Rebecca Schwartz
MEM 455 Fundamentals of Robotics
Professor James Tangorra
26 March 2019

Feedback Control with Optical Shaft Encoders

Demonstrate understanding of sensors used. Include what sensors were used, how they were used, and what the effects were.

Introduction

Sensors can be used to achieve more autonomous function of robots. Sensors can also increase a robot's precision and accuracy. In this project, optical shaft encoders are used to determine the angle of each revolute joint of the 2R robot. A feedback loop is initialized, allowing the robot to correct itself based on the desired angle and the actual angle. Since the shaft encoders used are only accurate to +- 4 degrees, a different approach is used to demonstrate the use of shaft encoders.

This project consists of a 2R robot which includes two servos and two optical shaft encoders (**Figure 1**). The motors and encoders are controlled by an Arduino Uno. In this project, inverse kinematics is used to output angles given a desired location of the end effector (**Appendix A**). Those angles may be inputted into Arduino and written to the servos, with an error. Then the feedback loop mentioned is initialized and any necessary corrections are made to the joint angles.

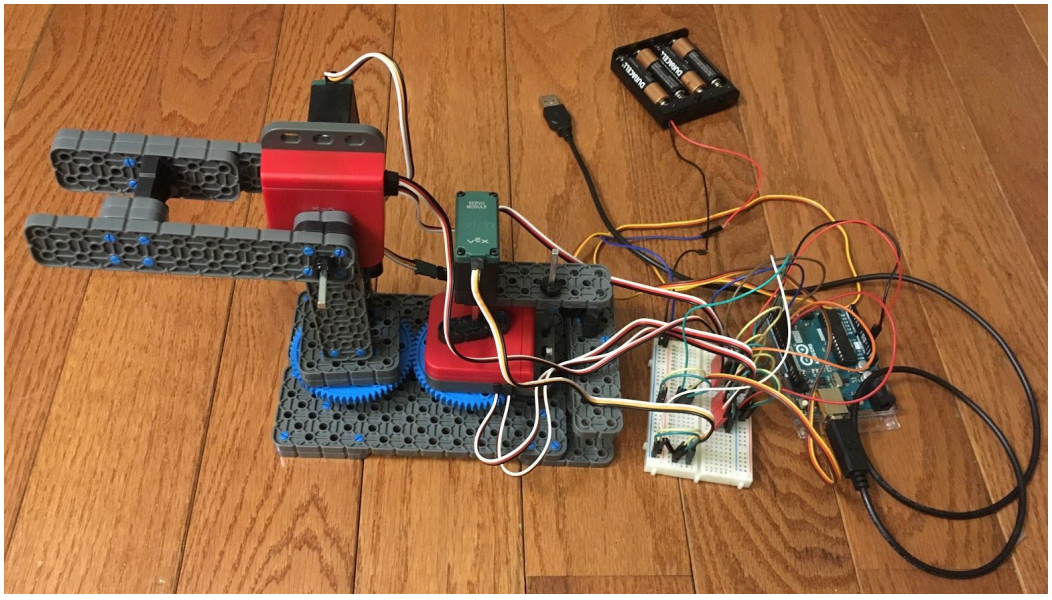

Figure 1: 2R Robot with Servos and Optical Shaft Encoders

Optical Shaft Encoders

Simple optical shaft encoders are excellent for sensing position and motion of shafts. A disk with slots in it is illuminated by an LED and detected on the other side of the disk with a light sensor. The light sensor will output a square wave as the disk rotates, that is, as the shaft rotates. A peak in the square wave (1) indicates that the LED light has been blocked by the disk. A trough in the square wave (0) indicates that the LED light has been sensed by the internal light sensor.

The shaft encoders used are considered to be quadrature optical shaft encoders. They can indicate position and direction of motion, which is useful for use with servos. These encoders output two square waves which are 90 degrees out of phase.

When the first square wave is 90 degrees ahead of the second square wave, that indicates that it is going in a particular direction. When the same square wave is 90 degrees behind the second square wave, then the disk is rotating in the opposite direction.

When the square waves continue, then the shaft is rotating in a consistent direction. This particular shaft encoder (Vex) has a disk with 90 slots in it. This means that the encoder can measure the angular precision with an accuracy of +- 4 degrees.

