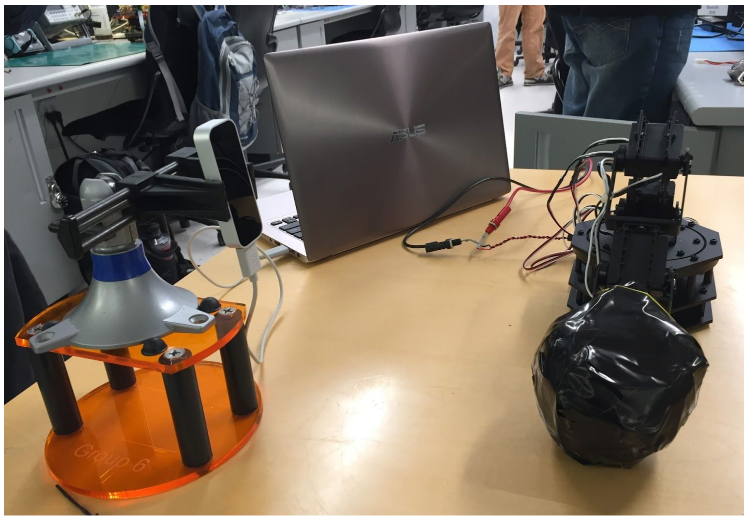
Akash Levy and Bink Sitawarin (Bench 306)
ELE 302
Independent Project Writeup

Brobot

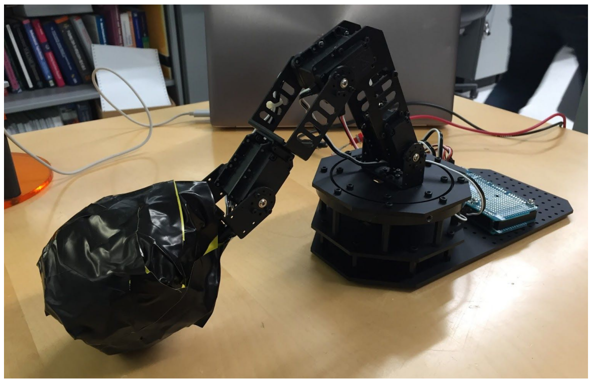
A fist-bumping robot who's a bro

Hardware Design

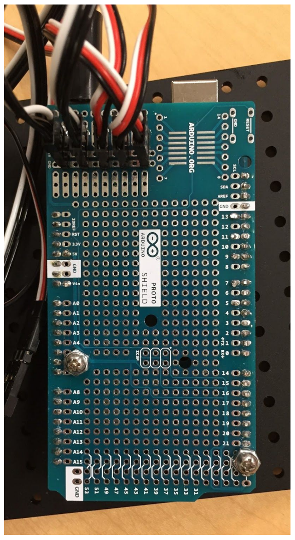
The setup of Brobot (power supply not shown)



RobotGeek Snapper Core Arduino Robot Arm



Arduino Mega 2560 Rev3 with Proto Shield



Leap Motion

Brobot consists of five main hardware components:

1. RobotGeek Snapper Core Arduino Robot Arm

The robotic arm has four degrees of freedom and is able to open and close its gripper. It uses five servo motors in total to control four joints (Base, Shoulder, Elbow, Wrist) and the gripper (which is unused in our project). Each servo can rotate up to 180 degrees with PWM duty cycle ranging from 600 - 2400 ms. Following the instructions provided on the website, we assembled the arm from the parts we got in the box. We decided to wrap the gripper in many layers of napkin followed by a final layer of black duct tape. The reason we did this is that the gripper is made of mildly sharp plastic which would scrape our hands when we fistbumped it. The black duct tape was also necessary to help the Leap Motion focus on tracking our hand and ignore the robot fist. Originally, we used black and yellow tape, but the tape was sometimes recognized as being part of a hand by the Leap Motion. Black tape worked better because it blended in with the background. The overall effect of the napkins and the black tape was to make the robotic arm look like a fist inside of a boxing glove. Each servo on Brobot had three wires connecting it to the Arduino, one for input voltage (7V, 1A), one for ground and one for a PWM signal (0-5V).

