Hands-on Lab

XL-320 NXC Programming – Write Joint and Wheel Modes

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 (joint mode) or angular velocities (wheel mode)

Preliminary: 1-DOF Planar Manipulator

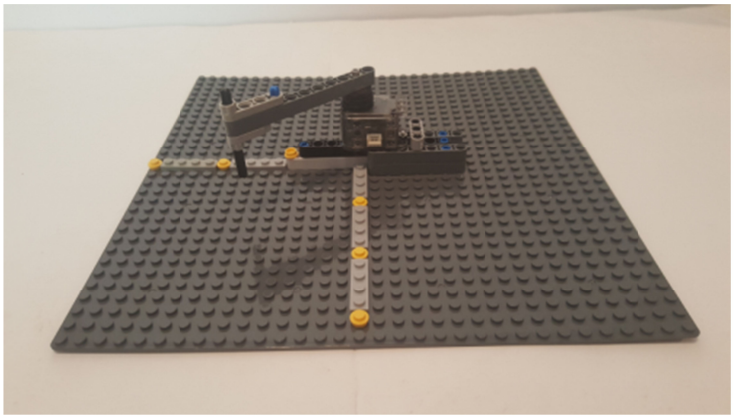


Figure A: Pictured is an XL-320-based 1-DOF planar manipulator using Lego and M2.5 fastener hardware. The manipulator and XY Cartesian axes are mounted on a 32 stud by 32 stud Lego base plate.

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 (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
```

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 1A: Addresses (in Decimal) for each Data Name in **RAM**. This table can be found in Section 2.2 (Control Table) of the Robotis XL-320 E-Manual.

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:

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

Section 3 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.

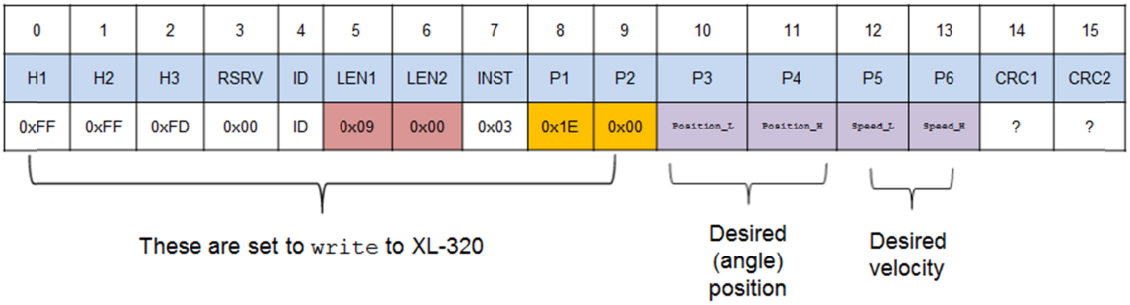


Figure 1B: Packet to command XL-320 to desired position and/or velocity

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
```

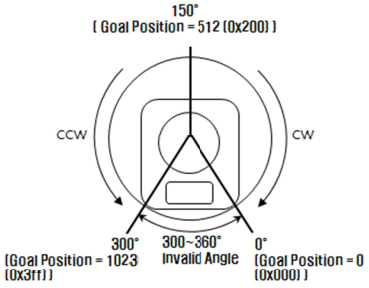
Figure 1D: NXC program xl-320-helloServo1_0a.nxc

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

2. 4. 16. Goal Position(30)

TOP

It is a position value of destination. 0 ~ 1,023 (0x3FF) is available. The unit is 0.29°. If Goal Position is out of the range, Angle Limit Error Bit (Bit 1) of Status Packet is returned as '1' and Alarm is triggered as set in Alarm LED/Shutdown.



The picture above is the front view of Dynamixel

NOTE : If it is set to Wheel Mode, Goal Position value is not used.

Exercises

1.1 From the above figure, what is the resolution of the XL-320? Hint: 1024 range yields 300 degrees of motion

1.2 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?

1.3 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 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

2.1 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.

2.2 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.