Arduino Code

When translated to code, the desired output of a shaft encoder is the angular position. Counterclockwise motion is determined by increasing angular position and clockwise motion is determined by decreasing angular position. The output of the shaft encoder will be in steps of 4 degrees.

A particular angle is selected. After going from the zero position, the servos reach that initial position. The encoders are set to reach another position however, which is very close to the initial position. This is the desired location and is characterized by the initial position +- an error. A proportional controller set in place in the Arduino code works to adjust the angular position until the position as read by the encoder is within 4 degrees of the desired position. At that point, the motors are written to stay in the current position. See **Appendix B** for the Arduino code.

Results

The following videos show the robot working as expected. The robot begins at the (0,0) position and then goes to an initial angle (90 deg for **Video 1** and 35 for **Video 2**) and then to the desired position (100 for **Video 1** and 45 for **Video 2**).

https://youtu.be/Dv65Dk4C5sY

Video 1: 2R Robot: 0 deg to Initial = 90 deg to Desired = 100 deg

https://youtu.be/Q-SBVs92rhA

Video 2: 2R Robot: 0 deg to Initial = 35 deg to Desired = 45 deg

Each video starts at joint positions (0,0). Then the initial position is reached, characterized by a short delay. Then the desired position is reached, which is characterized by small changes in the angular position until the robot rests at the desired position. As mentioned, the robot rests when the angle read by the shaft encoder is within 4 degrees of the desired angular position.

Discussion

The Arduino code provided does allow for feedback based on the desired joint angles and the current joint positions. Precision is problematic however with shaft encoders which are only accurate up to +- 4 degrees.

Another issue faced is that Arduino was not keeping up with the output measurements. Through testing, it was found that the Arduino was updating every 13 ms, which was found to be slower than was necessary to keep up with the changing joint angles. Slowing the motor speed down was completed via an external library.

To improve the performance of the robot, a more sensitive shaft encoder should be used. Additionally, a faster microprocessor should be used in order to allow the servos to move at their set speed. This would rid the need for an external library, which ultimately causes unwanted side effects.

Conclusion

Sensors are excellent for improved automation capabilities, precision, and accuracy of robots. The quadrature optical shaft encoder is an adequate choice for tracking position and direction of joint angles. When paired with servos, shaft encoders can be great tools for achieving more accurate joint angles.

Acknowledgments

Thank you Professor Tangorra for suggesting a workaround for this project and for your support.
Thank you Andy Drago for once again providing valuable insights.
Thank you Reed Cornish for assistance with cleaning up the Arduino code.

Appendix

Appendix A: MATLAB code for Inverse Kinematics

```
%Inverse Kinematics used to find desired joint angles given desired
%position of end effector

clear all
```

```
close all
clc

% length of links in m
L1 = 0.1143;
L2 = 0.1016;

Blist = [[0;0;1;0;0;0],[0;1;0;0;0;0]];
% zero config determined to be when robot arm is straight up
M = [[1, 0, 0, 0]; [0, 1, 0, 0]; [0, 0, 1, L1+L2]; [0, 0, 0, 1]];

% GIVEN COORDINATES

% test 0
% (L2,0,L1) -> 90 deg Ang2
T0 = [[0, 0, 1, -L1]; [0, 1, 0, 0]; [-1, 0, 0, -L2]; [0, 0, 0, 1]];
thetalist0 = [0; pi/4]; %rad
eomg = 0.01; %rad
ev = 0.01; %1mm
[thetalist0, success0] = IKinBody(Blist,M,T0,thetalist0,eomg,ev);

thetalist_pi_deg_0 = thetalist0*180/pi

% test 1
% (0,L2,L1) -> Ang1 = 90 deg, Ang2 = 90deg
T1 = [[0, -1, 0, -L1]; [0, 0, 1, 0]; [-1, 0, 0, -L2]; [0, 0, 0, 1]];
thetalist1 = [0; pi/3]; %rad
eomg = 0.01; %rad
ev = 0.01; %1mm
[thetalist1, success1] = IKinBody(Blist,M,T1,thetalist1,eomg,ev);

thetalist_pi_deg_1 = thetalist1*180/pi



% test 2
% (L2/sqrt(L2),0,L1-(L2/sqrt(L2)) -> Ang1 = 0 deg, Ang2 = 135 deg
T2 = [[-cosd(45), 0, cosd(45), -L1*cosd(45)]; [0, 1, 0, 0]; [-cosd(45), 0, -cosd(45),
-L2*cosd(45)]; [0, 0, 0, 1]];
thetalist2 = [0; pi/2]; %rad
```

```
eomg = 0.01; %rad
ev = 0.01; %1mm
[thetalist2, success2] = IKinBody(Blist,M,T2,thetalist2,eomg,ev);

thetalist_pi_deg2 = thetalist2*180/pi
```