2. Arduino Mega 2560 Rev3 with Proto Shield

The Arduino is used as the main connection between the laptop and the arm. It receives the desired hand position in 3D space via USB from the host laptop, performs inverse kinematics computations to obtain the corresponding arm servos' degree of rotation, and then sends five PWM signals to the servos. We use the Proto Shield to map the Arduino's output pin to a correct servo (pin 5 to Base, pin 7 to Shoulder, pin 9 to Elbow, pin 11 to Wrist and pin 13 to Gripper). The Arduino is powered by a 7V-5A input source generated from a power supply. We also set up the wires so that all five servos are powered by the same source used to power Arduino. Setting up the Proto Shield required us to use some clever tricks; since it is designed for use with specific Arduino components, we had to hack it by resoldering some of the connections on the board and cutting pins off. It was one of the most challenging parts of the project.

3. Power Supply

We used the DC voltage power supply that was capable of providing 3A per channel; however, each servo requires a maximum of 1A. For all joints of the arm to move simultaneously and smoothly, we needed a power supply of at least 5A. Otherwise, only some of the servos would move at the same time, and then the rest will begin to move after, creating a discrete, jerky movement. We solved this problem by setting two channels of our power supply to be in parallel with each other providing 7V 3A. The net result was a 7V power supply that could provide up to 6A of current.

4. Leap Motion

The Leap Motion is a motion sensor specialized in hand tracking. We originally planned on using the Kinect for hand tracking, but switched to the Leap Motion since it is specifically designed for what we wanted to do. Using two cameras and three infrared LEDs, the Leap Motion can accurately track multiple hand and arm movements in real time with very low latency. The device needs to be used with a computer, and it has a well-documented library that supports multiple languages. We used the Leap Motion to detect whether a hand is present, then tracked the position of it in a 3D space. We also registered a hand gesture, which we used to start the system (a screen tap). The Leap Motion is powered by the laptop through a USB cable which is also used to send the data. We mounted the Leap Motion on a circuit board holder, which was placed on top of the car stand. The Leap Motion needs to be positioned high enough that it can track a hand above the robot arm.

5. Laptop

A laptop is required for receiving and processing data from the Leap Motion and sending the desired hand position to the Arduino. The laptop runs a C++ program that reads frames from the Leap Motion connected via a USB port, extracts the hand position, transforms it to a desired robot arm position in a correct coordinates and sends the information to the Arduino via a USB Serial port with baud rate set to 230400 Hz.

A 10-second Snapchat demo is available at: https://youtu.be/7P67wSFvnMc
Our code is available at: https://github.com/akashlevy/Brobot

Software Design

Our software consisted of two components, one to process sensor input and one to control the arm. The sensor input processor communicates with the arm via a serial connection. We chose to use serial communication for its minimalism, lack of overhead, and ease of use. Both Arduino and C++ with Linux support serial communication and have relatively clean interfaces.

