1. Year and Semester : 2013 SRING

2. Course Number : CS-492

3. Course Title : Kinect Programming

4. Work Number : LK-05

5. Work Name : Kinect Laboratory Report

6. Work Version : Version 1

7. Long Date : Wednesday, 3, April, 2013

8. Author(s) Name(s) : Jake Waffle

Chapter 6 – Using the Kinect for Robotics

Chapter Overview:

In this chapter I learned how the Kinect can be used along with an Arduino to create dependent programs that talk to each other through Processing's Serial library (which is for reading and writing data to and from external devices.) The chapter introduces how to do this by separating the different parts of the program into different sections. But I'll only be going over the crucial pieces that I used to create my modified project.

The chapter first goes over how to get the angle of the shoulder and elbow of the user with the Kinect (Borenstein pp 347-353). Then the program that operates the robotic arm is introduced and is what I used for sending the angles of the joints to the robotic arm from the Kinect's program (Borenstein pp 356-360). After that there is a test program that will send integer angles to the program that operates the robotic arm (Borenstein pp 360-362). Then the book goes over how to assemble the robotic arm (Borenstein pp 362-365). After that a program that demonstrates how the robotic arm works on the Kinect-side is introduced, but doesn't talk with the Arduino (Borenstein pp 367-371). And finally, the book shows how to make a program that controls the robotic arm with your head alone (Borenstein pp 372-378).

Building the Robotic Arm

Overview:

I did this before even reading the chapter, so I won't be referencing the book to explain this part. But the materials I used to put together the robotic arm were scotch tape, two pencils and two servos.

I'll go over the steps I took to get it together:

1. Tape a pencil to the servo that will represent the shoulder joint (use the part of the servo that rotates.) Keep in mind that the limb attached to the "shoulder" should be thought to be at a 90 degree angle with respect to the torso while the servo is off (Figure 1 shows my robotic arm.)
2. Then tape the second servo (the body of the servo specifically) to the pencil that was just attached to the first servo and this servo will represent the elbow. But when taping this servo on, remember to allow the forearm to have the full range of motion for the elbow correct.
3. Finally we can tape the "forearm" (the other pencil) of the robotic arm onto the second servo's rotating part. And if there is any confusion with how the arm's movements will look, then there will be a chance to test it out later in my report. But the book has a nice sequence of events to get the robotic arm built too (Borenstein p 363).

Robotic Arm Circuit

Overview:

When creating the circuit for the robotic arm, I also didn't really consult the book. That was because connecting servo's up to an Arduino is something I've already done before. Essentially each of the two servos needs to be connected to ground, 5volts and a pin that will allow analog outputs for the servo (Figure 1 can be referenced to see how I put it together.) That all can be done pretty easily, but if confusion appears the Kinect book goes over how to do it also (Borenstein pp 355-356).