MATLAB's OUTPUT

```
thetalist_pi_deg_0 =

     0
    90

thetalist_pi_deg_1 =

   90.0000
   90.0000

thetalist_pi_deg2 =

     0
   135
```

Appendix B: Arduino Code for Controlling Servos and Using Shaft Encoders

```
#include <VarSpeedServo.h>


const int SERVO_SPEED = 10;
int startTime = millis();

int enc1PinA = 6;
int enc1PinB = 7;
int enc1Pos = 0;

int enc2PinA = 4;
int enc2PinB = 5;
int enc2Pos = 0;
```

```
int servoPin1 = 3;
VarSpeedServo servo1;
const int ANGLE1_INIT = 0;
int servo1ang = 0;

int servoPin2 = 2;
VarSpeedServo servo2;
const int ANGLE2_INIT = 10;
int servo2ang = 45;

int servo1angGoal = 0;
int servo2angGoal = 55;

void setup() {
  pinMode (enc1PinA,INPUT);
  pinMode (enc1PinB,INPUT);
  pinMode (enc2PinA,INPUT);
  pinMode (enc2PinB,INPUT);
  Serial.begin(9600);
  servo1.attach(servoPin1);
  servo2.attach(servoPin2);
  servo1.write(ANGLE1_INIT, SERVO_SPEED, true);
  servo2.write(ANGLE2_INIT, SERVO_SPEED, true);
  delay(3000);
  Serial.print("Servo 1 Ang: 0");
  Serial.print(", Servo 2 Ang: 0");
  Serial.print(", Encoder 1: ");
  Serial.print(enc1Pos*4);
  Serial.print(", Encoder 2: ");
  Serial.print(enc2Pos*4);
  Serial.println("");
}

void loop() {
  Serial.println("Writing to Servos");
  servo1.write(servo1ang, SERVO_SPEED, false);
  servo2.write(servo2ang, SERVO_SPEED, false);

  watch(&enc1Pos, &enc2Pos, enc1PinA, enc1PinB, enc2PinA, enc2PinB);
```

```
  int enc1Ang = enc1Pos * 4;
  int enc2Ang = enc2Pos * 4;

  int diff1 = enc1Ang - servo1angGoal;
  if (abs(diff1) >= 4) {
   servo1ang = servo1ang - diff1;
  }
  int diff2 = enc2Ang - servo2angGoal;
  if (abs(diff2) >= 4) {
   servo2ang = servo2ang - diff2;
  }
}

int updateEncoder(int aBinLast, int aBin, int bBin, int encPos) {
  if (aBinLast == LOW && aBin == HIGH) {
   if (bBin == LOW) {
    encPos--;
   }
   else {
    encPos++;
   }
  }
  return encPos;
}

void watch(int *pEnc1Pos, int *pEnc2Pos, int enc1PinA, int enc1PinB, int enc2PinA, int
enc2PinB) {
  int enc1ABinLast = LOW;
  int enc1ABin = LOW;

  int enc2ABinLast = LOW;
  int enc2ABin = LOW;

  long msLastUpdate = millis();
  while (millis() - msLastUpdate < 1000) {
   enc1ABin = digitalRead(enc1PinA);
   int enc1BBin = digitalRead(enc1PinB);
   *pEnc1Pos = updateEncoder(enc1ABinLast, enc1ABin, enc1BBin, *pEnc1Pos);
```

```
    enc2ABin = digitalRead(enc2PinA);
    int enc2BBin = digitalRead(enc2PinB);
    *pEnc2Pos = updateEncoder(enc2ABinLast, enc2ABin, enc2BBin, *pEnc2Pos);

    Serial.print(*pEnc2Pos);
    Serial.print(", ");
    Serial.println(*pEnc2Pos * 4);

    if (enc1ABinLast != enc1ABin || enc2ABinLast != enc2ABin) {
      msLastUpdate = millis();
    }
    enc1ABinLast = enc1ABin;
    enc2ABinLast = enc2ABin;
  }
}
```