Arduino Program

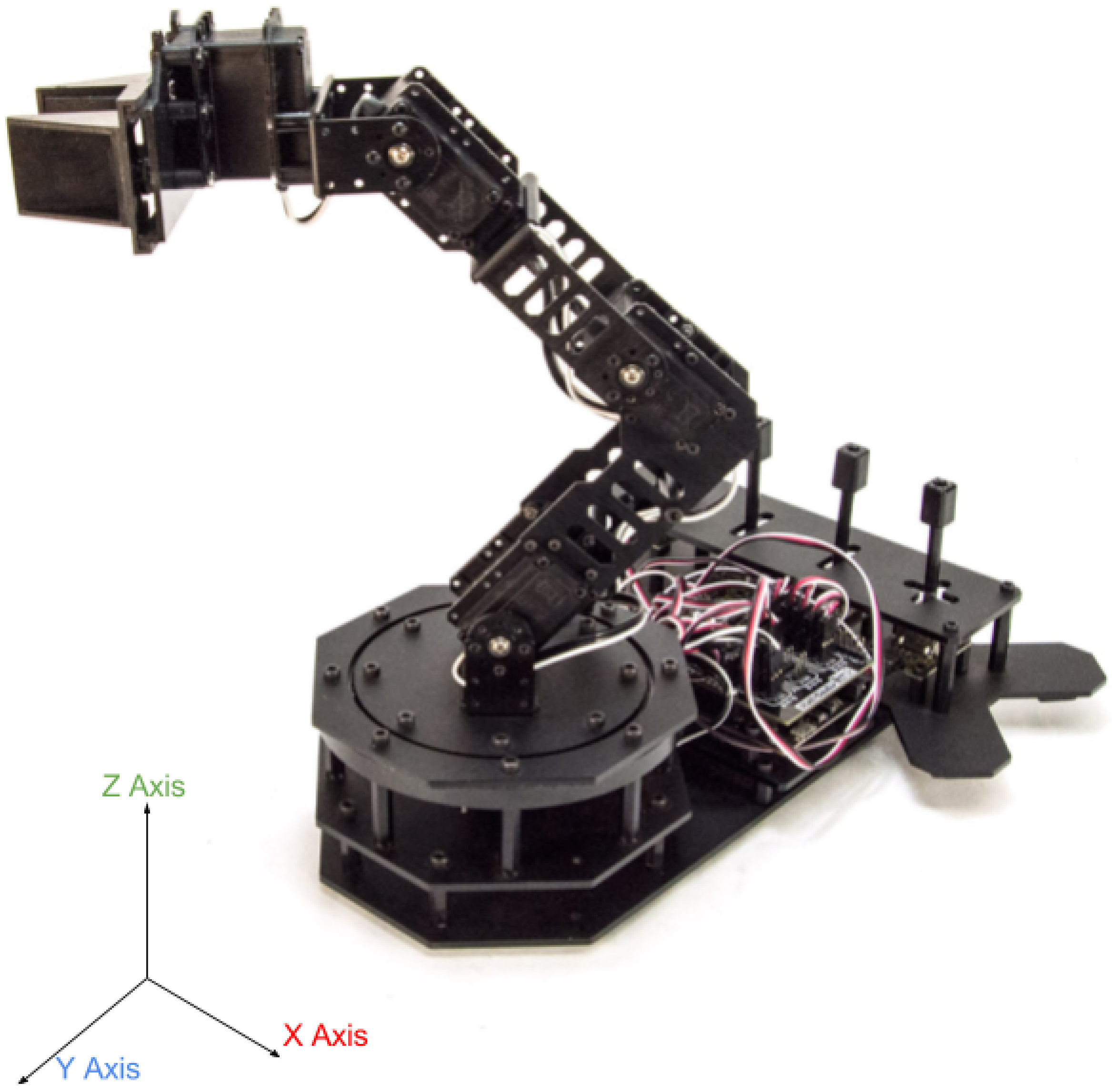
For the Arduino, we adapted code from the RobotGeek code repository for performing inverse kinematics. Their code involved the use of joystick sensors for moving the arms and allowed three different modes: cartesian, cylindrical, and backhoe. Most of this code was unnecessary for our purposes. We only needed cartesian mode and had no need for built-in sensors at all (we only wanted to interface with the Leap Motion). We ended up eliminating most of it. Our refactored code was quite short and intuitive, and this refactoring allowed us to quickly reverse engineer it and understand how it could be modified to suit our purposes.

The inverse kinematics function `doArmIK` in our code allowed us to map a desired arm location to servo angles. This required very accurate values for the lengths of the joints which were provided by RobotGeek. Below are the lengths used in the code:

Base Height	88.75 mm
Shoulder-to-Elbow (Humerus) Length	96.00 mm
Elbow-to-Wrist (Ulna) Length	90.60 mm
Wrist Gripper Length	96.00 mm

There were also limits for both the servo duty cycles and the coordinates to be mapped to. This was a safety precaution to prevent the motors/metal components from being damaged from unbounded values that might result from errors or misuse.

