Robot Arm Project

By Greg Flynn, Craig Lombardo and Austin Wiles

# Introduction

Microcontrollers are small computers capable of interacting with electric circuits. The arduino is a revolutionary device; it is a microcomputer that draws virtually no power compared to a full blown computer and it is very easy to integrate into electric circuits compared to a full computer. The robot arm we were working with is similar to robots found on the floor of factories, performing repetitive, dull or dangerous tasks. Our arm wasn't very precise but there are arms out there that are incredibly precise.

# Purpose of lab

The lab taught us how to program arduino based software using C, a language none of us were very familiar with. Also we learnt the concepts of PWM and how to use it to control a servo with precision. Another aspect of the lab that taught us a lot was the trigonometry for calculating all of the angles. The purpose, or takeaway from this project, was to learn how to actively control multiple changing components at once to complete a task.

# Requirements

The requirements of this lab was to control 5 servos to work together in order to navigate around and move blocks. The goal of this lab was to pick up 4 blocks from pre-specified positions and to stack them all at a single location. This robot is a small scale version of robots that work on assembly plants helping to manufacture new products. In this document we will discuss how we implemented the autonomous movement of the arm and tested it safely.

# High-level Design

The first thing we did was establish our final objective; we wanted to be able to give the robot arm any x, y, z and claw state values and have the robot do that. The robot arm was programmed so we would provide four values, the x, y and z positions as well as the state of the grip. The arm would then convert these values into certain angles for the servos. We used trigonometry to calculate all of the required servo angles. We chose to fix the wrist angle relative to the horizon since we had to fix an angle to get a unique solution. We then stepped the servos until they were in the correct position.

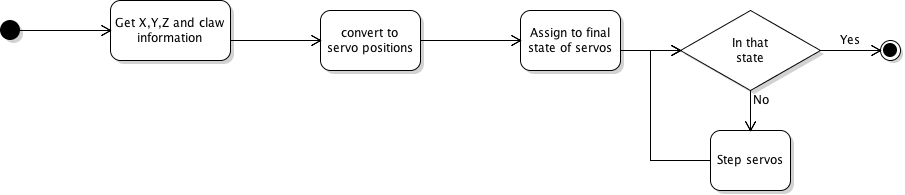


Figure 1 - High level flow

# Detailed Design

In this section means servo setting and means angle relative to the horizon.

## Auto level function

To provide one unique solution we had to fix an angle on the robot arm. The most logical angle to fix was the robot wrist angle. Instead of keeping the angle fixed relative to the previous arm section we kept it flat to the horizon. The algorithm for setting the servo position for the wrist is based off the angles of the other servos.

Wrist level is the parameter to specify what angle the wrist should be compared to the horizon.

## Smoothing function

The smoothing functions objective was to slowly move a servo from one position to another. The first thing the function does is to check how many degrees the servo needs to turn. From there it checks if it needs to move clockwise or anticlockwise. It then enters a for loop for the number of steps and moves the servo by 1 degree and waits 20ms between each step. This gives the illusion of the robot arm moving smoothly.

## Algorithm for motion[[1]](#footnote-1)

We used trigonometry to find the shoulder and elbow angles. (0,0,0) is located at the base of the robot. First we calculated the radial distance with the Pythagorean theorem on the x and y components.

Then we did found the arctan of the x and y to get the base angle.

This lets us convert from Cartesian coordinates into cylindrical. Since the wrist angle is pre defined relative to the horizon the values of delta distance and delta height can be calculated.

Using these values we can calculate where the robot elbow and shoulder should be

By fixing the wrist angle we have one unique solution rather than a set of solutions for the servo angles.

Since two sides of triangle b are known and triangle b is a right angle triangle the hypotenuse can be calculated using the Pythagorean theorem.

