***Inverse Kinematics***

*for*

***A Robot Arm with Hand to reach a target***

I had no idea what inverse kinematics (IK) was before this assignment. One of the tasks my team wants for our Robot Arm with Hand is to be able to shake hands with someone standing in front of it. We had planned to place a proximity sensor on the hand and use a search method to discover objects in front of it. When the search is complete, we have arrived at our target (a human hand extended to shake). However, this is not a natural process for shaking hands. After researching IK I realized that this method mimics the natural motion someone uses to shake hands, so it will be perfect for use in our project.

**Initial Idea: PID Method**

The first idea I had was to use an error correction method such as PID to reduce the error between where the robot is and the target. Very quickly it became apparent that coordinating the different degrees of freedom (DOF) was not only difficult, but there were no unique solutions in many cases.

Algorithm for PID method:

* 2-3 servos move in unison
	+ Move servo 1 small distance toward goal
	+ Move servo 2 small distance toward goal
	+ Etc
	+ Repeat until hand is at target

The problem with the above algorithm is there are possible positions where one servo would need to move away from the target to allow another servo to move. This is an example of local minima prohibiting the realization of a goal. I had to simplify the model and create a mathematical method (such as a computer program) to input the values and output the solution.

**A better solution: IK**

Simplifying the model to 2 DOF (shoulder, elbow) we are able to bypass some of the trickier math involving transforms and translations and use the Law of Cosines with a simple triangle to find the angles needed to place the hand at the target. Here is an example of the Arm:



And here is the geometric representation with a desired target:



From the Law of Cosines we have:

$$c^{2}=a^{2}+ b^{2}- 2ab cos⁡(c)$$

Inserting our values and solving for the desired angles gives us:

$$θ\_{2}=arccos\left[\frac{x^{2}+ y^{2}- L\_{1}^{2}-L\_{2}^{2}}{2L\_{1}L\_{2}}\right]$$

And

$$θ\_{1}= arcsin\left[\frac{L\_{2}sin⁡(θ\_{2})}{\sqrt{x^{2}+ y^{2}}}\right]+ arctan2\left(\frac{x}{y}\right)$$

Using the above angle formulas and some input values, a simple c++ program was written to calculate the output angles needed to place the hand at the target. Below is an example of the program running. Note that the angle calculations are converted from radians to degrees by multiplying by ($^{180}/\_{π})$. When the function is integrated into our control program the degrees will be converted to the pulse width necessary to move the servos to the desired angles.

There are some problems with this method for our use, but they can be resolved with logic. The $θ\_{1}$ value is not unique; the solution can be in the first quadrant or fourth quadrant and be a viable solution. Therefore some solutions are not natural solutions for humanoid movement (most people are unable to bend their elbows both directions). Some logic needs to be introduced to not allow certain combinations of angles.

Also, if the target is out of range a decision needs to be made to: not move the arm until it is in range; have the arm extended toward the target as it approaches. A more human-like method would be to wait until the target (human hand) is within range before moving. Adding these behavior decisions will be included in the fuzzy logic homework.



As can be seen below, the solutions do necessarily represent desired angles:



This is the code for the above program:

#include <iostream>

#include <string>

#include <math.h>

using namespace std;

// Three points in space: base, elbow, hand

float \_points\_x [3] = {0, 0, 0};

float \_points\_y [3] = {0, -500, -1000};

//private var \_distance:Number = 100;

int \_distance = 0; //millimeters, assume 1m arm reach:(upper arm)=(lower arm)=500 mm

int armLength = 500;

// Give a starting target in millimeters

float targetX = 250;

float targetY = 250;

//arm angles start straight, ie hanging down

float \_theta1 = 0;

float \_theta2 = 0;

float RAD\_CONVERT = (180 / 3.14159);

void solve()

{

 // Compute difference between start and end points

 float dx = (targetX - \_points\_x[0]);

 float dy = (targetY - \_points\_y[0]);

 // Compute distance from hand starting position to target points

 float dx\_hand = (targetX - \_points\_x[2]);

 float dy\_hand = (targetY - \_points\_y[2]);

 //cout << "Difference of (x, y): " << "(" << dx << ", " << dy << ")" << endl;

 cout << "Location of base (x, y): " << "(" << \_points\_x[0] << ", " << \_points\_y[0] << ")" << endl;

 cout << "Location of elbow (x, y): " << "(" << \_points\_x[1] << ", " << \_points\_y[1] << ")" << endl;

 cout << "Location of hand (x, y): " << "(" << \_points\_x[2] << ", " << \_points\_y[2] << ")" << endl;