The inverse kinematics code uses simple equations to deterministically map specified (X, Y, Z) coordinates to joint positions. The variables/equations/parameters used, as well as the coordinate system for the arm, are given below:



Input variables	**Output variables**	**Parameters**
θ_G : grip angle	θ_B : base angle	H_B : base height
x (see figure above)	θ_S : shoulder angle	H_{SE} : shoulder-elbow length
y (see figure above)	θ_E : elbow angle	H_{EW} : elbow-wrist length
z (see figure above)	θ_W : wrist angle	H_{WG} : wrist-gripper length

Inverse Kinematics Equations:

$$\theta_B = atan2(x, y)$$

$$r = \sqrt{x^2 + y^2}$$

$$gripoffset_z = H_{WG}\, sin(\theta_G)$$
$$gripoffset_y = H_{WG}\, cos(\theta_G)$$

$$wrist_z = (z - gripoffset_z) - H_B$$
$$wrist_y = (r - gripoffset_y)$$

$$shouldertowrist = \sqrt{{wrist_y}^2 + {wrist_z}^2}$$

$$\theta_S = atan2(wrist_z, wrist_y) + acos(((H_{SE}{}^2 + H_{EW}{}^2) + shouldertowrist^2) \,/\, (2\, H_{SE}\, shouldertowrist))$$

$$\theta_E = acos(((H_{SE}{}^2 + H_{EW}{}^2) - shouldertowrist^2) \,/\, (2\, H_{SE}\, H_{EW})) - \pi$$

$$\theta_W = \theta_G - \theta_E - \theta_S$$

Once the inverse kinematics equations produced the angles, these angles were converted to servo pulses using simple linear mappings.

We then wrote a serial port receiver that reads the target cartesian coordinates as comma-separated values. The program continuously reads bytes from this port at a baud rate of 230400 Hz and calls `doArmIK` to update the arm's position. We read into a buffer before processing, and used newlines to separate commands. We also used a linear mapping of height to gripper angle to make a more fistbump-like motion when the arm moves up and down. The high baud rate of 230400 Hz was necessary for minimizing latency in our system. Our original baud rate of 9600 Hz had about a 1 second delay between the command and the action. At 230400 Hz, the latency was virtually unnoticeable, making for a very smooth user experience.

We tested our receiver using Arduino's built-in serial communication tool. Ultimately, we used this receiver to interface with the C++ client, which sent commands based on the Leap Motion's sensor input.

C++ Program

Primarily, the C++ program has four different operational steps. First, it continuously reads frames sent from the Leap Motion through its USB port. Second, it extracts the hand coordinate. Third, it transforms the coordinate into a correct space. Fourth, it writes the data to the serial port that is read by the Arduino. We accomplish the above steps by writing a Leap Motion Listener, adapted from example code given on Leap Motion tutorial website. The Listener contained functions that are called on various events: `onInit, onConnect, onDisconnect, onFrame, onFocusGanined, onFocusLost,` etc. The most important event here is `onFrame`. `onFrame` is called every time a frame is captured and received from the Leap Motion. A `Frame` object can then be accessed inside the function through a `Controller` object. `Frame` contains all the information captured by the Leap Motion including arm position, hand position, hand direction as well as each finger position and direction. It also keeps track of a hand and assign an id to it.

Inside `onFrame`, initially, the program starts off in an “off” state and keeps looking for a screen tap gesture. Once a detected hand performs the gesture, it triggers the “on” state and then starts tracking hand position. If the hand leaves the frame for 3 seconds, the program will revert to the “off” state and move the robot arm back to its original position. Once the program is in the “on” state, it extracts a `Vector` object containing 3D coordinate of the hand from a `Hand` object inside `Frame`. Then, the desired robot arm coordinate is given by the following set of equations:

$$x_{robot} = y_{hand} - 250 \quad (1)$$
$$y_{robot} = -z_{hand} + 80 \quad (2)$$
$$z_{robot} = x_{hand} + 250 \quad (3)$$

Note that these equations depend on the orientation of the Leap Motion with respect to the robot arm. The constant offsets can also be manually calibrated.

Once the coordinates have been calculated, the program proceeds to write them to the serial communication stream that is opened at the beginning of the program using a C++ file abstraction. The coordinate is written as comma-separated and newline-terminated string. The Arduino then parses the string back into the coordinate (three floats).