And part of the shoulder angle can be calculated

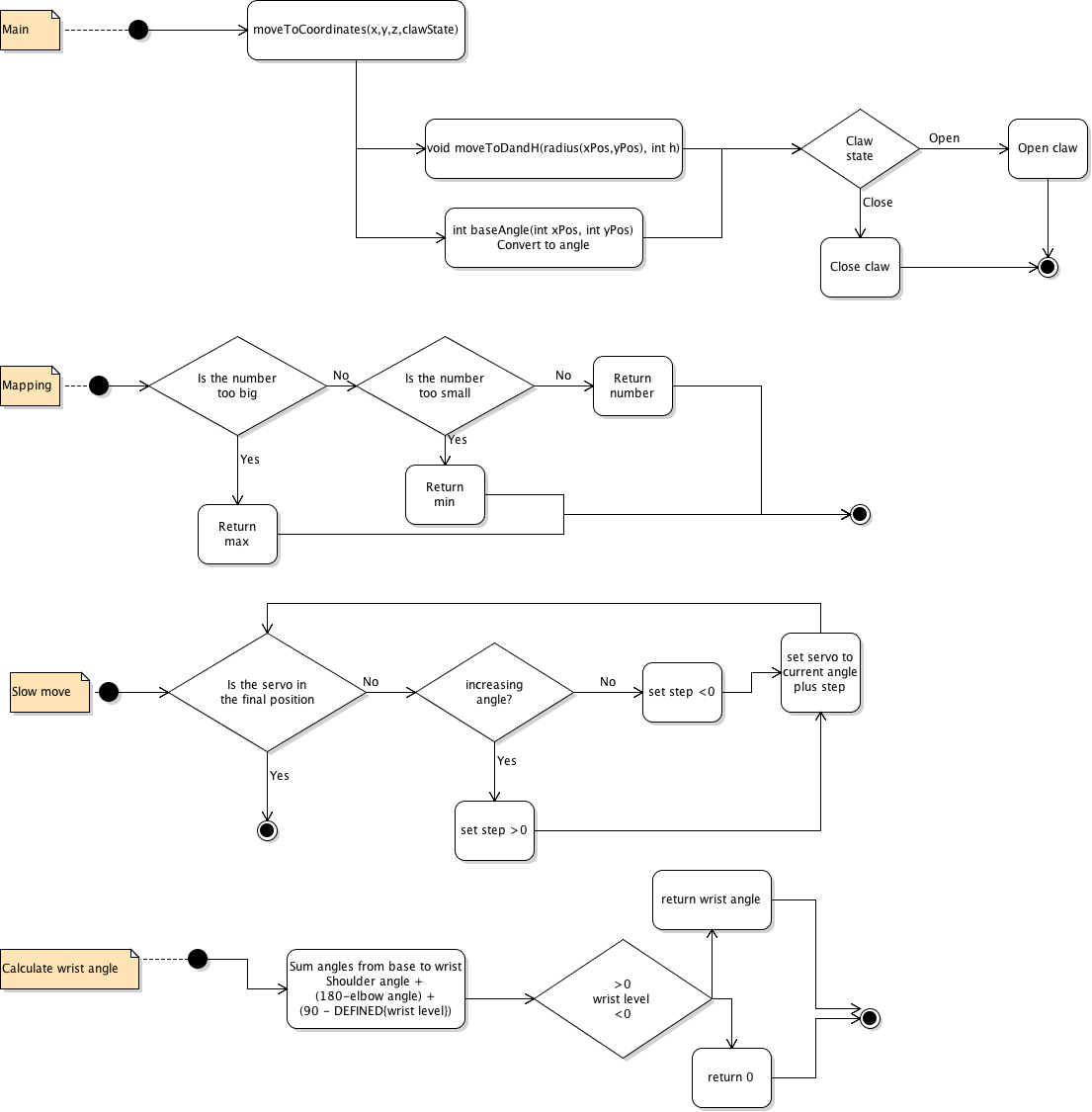
Using B3 the elbow angle can be calculated by the law of cosines

Also another component of the shoulder angle (c1) can be calculated using the sine law

The elbow angle can now be calculated

# Software Design

Our final objective was to have a function that accepts 4 parameters, x,y and z co-ordinates and a Boolean specifying if the claw is closed. This enabled us to move the arm from any position on the grid to any other. While moving the arm would first move up to ensure that no collision with objects on the grid would happen. We then modified each servo from the base to the wrist in that order. This was because the servos closest to the base cause larger movements; if we fine tuned first then moved the shoulder then the blocks would be knocked over.



# System Testing

## Testing the leveling function

To test the robot wrist angle we reverted back to the code that controlled the robot arms with servos but removed the wrist from the servo control. This let us quickly test a variety of different angles and visually inspect what was happening.

## Radial angle

To test the base angle function was working we did something similar to the wrist leveling program; we removed control of the base from the potentiometers to the microcontroller. Once that was done we programmed the robot to pick up the blocks. Since the servos that control height were not functional the robot couldn't actually pick up any blocks but it could line up above the blocks. We visually inspected the robot to ensure that the base angle worked. We held a ruler in line with the robot arm to ensure that the line between the claw position and the base went through the coordinates we specified.