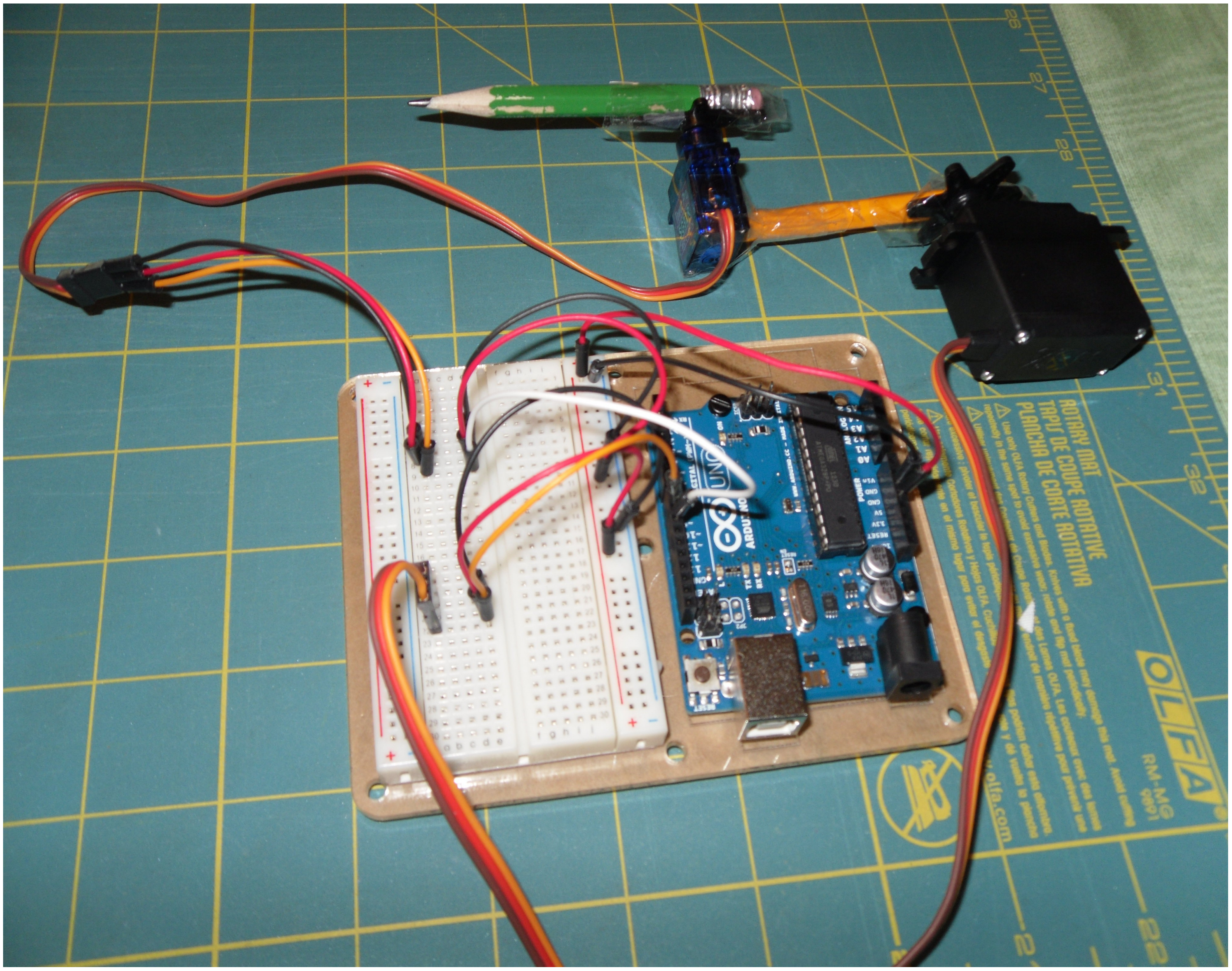


Figure 1. - This picture is of the robotic arm and the circuit that it uses.

Programming the Arduino

Overview:

This part of the report will cover the section in the book that goes over how to setup the robotic arm's program so that it will receive the angles for the servos and alter the robotic arm accordingly.

I'll explain how the code works:

1. First copy the code from the book (Borenstein pp 357-358).
2. Then we're going to look it over and analyze it.
3. There are four variables that are being used in the global section of the program. The first two are for talking with the two servos. The third one is used to differentiate between which of the two servos is going to have their angle updated next (because only one servo gets updated every loop() iteration.) And the last variable is an array that stores the angles for the two servos.
4. Within the setup() function, the Servo data types get their pins assigned to them (the pins that the Servos on the circuit are connected to.) And Serial is instructed to begin (which is what is used to relay information from the Kinect's program to this program.)
5. Then within the loop() function, as long as Serial has data that can be read, we will read that data (integer) and update one of the Servo's angle (depending on which Servo was updated last time.) After that the variable that determines which Servo is updated next will be changed. And independently to all of that, the two Servos get their angles sent to them each loop() iteration.

(Optional) Testing the Robotic Arm

Overview:

Since by now the robotic arm is put together from the arm itself to the program that operates it, we can now go about testing the arm to see if it is working the way it should be. This can be done simply by running a program from the book that uses the regular Processing environment (Borenstein p 361). This program essentially will look to see if there is another program using Serial, then if there is it will create a port to that program that will allow the programs to talk to each other (but the talking will only be in one direction in this example.) Then depending on the key that is pressed the test program will send different hard-coded angles to the Arduino's program.

Before running the test program, the Arduino's program should already be uploaded onto the Arduino and the Arduino should still be connected to the computer. Then there shouldn't be any worries when running the test program. But if there are multiple Serial ports active, then a different port name may need to be grabbed from the Serial.list() array (that is done within the setup() function.)

Inverse Kinematics

Overview:

This section in the book basically demonstrates some math that can work for determining the user's arm position. The code snippet for this section uses the law of cosines to calculate the angle of the elbow based off of the three joints in the arm (Borenstein pp 368-369). But the angle of the shoulder is calculated using some geometry (that assumes the torso doesn't tilt in any direction.)

