Hands-on Lab

XL-320 NXC Programming – Inverse Kinematics

Forward kinematics seeks to calculate end-effector positions given joint positions. More common however is the opposite – calculate the joint configurations to move the end-effector to a desired position. This lab uses the LEGO base plate to define desired end-effector positions. Inverse kinematics is used to calculate the angles the 2-link planar manipulator must go to, to reach those desired positions.

Preliminary: 2-link Planar Manipulator and Inverse Kinematics

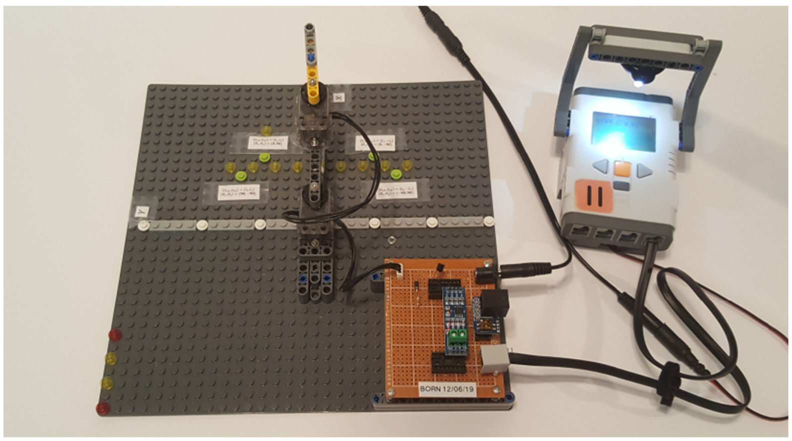

Figure A: The green colored 1-stud bricks mark desired end-effector positions. The XL-320 servos serve as Joints 1 and 2 of a LEGO-based 2-link planar manipulator.

Figure B shows a 2-link planar manipulator with link lengths l_1 and l_2. Recall that the inverse kinematics (IK) yielded the following equations.

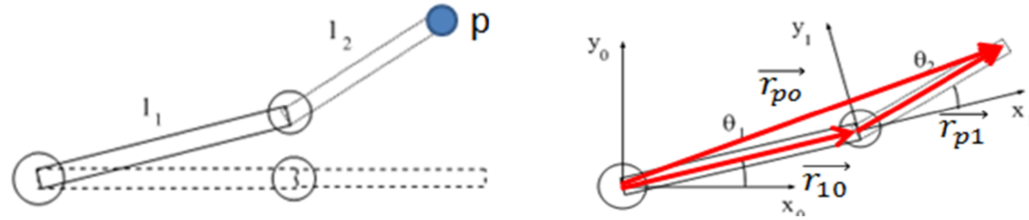

Figure B: 2-link planar manipulator (left) and with reference frames and rotations $\theta_1$andθ_2(right)

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

$$\theta_2 = atan2\left(\pm\left\{1 - \left(\frac{x_p^2 + y_p^2 - l_1^2 - l_2^2}{2l_1 l_2}\right)^2\right\}^{1/2}, \frac{x_p^2 + y_p^2 - l_1^2 - l_2^2}{2l_1 l_2}\right) \quad (1)$$

$$\theta_1 = atan2(y_p, x_p) - atan2(k_2, k_1) \quad (2)$$

$$k_1 = l_1 + l_2 \cos\theta_2 \qquad r_p^2 = k_1^2 + k_2^2 \qquad \gamma = \tan^{-1}\left(\frac{k_2}{k_1}\right) \qquad k_1 = r_p \cos\gamma \quad (3)$$