Code

ARDUINO CODE:

ServoOutPos.ino

```
#include "ServoEx.h"
#include "Kinematics.h"

ServoEx ArmServo[5];

//===============================================================================
// Setup
//===============================================================================
void setup(){
 // Attach servo and set limits
 ArmServo[BAS_SERVO].attach(5, BASE_MIN, BASE_MAX);
 ArmServo[SHL_SERVO].attach(7, SHOULDER_MIN, SHOULDER_MAX);
 ArmServo[ELB_SERVO].attach(9, ELBOW_MIN, ELBOW_MAX);
 ArmServo[WRI_SERVO].attach(11, WRIST_MIN, WRIST_MAX);
 ArmServo[GRI_SERVO].attach(13, GRIPPER_MIN, GRIPPER_MAX);

 // Initialize servo position
 doArmIK(0, 185, 185, 0);
  // Start serial
 Serial.begin(230400);
}

// Define number of pieces
const int numberOfPieces = 3;
String pieces[numberOfPieces];

// This will be the buffered string from Serial.read() up until \n
// Should look something like "123,456,789,"
String input = "";

// Keep track of current position in array
int counter = 0;
```

```
// Keep track of the last comma so we know where to start the substring
int lastIndex = 0;

void loop(){
 // Check for data coming in from serial
 if (Serial.available() > 0) {

   // Read the first byte and store it as a char
   char ch = Serial.read();

   // Do all the processing here since this is the end of a line
   if (ch == '\n' || ch == '\r') {
     for (int i = 0; i < input.length(); i++) {
       // Loop through each character and check if it's a comma
       if (input.substring(i, i+1) == ",") {
         // Grab piece from last index up to the current position and store it
         pieces[counter] = input.substring(lastIndex, i);
         // Update last position and add 1, so it starts from next character
         lastIndex = i + 1;
         // Increase the position in the array that we store into
         counter++;
       }

       // If we're at the end of the string (no more commas to stop us)
       if (i == input.length() - 1) {
         // Grab the last part of the string from the lastIndex to the end
         pieces[counter] = input.substring(lastIndex, i);
       }
     }

     // Update the position
     float x = pieces[0].toFloat();
     float y = pieces[1].toFloat();
     float z = pieces[2].toFloat();
     float ga = map(z, 0, 500, -30, 90);
     doArmIKLimits(x, y, z, ga);

     // Clear out string and counters to get ready for the next incoming string
     input = "";
```

```
      counter = 0;
      lastIndex = 0;
    }
    else {
      // If we havent reached newline yet, add current character to string
      input += ch;
    }
  }
}
```

GlobalArm.h

```
#ifndef GLOBALARM_H
#define GLOBALARM_H

// Motion limits (mm)
#define IK_MAX_X 150
#define IK_MIN_X -150

#define IK_MAX_Y 200
#define IK_MIN_Y 50

#define IK_MAX_Z 260
#define IK_MIN_Z 80

#define IK_MAX_GA 30
#define IK_MIN_GA -30

// Arm dimensions (mm)
#define BASE_HGT      88.75   // base height
#define HUMERUS       96.00   // shoulder-to-elbow "humerus bone"
#define ULNA          90.6    // elbow-to-wrist "ulna bone"
#define GRIPPER       96      // wrist-gripper length


//============================================================================
// SERVO CONFIG
//============================================================================
// Declare servos
enum { BAS_SERVO, SHL_SERVO, ELB_SERVO, WRI_SERVO, GRI_SERVO };

// Servo position limitations - limits in microseconds
#define BASE_MIN      600     // full CCW for RobotGeek 180 degree servo
#define BASE_MAX      2400    // full CW for RobotGeek 180 degree servo
#define SHOULDER_MIN  600
#define SHOULDER_MAX  2400
#define ELBOW_MIN     600
#define ELBOW_MAX     2400
#define WRIST_MIN     600
#define WRIST_MAX     2400
#define GRIPPER_MIN   750    // fully closed
#define GRIPPER_MAX   2400   // fully open

// Define servo offsets in +/- uS
#define BAS_SERVO_ERROR 0 // (+ is CW, - is CCW)
#define SHL_SERVO_ERROR 0 // (+ is forward, - is backward)
#define ELB_SERVO_ERROR 0 // (+ is up, - is down)
#define WRI_SERVO_ERROR 0 // (+ is up, - is down)
#define GRI_SERVO_ERROR 0 // (+ is tighten grip, - is loosen grip)

#endif
```