 // Compute distance between start and end points

 \_distance = sqrt(dx\*dx + dy\*dy);

 cout << endl << "Distance from base to target: " << \_distance << endl;

 // Compute distance from hand to target

 float hand\_distance = sqrt(dx\_hand\*dx\_hand + dy\_hand\*dy\_hand);

 cout << "Distance from hand to target: " << hand\_distance << endl;

 // Compute angle between start and end points

 float theta = atan2(dy,dx);

 float temp = (theta \* RAD\_CONVERT);

 cout << "Angle between base and target: " << temp << endl;

 // Clamp the distance

 float totalLength = armLength \* 2;

 if( \_distance < totalLength ) {

 // Calculate first angle:

 \_theta1 = acos( \_distance / totalLength ) + theta;

 dx = dx - \_distance \* cos( \_theta1 );

 dy = dy - \_distance \* sin( \_theta1 );

 // Calculate second angle from first angle and segment

 \_theta2 = atan2(dy, dx);

 } else {

 // If the distance is greater than arm length, arm is straight

 \_theta1 = \_theta2 = int(theta);

 }

 // Compute positions from angles and update joint positions

 \_points\_x[1] = \_points\_x[0] + cos( \_theta1 ) \* armLength;

 \_points\_y[1] = \_points\_y[0] + sin( \_theta1 ) \* armLength;

 \_points\_x[2] = \_points\_x[1] + cos( \_theta2 ) \* armLength;

 \_points\_y[2] = \_points\_y[1] + sin( \_theta2 ) \* armLength;

 cout << endl << "Post-Calculation values:" << endl;

 cout << "Location of base (x, y): " << "(" << \_points\_x[0] << ", " << \_points\_y[0] << ")" << endl;

 cout << "Location of elbow (x, y): " << "(" << \_points\_x[1] << ", " << \_points\_y[1] << ")" << endl;

 cout << "Location of hand (x, y): " << "(" << \_points\_x[2] << ", " << \_points\_y[2] << ")" << endl;

 float temp1 = (\_theta1 \* RAD\_CONVERT);

 float temp2 = (\_theta2 \* RAD\_CONVERT);

 cout << "Theta 1 is " << temp1 << endl << "Theta 2 is " << temp2 << endl;

}

int main()

{

 int xVal, yVal, qVal;

 cout << "This program will calculate the angles needed to" << endl;

 cout << "move a 2 DOF robot hand to a target point in a 2D" << endl;

 cout << "(x,y) grid. Each segment of the robot arm is 500" << endl;

 cout << "mm long and its starting position is hanging down" << endl;

 cout << "from the origin (0,0) to (0,-1000) at the hand." << endl << endl;

 int loop = 1;

 while (loop == 1){

 cout << "Enter the X-coordinate (mm) for the target:" << endl;

 cin >> xVal;

 cout << "Enter the Y-coordinate (mm) for the target:" << endl;

 cin >> yVal;

 targetX = float(xVal);

 targetY = float(yVal);

 cout << "Your target coordinates are (" << xVal << ", " << yVal << ") in mm" << endl;

 solve();

 cout << "New target from the current position? (enter 1 = yes)" << endl;

 cin >> qVal;

 if (int(qVal) == 1) {

 loop = 1;

 qVal = 0;

 }

 else {

 loop = 0;

 }

 }

}

**Conclusions:**

In this homework I have learned the concepts of inverse kinematics and created a function to solve a 2 DOF robot arm problem. This can be incorporated into our program design for our control logic of our Robot Arm. My project partner has been working on a sensor array to calculate the location and distance of an object in front of the arm.

Once we calibrate the distance and the servo angles, the above program will allow us to smoothly emulate a shaking Robot Arm with Hand. We need to keep the servo angles within range of natural humanoid movement. In further homework we will use search algorithms to find targets and fuzzy logic to determine our Robot Arm responses.

**References:**

**Braunl, Thomas, "EMBEDDED ROBOTICS. Mobile Robot Design and Applications with Embedded Systems" Springer, 2008**

**Luger, George, “Artificial Intelligence” Addison-Wesley, 2008**

[www.lorenzonuvoletta.com/tag/inverse-kinematics](http://www.lorenzonuvoletta.com/tag/inverse-kinematics)

<http://generalrobotics.org/ppp/Kinematics_final.ppt>

<http://freespace.virgin.net/hugo.elias/models/m_ik2.htm>