## Simulation

We implemented a java based virtual robot arm which accepted three parameters for the angles of the servo. The simulator only showed two dimensions, the radial distance and the height. We felt like that was adequate since we already had solved the base angle. By having a simulator we were able to check our math without causing unnecessary damage to the robot due to it slamming into the ground.

## Final test

Once we proved that each individual angle worked we set up the robot to stack and un-stack the blocks. This let us check not only that the robot worked but that it was consistent. Also we didn't need to supervise it as intensely since it was repeating the same task over and over again. This proved to be a very effective way of fine tuning the robot arm to ensure that it could keep stacking the blocks repeatedly.

# Conclusion

We completed all primary objectives of the assignment; our robot arm was capable of picking up the blocks and stacking them. We also took the project one step further since we have our arm capable of going to any position on the grid with the change of a single parameter. Also we have a simulator of the robot arm which reduces the damage cause on the arm due to poor logic or coding technique. The largest unsolved issue was the accuracy of the servos on the arm. The best way to improve this would be to have a feedback wire from the servos to know what position they actually went to. Moreover this would allow us to check if the robot arm had crashed into the ground.

# Appendix

Introduction: A paragraph or two that explains the purpose of the project, which was to programming a robot arm to stack blocks. State your intended audience for the document, which for this lab would be another student at your level who needs to implement the system. Feel free to explain why microcontrollers and robots are important and get the reader interested. Also, briefly explain what to expect in the remainder of the document.

1. High-Level Design: Describe your design from a high level. Draw a simplified block diagram of the system and explain what you are trying to accomplish. State any requirements or assumptions you had for your design. If there were multiple ways to do the design, explain why you chose your particular approach.

2. Detailed Design: Include one or more sections that describe the internal details of the blocks in the high-level design. For example, if you used trigonometry to position the arm, you could have a section called "Algorithms." Another section could be labeled "Servo Control," etc. The description given in these sections should be sufficiently detailed that someone could understand what you did and implement it themselves.

3. Software Design: Explain how your code works. For example, you could show the basic series of operations with a flow chart. You can also refer to different functions in your code, but I recommend you do not bore the reader with your full code, but rather put it in an appendix.

4. System Testing: Describe the test cases you used, which should be sufficient to convince your reader that the system is working.

5. Conclusion: Give some parting words to reader summarizing what was done in the project and whether it was successful or not. Briefly explain outstanding things that would have to be looked at in the future if you were not completely successful or discovered unexpected things.

6. Appendix: Here you could put a listing of your code or any information you feel is too detailed (or boring) for the main project description.

# Appendix A: Robot arm diagram

## Colored triangles

Cyan triangle: triangle a

Yellow triangle: triangle b

Green triangle: triangle c

## Naming convention:

a# = angle # of triangle a

A# = length opposite angle # of triangle a

White lines: robot pieces

Red angles: servo angles drawn from 0 to current angle

## Direction of rotation

Shoulder angle CW

Elbow angle CW

Wrist angle CW

## Values provided by user

radial distance (calculated from x and y)

height (z)

Note wrist angle is absolute, not local. It is relative to the horizon not the previous arm.

# Appendix B: Source code

#include <Servo.h>

#include <Math.h>

/\*

\* The value D and H are the absolute length

\* d and h are the position of the wrist before the final movement

\*/

Servo myservo[5];

int pin[] = {3,5,6,9,10};

String names[] = {"Base", "Shoulder","Elbow","Wrist","Hand"};

int minimum[] = {36,10,54,18,89};

int maximum[] = {153,108,174,94,100};

int currentAngle[] = {95,65,130,60,110};

int resetAngle[] = {95,65,130,60,110};

const int wristLevel = 65; // this is the angle between the wrist and the horizon

#define PI 3.1415

void setup(){

for(int i = 0; i<5;i++){

myservo[i].attach(pin[i]);

}

Serial.begin(9600);

}

void loop(){

for(int i=0; i < 5; i++){

myservo[i].write(resetAngle[i]);

currentAngle[i] = resetAngle[i];

Serial.println("here");

}

delay(1000);

moveToCoord(202,168,-68,true);

moveToCoord(108,-123,-77,false);

moveToCoord(268,0,-66,true);

moveToCoord(112,-123,-56,false);

moveToCoord(250,100,-66,true);

moveToCoord(115,-126,-34,false);

moveToCoord(254,-104,-66,true);

moveToCoord(116,-127,-10,false);

moveToCoord(268,0,0,false);