Kinematics.h

```
#ifndef KINEMATICS_H
#define KINEMATICS_H

#include "GlobalArm.h"
#include <Arduino.h>



//=============================================================================
// Global Variables
//=============================================================================

extern ServoEx ArmServo[5];
```

```
//================================================================================
// Miscellaneous Definitions
//================================================================================

// Float-to-long conversion
#define ftl(x)  ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))

// Pre-calculations
float hum_sq = HUMERUS*HUMERUS;
float uln_sq = ULNA*ULNA;

// Old/Current IK values
float            sIKX  = 0.00;
float            sIKY  = 150.00;
float            sIKZ  = 150.00;
float            sIKGA = 0.00;

//================================================================================
// doArmIK: Floating Point Arm IK Solution for PWM Servos
// Arm positioning routine utilizing inverse kinematics
// z is height, y is distance from base center out, x is side to side
// y,z can only be positive
//================================================================================
void doArmIK(float x, float y, float z, float grip_angle_d) {
 // Grip angle in radians for use in calculations
 float grip_angle_r = radians( grip_angle_d );

 // Base angle and radial distance from x,y coordinates
 float bas_angle_r = atan2( x, y );
 float rdist = sqrt(( x * x ) + ( y * y ));

 // rdist is y coordinate for the arm
 y = rdist;

 // Grip offsets calculated based on grip angle
 float grip_off_z = ( sin( grip_angle_r )) * GRIPPER;
 float grip_off_y = ( cos( grip_angle_r )) * GRIPPER;
```

```
// Wrist position
float wrist_z = ( z - grip_off_z ) - BASE_HGT;
float wrist_y = y - grip_off_y;

// Shoulder to wrist distance
float s_w = ( wrist_z * wrist_z ) + ( wrist_y * wrist_y );
float s_w_sqrt = sqrt( s_w );

// Shoulder to wrist angle to ground
float a1 = atan2( wrist_z, wrist_y );

// Shoulder to wrist angle to humerus
float a2 = acos((( hum_sq - uln_sq ) + s_w ) / ( 2 * HUMERUS * s_w_sqrt ));

// Shoulder angle
float shl_angle_r = a1 + a2;
float shl_angle_d = degrees( shl_angle_r );

// Elbow angle
float elb_angle_r = acos(( hum_sq + uln_sq - s_w ) / ( 2 * HUMERUS * ULNA ));
float elb_angle_d = degrees( elb_angle_r );
float elb_angle_dn = -( 180.0 - elb_angle_d );

// Wrist angle
float wri_angle_d = ( grip_angle_d - elb_angle_dn ) - shl_angle_d;
// Servo pulses
float Base = (ftl(1500.0 - (( degrees( bas_angle_r )) * 10.55 )));
float Shoulder = (ftl(1500.0 - (( shl_angle_d - 90) * 10.55 )));
float Elbow = (ftl(1500.0 + (( elb_angle_d - 90.0 ) * 10.55 )));
float Wrist = (ftl(1500 + ( wri_angle_d  * 10.55 )));

// Move the servos
ServoGroupMove.start();
ArmServo[BAS_SERVO].writeMicroseconds(Base + BAS_SERVO_ERROR);
ArmServo[SHL_SERVO].writeMicroseconds(Shoulder + SHL_SERVO_ERROR);
ArmServo[ELB_SERVO].writeMicroseconds(Elbow + ELB_SERVO_ERROR);
ArmServo[WRI_SERVO].writeMicroseconds(Wrist + WRI_SERVO_ERROR);
ServoGroupMove.commit(0);
```

```
 // Save old values
 sIKX  = x;
 sIKY  = y;
 sIKZ  = z;
 sIKGA = grip_angle_d;
}

void doArmIKLimits(float x, float y, float z, float grip_angle_d) {
 // Restrict range
 x = max(min(x, IK_MAX_X), IK_MIN_X);
 y = max(min(y, IK_MAX_Y), IK_MIN_Y);
 z = max(min(z, IK_MAX_Z), IK_MIN_Z);
  // Get the updated angles
 doArmIK(x, y, z, grip_angle_d);
}

#endif
```

