Hands-on Lab

XL-320 NXC Programming – Forward Kinematics

Reference frames determine the relationship of the end-effector's position relative to the base. Denavit-Hartenberg (DH) notation prescribes the position and orientation of each joint's frame. The resulting frames define the tool transformation matrix and hence the robot's forward kinematics. This lab commands the 2-link planar manipulator's joint angles to verify its end-effector reaches the theoretical position.

Preliminary: 2-link Planar Manipulator and Forward Kinematics

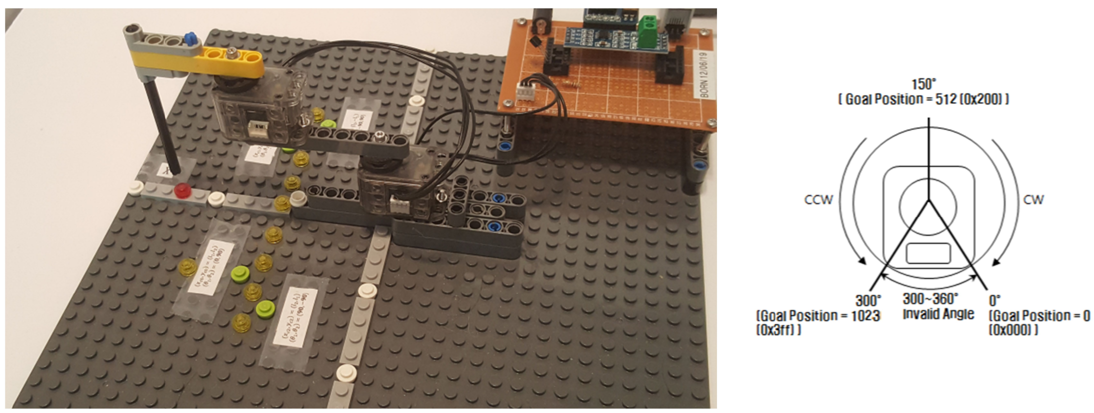

Figure A: XL-320 servos serve as Joints 1 and 2 of a LEGO-based 2-link planar manipulator (left). The colored circular 1-stud bricks on the base plate are various (x, y) goals positions for the end-effector. XL-320 manual shows the servo is centered at position 512 [counts] (right).

Figure B shows a 2-link planar manipulator with link lengths l_1 and l_2.

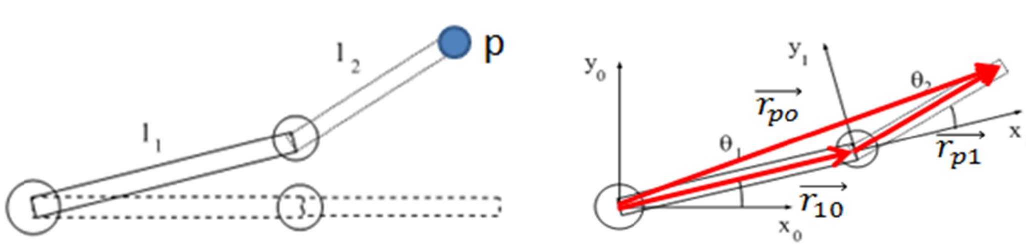

Figure B: 2-link planar manipulator (left) and with reference frames and rotations $\theta_1 \text{and} \theta_2$(right)

In lecture, the end-effector (EE) p has the position (x_{p0}, y_{p0}) given by:

$$\begin{aligned} x_{p0} &= l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \\ y_{p0} &= l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) \end{aligned} \tag{1}$$

Concept 1 Implement Forward Kinematics Equations xl320-2dof-fk-1_0.nxc

Figure 1A is the NXC code for implementing (1) for the 2-link planar manipulator. The code uses the previously created H-files `xl320-defines1_0a.h` and `xl320-functions1_0d.h` which define the constants and functions for using the XL-320 servo.

```
// FILE: xl320-2dof-fk-1_0.nxc - Works!
// DATE: 01/11/20 19:20
// AUTH: P.Oh
// DESC: Forward kinematics for 2-DOF planar manipulator using Dynamixel XL-320

#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 1A: Forward kinematics program **xl320-2dof-fk-1_0.nxc**

```
task main() {

  // 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;
  int i;
  // 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);
  // Second, user sets desired theta 1 and theta 2 here
  theta1InDegrees = 0.0; // [deg]
  theta2InDegrees = 90.0; // [deg]
  theta1 = theta1InDegrees * PI/180; // [rad]
  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]
  ClearScreen();
  TextOut(0, LCD_LINE1, "Will go to:" );
  TextOut(0, LCD_LINE3, FormatNum("xP0 = %3d studs" , xP0InStuds) );
  TextOut(0, LCD_LINE5, FormatNum("yP0 = %3d studs" , yP0InStuds) );
  // Prompt user to begin motion
  TextOut(0, LCD_LINE8, "Yes: hit ->");
  do {
     rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
  } while(!rightArrowButtonPushed);
  ClearScreen();

  rotateMotorAbsolutely(theta1InDegrees, theta2InDegrees);