$$k_2 = l_2 \sin\theta_2 \qquad k_2 = r_p \sin\gamma$$

Concept 1 Implement Inverse Kinematics Equations xl320-ik-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. Note: much of this code follows `xl320-2dof-fk-1_0.nxc` developed in the previous forward kinematics lab.

```
// FILE: xl320-ik-1_0.nxc - Works!
// DATE: 01/16/20 09:12
// AUTH: P.Oh
// DESC: Inverse 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

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: Inverse kinematics program **xl320-ik-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 xCalibrate[5], yCalibrate[5]; // 4 (x,y) calibration points wrt x0y0 frame [mm]
  ArrayInit(xCalibrate, 0, 5); // initialize the (4x1) x vector with zeros
  ArrayInit(yCalibrate, 0, 5); // initialize the (4x1) y vector with zeros
  float xP, yP; // end-effector absolute position i.e. wrt x0y0 frame [mm]

  // 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
  // xCalibration[i] and yCalibrate[i] in [mm]
  // 90-degree calibration points easiest to envision end-effector location
  xCalibrate[0] = l1; yCalibrate[0] = l2;  // +'ve root: (theta1, theta2)=(0,90)
  xCalibrate[1] = l1; yCalibrate[1] = -l2; // -'ve root: (theta1, theta2)=(0,-90)
  xCalibrate[2] = l2; yCalibrate[2] = l1;  // -'ve root: (theta1, theta2)=(90,-90)
  xCalibrate[3] = l2; yCalibrate[3] = -l1; // +'ve root: (theta1, theta2)=(-90,90)
  // slightly harder to envision example
  // [mm] (theta1, theta2) will = +'ve root (8.7, 43.2) or -'ve (44.4, -43.2)
  xCalibrate[4] = 10 * mmPerStud; yCalibrate[4] = 5 * mmPerStud;

  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);

  // Next, go to desired points
  for(i=0; i<=4; i++) { // cycle thru the 4 calibration points and 5th point
      xP = xCalibrate[i];
      yP = yCalibrate[i];
      // pow function for power.  Using ^ is incorrect
      C = ( pow(xP,2)+pow(yP,2) - pow(l1,2)-pow(l2,2) ) / (2*l1*l2);
      if(i==0 || i==3) { // choose +'ve root
         num = sqrt(1-pow(C,2));
      } else { // use -'ve root for xPCalibrate[1], yPCalibrate[1] theta2 should be -90
         num = sqrt(1-pow(C,2));
      };
      theta2 = atan2(num, C); // [rad]
      theta2InDegrees = theta2 * 180/PI; // [deg]
      k1 = l1 + l2*cos(theta2);
      k2 = l2*sin(theta2);
      theta1 = atan2(yP, xP) - atan2(k2, k1); // [rad]
      theta1InDegrees = (theta1 * 180/PI); // [deg]

      // Actuate the XL-320 motors
      rotateMotorAbsolutely(theta1InDegrees, theta2InDegrees);
  }; // end for-loop
```

Figure 1A continued: Inverse kinematics program **xl320-ik-1_0.nxc**

```
  // Go back to home position
  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: Inverse kinematics program **xl320-ik-1_0.nxc**

To repeat the NXC code in **Figure 1A** is very much like the previous lab's code (`xl320-2dof-fk-1_0.nxc`). Thus only the key differences of **xl320-ik-1_0.nxc** will be described.

The main function initializes (see yellow highlight in Figure 1A) the link lengths l_1 and l_2 in millimeters. Also, arrays are used to simplify coding. Five desired points (called `xCalibrate` and `yCalibrate`) are defined. For the first four, the necessary joint angles can be easily visualized, and confirmed when the program runs. The last point is harder to visualize what joint angles are needed but during run-time, one can visually observe that the end-effector indeed reaches that point.

To make the code more readable, one observes in **Figure 1A**:

```
C = ( pow(xP,2)+pow(yP,2) - pow(l1,2)-pow(l2,2) ) / (2*l1*l2);
```

This implements:

$$\frac{x_p^2 + y_p^2 - l_1^2 - l_2^2}{2l_1 l_2}$$

which is a part of (1). The remaining yellow highlighted lines in Figure 1A implement (2) and (3).

The `for-loop` commands the XL-320 servos to the necessary joint angles for each desired calibration point. **Figure A** shows green-colored 1-stud bricks. These are fixed on the LEGO base plate for each `xCalibrate` and `yCalibrate` position. Running the code, the 2-link planar manipulator's end-effector should hover over these green-colored 1-stud bricks.

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

Exercises

1.1 Edit **xl320-ik-1_0.nxc** to use the negative square roots of (1). Calculate by hand, what the resulting `xCalibrate` and `yCalibrate` points should be, Affix green-colored 1-stud bricks at these points. Confirm your program indeed commands the 2-link planar manipulator's end-effector to hover over those points.

1.2 Unscrew and reverse the beams such that Link 1 is a Beam 7 and Link 2 is a Beam 9 and appropriately change in **xl320-ik-1_0.nxc**. Affix green-colored 1-stud bricks at these points. Confirm your program indeed commands the 2-link planar manipulator's end-effector to hover over those points.