode and then command the XL-320 to rotate at a desired velocity.

**Step 1:** Open previous xl320-functions1\_0d.h file

Two functions are created and saved in xl320-functions1\_0d.h. The first is XL320\_setTorqueEnable. The Control Table in **Figure 1A** shows that Torque Enable is at address 25 Decimal (defined as RAM\_TORQUE\_ENABLE in xl-defines1\_0.h) and takes 1 byte. Thus the number of parameters will be 3 and the resulting packet (number of packets + 3) will be 6 Decimal (or 0x06). The function’s listing is given in **Figure 2A**.

The second function created is XL320\_controlMode. [Section 2.2](http://emanual.robotis.com/docs/en/dxl/x/xl320/#control-table) for the EEPROM area of the XL-320’s firmware shows that it has an address of 11 Decimal (defined as EEPROM\_CONTROL\_MODE in xl-defines1\_0.h) and takes 1 byte as well. The packet length will be 6 Decimal (or 0x06). This function’s listing is given in **Figure 2B**.

// XL320\_setTorqueEnable Function: Enable Torque on or off on XL-320 motor

void XL320\_setTorqueEnable(unsigned char XL320\_servoId,

unsigned char XL320\_torqueEnable) {

// Section 2.1.1 http://emanual.robotis.com/docs/en/dxl/x/xl320/

// says that changing EEPROM areas in Control table, requires setting

// Torque Enable to zero (i.e. off). EG: Baud Rate is under EEPROM Control

// area. So, if one wishes to set the baud rate, one probably needs to turn

// off Torque Enable

// Torque Enable Section 2.4.13

// http://emanual.robotis.com/docs/en/dxl/x/xl320/#torque-enable

// Takes 1 byte. 0 = Off; 1 = On

// Variables to set Length 1 and Length 2

unsigned char XL320\_setTorqueEnableLength\_L;

unsigned char XL320\_setTorqueEnableLength\_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 setTorqueEnable\_CRC;

byte CRC\_L, CRC\_H;

// 1. Calculate lengths

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

// Setting Torque Enable requires only 3 parameters: address, 0x00 and Torque Enable value

// Hence number of (paramters + 3) is (3 + 3) = 6 Dec = 0x06

XL320\_setTorqueEnableLength\_L = 0x06;

XL320\_setTorqueEnableLength\_H = 0x00;

// 2. Construct first part of packet

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

XL320\_setTorqueEnableLength\_L, XL320\_setTorqueEnableLength\_H, INSTRUCTION\_WRITE,

RAM\_TORQUE\_ENABLE, 0x00, XL320\_torqueEnable);

// 3. Perform checksum, see Section 1.2 of http://emanual.robotis.com/docs/en/dxl/crc/

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

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

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

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

CRC\_H = (setTorqueEnable\_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\_setTorqueEnable

**Figure 2A:** Listing for function XL320\_setTorqueEnable (see xl-functions1\_0d.h)

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

// Control Mode Function: set XL-320 to Wheel or Joint mode

// XL320\_controlModeDesired = 1 (Wheel Mode) or 2 (Joint mode)

void XL320\_controlMode(unsigned char XL320\_motorId,

unsigned char XL320\_controlModeDesired) {

// Variables to set Length 1 and Length 2

byte XL320\_setControlModeLength\_L;

byte XL320\_setControlModeLength\_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 setControlMode\_CRC;

byte CRC\_L, CRC\_H;

// 1. Calculate lengths

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

// Setting Servo requires only 3 parameters: Goal Position, 0x00, desired mode

// Hence number of paramters + 3 is 3 + 3 = 6 Dec = 0x06

XL320\_setControlModeLength\_L = 0x06;

XL320\_setControlModeLength\_H = 0x00;

// 2. Construct first part of packet

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

XL320\_setControlModeLength\_L, XL320\_setControlModeLength\_H, INSTRUCTION\_WRITE,

EEPROM\_CONTROL\_MODE, 0x00, XL320\_controlModeDesired);

// 3. Perform checksum, see Section 1.2

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

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

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

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

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

CRC\_H = (setControlMode\_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\_controlMode

**Figure 2B:** Listing for XL320\_controlMode in xl-functions1\_0d.h

**Step 1:** Write NXC Program to Rotate XL-320 continuously xl320-helloWheelMode1\_0a.nxc

**Listing 2C** is NXC code where hitting the NXT Brick’s buttons will rotate the XL-320 clockwise, counter-clockwise or quit. After RS-485 communications have been set (57,600 baud, 8N1), the process involves turning Torque Enable off, selecting Wheel Mode, and then commanding continuous angular rotations at the desired speed (e.g. 200).