//unstack

moveToCoord(119,-130,-10,true);

moveToCoord(252,-100,-66,false);

moveToCoord(115,-126,-34,true);

moveToCoord(250,100,-66,false);

moveToCoord(114,-125,-56,true);

moveToCoord(266,0,-66,false);

moveToCoord(108,-123,-77,true);

moveToCoord(199,165,-69,false);

moveToCoord(268,0,0,false);

}

void moveToCoord(int x, int y, int z, boolean closed){

slowMove(0,baseAngle(x,y));

delay(100);

moveToDandH(sqrt(x\*x+y\*y),z);

Serial.println("done");

if(closed && currentAngle[4] != 85) slowMove(4,85); // close grip

else if(!closed && currentAngle[4] != 110) slowMove(4,110); // open grip

delay(500);

moveUp();

}

void slowMove(int servoNum, int moveTo){

int steps = moveTo-currentAngle[servoNum];

boolean incr = steps > 0;

steps = steps > 0 ? steps : -steps;

for(int j=0; j < steps; j++){

int add = incr ? j : -j;

myservo[servoNum].write(currentAngle[servoNum]+add);

Serial.print(names[servoNum]);

Serial.println(currentAngle[servoNum]+add);

delay(20);

}

Serial.println("done");

currentAngle[servoNum] = moveTo;

}

int moveUp(){

slowMove(1,100);

slowMove(2,130);

}

/\*\*

\* Value 2 is the range of the output values

\*/

int mapping(int value, int low2, int high2){

bool outOfBoundsU = value > high2;

bool outOfBoundsL = value < low2;

if(outOfBoundsU) {

return high2;

}

if(outOfBoundsL) {

return low2;

}

return value;

}

int baseAngle(int xPos, int yPos){

int angle = atan2(yPos,xPos) \* 180 / (3.14192);

return 96 + angle;

}

int radius(int xPos, int yPos){

return sqrt((xPos\*xPos)+(yPos\*yPos));

}

/\*\*

\* To calculate the height and distance we need many cosine moves

\* The connection between the shoulder and the elbow is called A

\* The connection between the elbow and the wrist is called B

\* The connection between the wrist and the end is called C

\*/

const int A = 153;

const int B = 120;

const int C = 175;

// Angle 0 is shoulder

// Angle 1 is elbow

// Angle 2 is wrist

void moveToDandH(int d, int h){

int angles[3];

double newDist = (d-cos(degToRad(wristLevel))\*C);

double newHeight = (h+sin(degToRad(wristLevel))\*C);

angles[0] = 180-getCalculatedShoulderAngle(newDist,newHeight);

angles[1] = getTheta(newDist,newHeight);

angles[2] = getWristAngle(angles[0],angles[1]);//mapping(getWristAngle(),minimum[3],minimum[3]);

for(int i=0; i < 3; i++){

slowMove(i+1,angles[i]);

}

}

// This sets the wrist to be at 45 degrees to the horizon

int getWristAngle(int sAngle, int eAngle){

int wAngle = 270 - sAngle - eAngle - wristLevel; //- sAngle + (180-eAngle) + (90-wristLevel);

wAngle = wAngle < 0 ? 0 : wAngle;

return wAngle;

}

/\*\*

\* This method will move the wrist orgin to d and h

\* It uses the cosine rule on the hypotonouse between

\* d and h

\*/

int getElbowAngle(int d, int h){

int cosValue = ((A\*A+B\*B-(d\*d+h\*h))/(2\*A\*B));

return radToDeg(acos(cosValue));

}

double getL(int distance, int myHeight){

return sqrt(distance\*distance+myHeight\*myHeight);

}

double getTheta(double distance, double myHeight){

double lVal = getL(distance,myHeight);

double constants = (lVal\*lVal -(A\*A + B\*B));

double cosTheta = (constants/((-2)\*A\*B));

double theta = acos(cosTheta)\*(180/PI);

return theta;

}

double getThetaOne(double distance, double myHeight){

double lVal = getL(distance,myHeight);

double theta = getTheta(distance,myHeight);

double numerator = B\*sin(theta\*(PI/180));

double thetaOne = asin(numerator/lVal)\*(180/PI);

return thetaOne;

}

int getCalculatedShoulderAngle(double distance, double myHeight){

double thetaOne = getThetaOne(distance, myHeight);

double thetaH = atan2(myHeight,distance)\*(180/PI);

double thetaS = (180 - thetaOne - thetaH);

return thetaS;

}

double radToDeg(int rad){

return rad\*180/PI;

}

double degToRad(int deg){

return deg\*PI/180;

}

1. See Appendix A for diagram of robot arm [↑](#footnote-ref-1)