  // Last, go back to home position and quit
  ClearScreen();
  TextOut(0, LCD_LINE2, "Back to Home" );
  Wait(2000);
  theta1InDegrees = theta2InDegrees = 0.0;
  rotateMotorAbsolutely(theta1InDegrees, theta2InDegrees);
  Wait(2000);
  PlaySound(SOUND_DOUBLE_BEEP);
} // end main
```

Figure 1A continued: `xl320-2dof-fk-1_0.nxc`

This particular 2-link planar manipulator uses two XL-320 servos; joints 1 and 2 have IDs `0x03` and `0x07` respectively and hence defined accordingly. Recall, the motivation to use LEGO stems from its standard 8 mm stud spacing. This spacing is universally used in all LEGO parts and hence provides a standard basis to calibrate lengths and assess positioning accuracies. As such the `mmPerStud` is also defined.

The links are affixed to the XL-320's horns with M2.5 screws. Before screwing these LEGO beams, they are oriented to align with the +X-axis. This means each XL-320 is centered (i.e. 512 [count]) as shown in Figure A (right). This orientation is defined to be zero degrees. The function `rotateMotorAbsolutely` is written to account for this 512 [count] offset as seen by the yellow-highlighted lines for `motor01Offset` and `motor01Offset`.

The main function defines the link lengths. This particular 2-link planar manipulator uses Technic Beams 7 and 5 for link lengths l_1 and l_2 respectively.

The Brick sets Port 4 for RS485 communications at 57,800 baud (8N1) and upon execution, prompts the user to push the right arrow button to commence. Once pushed, the XL-320 servos are commanded to their home position. For this example, home position is defined by having the two links aligned with the +X axis.

For this example, joint angles are set at $\theta_1 = 0.0$ [deg] and θ_2[deg] respectively. One can envision the result as shown in **Figure 1B**.

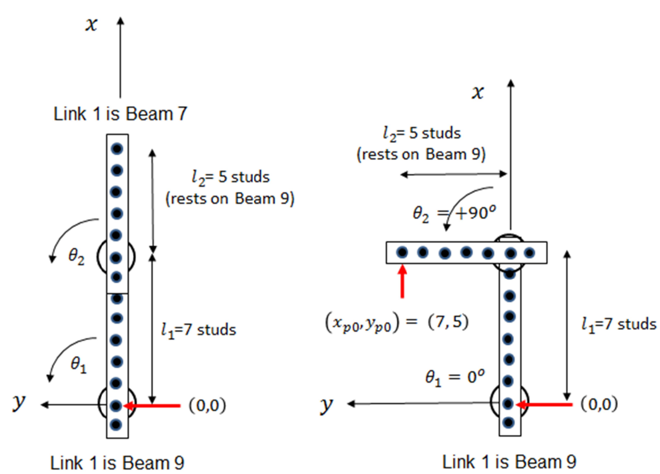

Figure 1B: 2-link planar manipulator in home position (left). Configuration after execution (right)

The lines below are the forward kinematics for the 2-link planar manipulator as given in (1). Note that NXC says cos and sin take radians as arguments.

```
xP0 = l1*cos(theta1) + l2*cos(theta1 + theta2); // [mm]
yP0 = l1*sin(theta1) + l2*sin(theta1 + theta2); // [mm]
```

The program displays the calculated EE position in stud values. Since these calculations are in float, the NXC `ceil` function rounds up to the nearest integer of studs. After reaching the EE position, the program then returns to the home position.

Congratulations! You implemented Forward Kinematics for the 2-link planar manipulator

Exercises

1.1 Edit **xl320-2dof-fk-1_0.nxc** to also display the EE's position in millimeters on the Brick

1.2 Calculate (1) by hand, execute program with the commanded angles and verify stud values to complete the table below

θ_1 [deg]	θ_2 [deg]	Equation (1) [studs]	Observed value [studs]
0	+90	(7, 5)	(7, 5)
0	-90		
+90	-90		
-90	-90		
+45	+45		

1.3 Unscrew and reverse the beams such that Link 1 is a Beam 7 and Link 2 is a Beam 9. Repeat 1.2 to complete a new table

1.4 Introduce an offset by add a L-shape 3 x 5 Liftarm (Part# 32526) between Links 1 and 2 (see figure below). Link 1 will still remain aligned with the +X axis but this Beam 5 causes Link 2 to be offset (but parallel) to the +X axis. Use DH notation to derive the resulting tool transformation matrix and complete a new table like in 1.2