The shoulder and elbow position aren't directly alterable with the Kinect yet in the program for this section. The mouse is just able to control the position of the hand and the elbow moves according to how far away the mouse is from the shoulder. The shoulder is also at a fixed position.

Controlling Our Robot Arm with Inverse Kinematics

Overview:

This section's code snippet literally builds off of the preceding section's code (Borenstein pp 374-376). But it combines the Arduino program that operates the robotic arm, the robotic arm, the two previous code snippets and the Kinect (since the “Inverse Kinematics” section doesn't use the Kinect.) This program essentially moves the robotic arm around based off of where your head is (the shoulder is fixed and the elbow changes based on the position of the “hand.”)

What is Different in the Code:

The math for calculating the position of the shoulder and elbow are the same as from the Inverse Kinematics section. The way a Serial port is created for talking with the Arduino is the same as in the test program. The Arduino program that operates the robotic arm is the same as it is in the “Programming the Arduino” section of the book (Borenstein p 357.) The robotic arm should be setup and ready. So the only real change to this program is the addition of the Kinect and the tracking of the head to use in place of the mouse for the “Inverse Kinematics” section's part of the code.

Closing Thoughts:

After going through preceding code snippets, this program seemed as though it was already explained throughout the entire chapter (which really helps in understanding the code in my opinion.) And the parts that weren't explained are pretty easy to figure out as long as you're familiar with skeleton tracking.

Complete Control of the Robotic Arm

Overview:

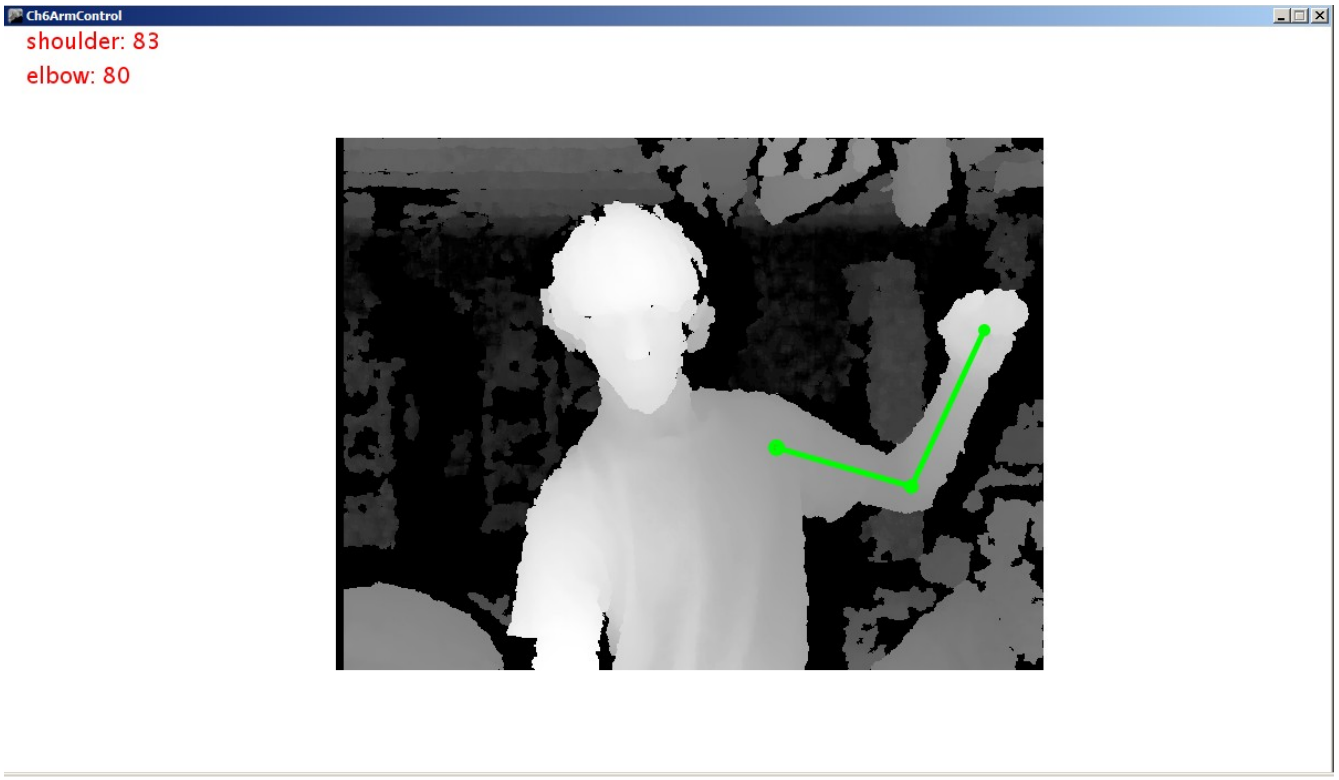
Here I'm going to explain my modification to the last code snippet shown in Chapter 6 (Borenstein pp 374-376). It essentially uses the same basic concepts, controlling the robotic arm using the angles calculated from an imaginary 2-dimensional arm. Except instead of creating an arm that is controlled through indirect means, the 2-dimensional arm is controlled by the arm of the user itself (using the user's hip, shoulder, elbow and hand joints.) It is easily possible to do this because the prior program was already setup to work with a shoulder and elbow joint, but they were calculated imaginary joints. So I just had to keep updating those joint positions based off of what the SimpleOpenNI library tells me about the user's joints.