ServoEx.h/ServoEx.cpp

This library for moving servos simultaneously. They were not modified from the RobotGeek code.

TestMotion.h

```
#ifndef TESTMOTION_H
#define TESTMOTION_H

#include "Kinematics.h"
#include "GlobalArm.h"


//=============================================================================
// Test Functions
//=============================================================================

/* Tests x = 0 */
void zero_x()
{
 for(double yaxis = 150.0; yaxis < 200.0; yaxis += 1){
   doArmIK(0, yaxis, 100.0, 0);
   SetServo(0);
   delay(10);
 }
 for(double yaxis = 200.0; yaxis > 150.0; yaxis -= 1){
   doArmIK(0, yaxis, 100.0, 0);
   SetServo(0);
   delay(10);
 }
}
/* Moves arm in a straight line */
void line()
{
   for(double xaxis = -100.0; xaxis < 100.0; xaxis += 0.5){
     doArmIK(xaxis, 200, 100, 0);
     SetServo(0);
     delay(10);
   }
   for(float xaxis = 100.0; xaxis > -100.0; xaxis -= 0.5){
     doArmIK(xaxis, 200, 100, 0);
     SetServo(0);
     delay(10);
   }
}
/* Moves arm in a circle */
void circle()
{
 #define RADIUS 20.0
 float zaxis, yaxis;
 for(float angle = 0.0; angle < 360.0; angle += 1.0){
     yaxis = RADIUS * sin(radians(angle)) + 150;
     zaxis = RADIUS * cos(radians(angle)) + 150;
     doArmIK(0, yaxis, zaxis, 0);
     SetServo(0);
     delay(5);
 }
}
```

```
#endif
```

LEAP MOTION CODE:

<u>LeapSDK.cpp</u>

```
#include <iostream>
#include <cstring>
#include <fstream>
#include <stdio.h>
#include <ctime>
#include "Leap.h"
using namespace std;


using namespace Leap;

class SampleListener : public Listener {
 public:
   virtual void onInit(const Controller&);
   virtual void onConnect(const Controller&);
   virtual void onDisconnect(const Controller&);
   virtual void onExit(const Controller&);
   virtual void onFrame(const Controller&);
   virtual void onFocusGained(const Controller&);
   virtual void onFocusLost(const Controller&);
   virtual void onDeviceChange(const Controller&);
   virtual void onServiceConnect(const Controller&);
   virtual void onServiceDisconnect(const Controller&);

 private:
};

const std::string fingerNames[] = {"Thumb", "Index", "Middle", "Ring", "Pinky"};
const std::string boneNames[] = {"Metacarpal", "Proximal", "Middle", "Distal"};
const std::string stateNames[] = {"STATE_INVALID", "STATE_START", "STATE_UPDATE",
"STATE_END"};

// File abstraction for serial communication with arduino
```

```
FILE *arduino;
// State of the arduino
bool state = 0;
// Time since hand disappears
std::clock_t start_time;

void SampleListener::onInit(const Controller& controller) {
 std::cout << "Initialized" << std::endl;

 // Open stream to Arduino serial port
 arduino = fopen("/dev/ttyACM0", "w");
}

void SampleListener::onConnect(const Controller& controller) {
 std::cout << "Connected" << std::endl;
 controller.enableGesture(Gesture::TYPE_CIRCLE);
 controller.enableGesture(Gesture::TYPE_KEY_TAP);
 controller.enableGesture(Gesture::TYPE_SCREEN_TAP);
 controller.enableGesture(Gesture::TYPE_SWIPE);
}

void SampleListener::onDisconnect(const Controller& controller) {
 // Note: not dispatched when running in a debugger.
 std::cout << "Disconnected" << std::endl;
}

void SampleListener::onExit(const Controller& controller) {
 std::cout << "Exited" << std::endl;
}

void SampleListener::onFrame(const Controller& controller) {
 // Get the most recent frame and report some basic information
 const Frame frame = controller.frame();

 // Get the first hand
 HandList hands = frame.hands();
 const Hand hand = *hands.begin();

 Leap::GestureList gestures = frame.gestures();
 for(Leap::GestureList::const_iterator gl = gestures.begin();
     gl != gestures.end(); gl++) {
   if ((*gl).type() == Leap::Gesture::TYPE_SCREEN_TAP) {
     // Turn on the arm
     state = 1;
   }
 }

 // Write palm's position to serial port
 if (state) {
   if (hands.count() > 0) {
     start_time = std::clock();
     Vector handPos = hand.palmPosition();
     float xOffset = -250;
     float yOffset = 80;
     float zOffset = 250;
     float x = handPos.y;
     float y = -handPos.z;
     float z = handPos.x;
     x = x + xOffset;
     y = y + yOffset;
     z = z + zOffset;
     std::cout << x << ',' << y << ',' << z << std::endl;
     fprintf(arduino, "%f,%f,%f\n", x, y, z);
   }
   else {
     double duration = (std::clock() - start_time) / (double) CLOCKS_PER_SEC;
     std::cout << duration << endl;
     if (duration > 0.05) {
       // Reset arm position
       fprintf(arduino, "0,185,230\n");
       state = 0;
     }
   }
 }
}

void SampleListener::onFocusGained(const Controller& controller) {
 std::cout << "Focus Gained" << std::endl;
```

```
}

void SampleListener::onFocusLost(const Controller& controller) {
 std::cout << "Focus Lost" << std::endl;
}

void SampleListener::onDeviceChange(const Controller& controller) {
 std::cout << "Device Changed" << std::endl;
 const DeviceList devices = controller.devices();

 for (int i = 0; i < devices.count(); ++i) {
   std::cout << "id: " << devices[i].toString() << std::endl;
   std::cout << "  isStreaming: " << (devices[i].isStreaming() ? "true" : "false") <<
std::endl;
 }
}

void SampleListener::onServiceConnect(const Controller& controller) {
 std::cout << "Service Connected" << std::endl;
}

void SampleListener::onServiceDisconnect(const Controller& controller) {
 std::cout << "Service Disconnected" << std::endl;
}

int main(int argc, char** argv) {
 // Create a sample listener and controller
 SampleListener listener;
 Controller controller;

 // Have the sample listener receive events from the controller
 controller.addListener(listener);

 if (argc > 1 && strcmp(argv[1], "--bg") == 0)
   controller.setPolicy(Leap::Controller::POLICY_BACKGROUND_FRAMES);

 // Keep this process running until Enter is pressed
 std::cout << "Press Enter to quit..." << std::endl;
 std::cin.get();
 // Close serial port
 fprintf(arduino, "0,185,185\n");
 fclose(arduino);

 // Remove the sample listener when done
 controller.removeListener(listener);

 return 0;
}
```

Makefile

```
OS := $(shell uname)
ARCH := $(shell uname -m)

ifeq ($(OS), Linux)
 ifeq ($(ARCH), x86_64)
   LEAP_LIBRARY := ./lib/x64/libLeap.so -Wl,-rpath,./lib/x64
 else
   LEAP_LIBRARY := ./lib/x86/libLeap.so -Wl,-rpath,./lib/x86
 endif
else
 # OS X
 LEAP_LIBRARY := ./lib/libLeap.dylib
endif

LeapSDK: LeapSDK.cpp
  $(CXX) -Wall -g -I ./include LeapSDK.cpp -o LeapSDK $(LEAP_LIBRARY)
ifeq ($(OS), Darwin)
  install_name_tool -change @loader_path/libLeap.dylib ./lib/libLeap.dylib LeapSDK
endif

clean:
  rm -rf LeapSDK LeapSDK.dSYM
```