// FILE: xl320-helloWheelMode1\_0a.nxc - Works!

// DATE: 12/23/19 08:38

// AUTH: P.Oh

// DESC: NXT commands Dynamixel XL-320 in wheel mode

// VERS: 1.0a: uses xl320-functions1\_0d.h

// - XL320\_TorqueEnable

// - XL320\_ControlMode

// REFS: wheelJointXl320-1.0b.nxc

// 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"

#include "xl320-functions1\_0d.h" // contains XL320\_ControlMode function

#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;

bool leftArrowButtonPushed, rightArrowButtonPushed;

UseRS485();

RS485Enable();

RS485Uart(HS\_BAUD\_57600, HS\_MODE\_8N1); //9600 baud, 8bit, 1stop, no parity

Wait(MS\_100);

// First, home to center position

TextOut(0, LCD\_LINE1, "Homing..." );

XL320\_servo(ID\_ALL\_MOTORS, 512, 200); // 512 should be center position

Wait(2000);

TextOut(0, LCD\_LINE2, "Homed..." );

// Second, turn XL-320 torque enable OFF (ON/OFF = 1/0)

XL320\_setTorqueEnable(ID\_ALL\_MOTORS, 0);

Wait(20);

// Third, select Wheel Mode

XL320\_controlMode(ID\_ALL\_MOTORS, 1); // 1 = Wheel Mode; 2 = Joint Mode

Wait(20);

ClearScreen();

TextOut(0, LCD\_LINE2, "In Wheel mode" );

TextOut(0, LCD\_LINE4, "<-/->/ORG CW/CCW/QUIT" );

do {

rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);

if(rightArrowButtonPushed) {

TextOut(0, LCD\_LINE6, "CCW" );

XL320\_servo(ID\_ALL\_MOTORS, 0, 250); // Continuous CCW rotation

// Section 2.4.21 says 0-1023 is CCW; 1024-2047 is CW

// http://emanual.robotis.com/docs/en/dxl/x/xl320/#moving-speed

Wait(2000);

};

leftArrowButtonPushed = ButtonPressed(BTNLEFT, FALSE);

if(leftArrowButtonPushed) {

TextOut(0, LCD\_LINE6, "CW " );

XL320\_servo(ID\_ALL\_MOTORS, 0, 1024 + 250); // Continuous CCW rotation

Wait(2000);

};

orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);

} while(!orangeButtonPushed);

**Figure 2C:** NXC code xl-320-helloWheelMode1\_0a.nxc rotates continuously CW or CCW

// Turn XL-320 torque enable ON (ON/OFF = 1/0)

XL320\_setTorqueEnable(ID\_ALL\_MOTORS, 0);

Wait(200);

TextOut(0, LCD\_LINE1, "Torque Enable: OFF..." );

// Return back to Joint Mode

XL320\_controlMode(ID\_ALL\_MOTORS, 2); // 1 = Wheel Mode; 2 = Joint Mode

Wait(200);

ClearScreen();

TextOut(0, LCD\_LINE3, "Joint mode..." );

TextOut(0, LCD\_LINE4, "Homing..." );

XL320\_servo(ID\_ALL\_MOTORS, 512, 200); // 512 should be center position

Wait(2000);

TextOut(0, LCD\_LINE6, "Quitting" );

PlaySound(SOUND\_DOWN);

} // end main

**Figure 2C continued:**

Congratulations! You can command the XL-320 to rotate continuously (i.e. Wheel Mode) at desired angular velocities

Exercises

* 1. Write an NXC program that reads the NXT Brick’s left and right buttons. When the right button is pushed, the XL-320 velocity increases by 100. When the left button is pressed, the velocity decreases by 100. Hitting the Orange button stops rotation.
  2. Write an NXT program that switches from Wheel Mode and Joint Mode. When the left arrow button is pushed, the XL-320 rotates continuously (say, at 200). When right arrow button is pushed, the XL-320 rotates from -90 to +90 degrees e.g. **wheelJointXl320-1.0b.nxc** demo.