What I Modified:

1. The first modification is the control of the robotic arm using the shoulder, elbow and hand joints positions of the user.
2. Second, the virtual limb for the robotic arm that is drawn in the window has been altered to be drawn over the user's right arm (this can be seen below in Figure 2.)
3. Finally, I made it so that the previous position of the robotic arm is saved so that the angles of the joint use interpolation to smooth out the movement of the arm (it was jittery because of the inaccuracies.)

What Went Wrong:

I had some troubles with the calculation of the shoulder joint from the “Inverse Kinematics” section. Whenever the elbow moved, the angle of the shoulder would also move. And to fix this, I created my own calculation for the angle of the shoulder using the law of cosines (the calculation uses the right hip, shoulder and elbow.) But I could have used the functions used in the code snippet from the beginning of Chapter 6 in order to do this.



shoulder: 83
elbow: 80

Figure 2 – This is a picture of how my modified program looks in the window.

Now I'll go over the changes to the code in the code directly (the comments using “///” will explain what has changed in the program and highlighted parts will denote a change):

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

import processing.serial.*;
Serial port;

///These PVectors are crucial for the interpolation modification. They save the previous positions of the
///        joints needed for the angle calculations of the shoulder and elbow.
PVector lastRHand, lastRHip, lastRElbow, lastRShoulder;

void setup(){
  size(1200,800);

  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
  kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);
  kinect.setMirror(true);
  ///Here the variables for interpolation are simply being initialized to position (0, 0).
  lastRHand = new PVector(0,0);
  lastRHip = new PVector(0,0);
  lastRShoulder = new PVector(0,0);
  lastRElbow = new PVector(0,0);
```

```
 println(Serial.list());
 String portName = Serial.list()[0];
 port = new Serial(this, portName, 9600);
}


void draw(){
 background(255);
 kinect.update();
 image(kinect.depthImage(), 300, 100);


 IntVector userList = new IntVector();
 kinect.getUsers(userList);


 if (userList.size() > 0){
   int userId = userList.get(0);
   if(kinect.isTrackingSkeleton(userId)){
     ///The following four blocks of code are all for getting the position of the joints needed for the
     ///  elbow. And the position is converted so that it is with respect of the 2d depth map image.
     PVector rShoulder = new PVector();
     kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER, rShoulder);
     kinect.convertRealWorldToProjective(rShoulder, rShoulder);

     PVector rElbow = new Pvector();
     kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW, rElbow);
     kinect.convertRealWorldToProjective(rElbow, rElbow);
```

```
    PVector rHand = new PVector();
    kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HAND, rHand);
    kinect.convertRealWorldToProjective(rHand, rHand);

    //This joint is needed for the shoulder angle calculation
    PVector rHip = new PVector();
    kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HIP, rHip);
    kinect.convertRealWorldToProjective(rHip, rHip);


    ///Here I'm using some temporary PVector variables to store the interpolated position of the joints,
    ///   so that the other two sets of variables (the previous and current positions) can update themselves
    ///   for the next draw() iteration.
    PVector interHand = new PVector();
    PVector interHip = new PVector();
    PVector interShoulder = new PVector();
    PVector interElbow = new PVector();










    ///Here is where the 2d positions of the joints are interpolated between the last position and the
    ///   position that is currently being fetched from SimpleOpenNI.
    interHand.x = lerp(lastRHand.x, rHand.x, 0.5f);
    interHand.y = lerp(lastRHand.y, rHand.y, 0.5f);
    interHip.x = lerp(lastRHip.x, rHip.x, 0.5f);
    interHip.y = lerp(lastRHip.y, rHip.y, 0.5f);
    interShoulder.x = lerp(lastRShoulder.x, rShoulder.x, 0.5f);
    interShoulder.y = lerp(lastRShoulder.y, rShoulder.y, 0.5f);
    interElbow.x = lerp(lastRElbow.x, rElbow.x, 0.5f);
    interElbow.y = lerp(lastRElbow.y, rElbow.y, 0.5f);

    ///Here the interpolated position is going to overwrite the “actual” position of the user's arm
    rHand = interHand;
    rHip = interHip;
    rShoulder = interShoulder;
    rElbow = interElbow;

    ///Then the position of the joints are going to be saved off into global variables so that they can
    ///   be used for interpolation next draw() iteration.
    lastRHand = rHand;
    lastRHip = rHip;
    lastRShoulder = rShoulder;
    lastRElbow = rElbow;


    //begin the kinematics math
    PVector difference = PVector.sub(rHand, rShoulder);
```

```
float distance = difference.mag();


//Sides of main triangle

//Calculate the distance between the elbow and right shoulder

float a = PVector.sub(rElbow, rShoulder).mag();


//Length between elbow and hand

float b = PVector.sub(rElbow, rHand).mag();


float c = min(distance, a + b);


///This is where the triangle for calculating the shoulder's angle is calculated. The sides are made

///  using the right hip, shoulder and elbow.

//Sides for the secondary triangle

//The length between the shoulders.

float d = PVector.sub(rShoulder, rHip).mag();


//The length between the elbow and right hip

float e = min(PVector.sub(rElbow, rHip).mag(), a + d);






//angles of the main triangle via law of cosines

float B = acos((a*a + c*c - b*b)/(2*a*c));
```

```
    float C = acos((a*a + b*b – c*c)/(2*a*b));


    ///This is where the shoulderAngle is calculated basically.

    //Angle of the secondary triangle via law of cosines

    float E = acos((a*a + d*d - e*e)/(2*a*d));


    ///These were altered so that they stay within the range of 0 and 180 and also work for the robotic

    ///   arm in the way that I made it (which can be seen in Figure 1.)

    float shoulderAngle = constrain(degrees(E), 0.0, 180.0);

    float elbowAngle = map(constrain(degrees(C), 0.0, 180.0), 0, 180, 180, 0);


    fill(255,0,0);

    textSize(20);

    text("shoulder: " + int(shoulderAngle) + "\nelbow: " + int(elbowAngle), 20, 20);


    byte out[] = new byte[2];

    ///Here is where this program talks to the Arduino program through the Serial port in order to send

    ///   the position of the joints needed to operate the robotic arm. The difference isn't in the code, but

    ///   in the fact that the angles represent the actual angles of the user's arm.

    out[0] = byte(int(shoulderAngle));

    out[1] = byte(int(elbowAngle));

    port.write(out);


    fill(255,0,0);

    ///Here I used the position of the user's joint to draw circles on the imaginary limb's joints.
```

```
      ellipse(rShoulder.x + 300, rShoulder.y + 100, 10, 10);

      ellipse(rElbow.x + 300, rElbow.y + 100, 8, 8);

      ellipse(rHand.x + 300, rHand.y + 100, 6,6);

      stroke(0,255,0);

      strokeWeight(5);

      ///And here I used the position of the user's joint to draw lines between the imaginary limb's joints.

      line(rShoulder.x + 300, rShoulder.y + 100, rElbow.x + 300, rElbow.y + 100);

      line(rElbow.x + 300, rElbow.y + 100, rHand.x + 300, rHand.y + 100);

    }

  }

}


void onNewUser(int userId){

  println("start pose detection!!!!!!");

  kinect.startPoseDetection("Psi", userId);

}


void onEndCalibration(int userId, boolean successful){

  if(successful){
    println("User calibrated!!!");
    kinect.startTrackingSkeleton(userId);
  }
  else{
    println("Failed to calibrate user!");
    kinect.startPoseDetection("Psi", userId);
  }
}

void onStartPose(String pose, int userId){
  println("Started pose for user");
  kinect.stopPoseDetection(userId);
  kinect.requestCalibrationSkeleton(userId, true);
}
```

References

-Borenstein, G (2012). Making Things See. Sebastopol,California: O'Reilly Media.

-Adel Karabegovic. (undefined). . In IndieGameMusic. Retrieved February 19, 2013, from http://indiegamemusic.com/viewtrack.php?id=1150.