ECE 478-578

PORTLAND STATE UNIVERSITY

Final report for Frankenstein Robot

Thuan Pham

Randon Stasney

Ram Bhattarai

December 10th 2016

# Table of Contents:

[**Abstract**](#_1fob9te)[**3**](#_1fob9te)

[**Introduction**](#_3znysh7)[**3**](#_3znysh7)

[**Leg Development**](#_2et92p0)[**4**](#_2et92p0)

[**Genetic Algorithm**](#_tyjcwt)[**4**](#_tyjcwt)

[**Upper body development**](#_3dy6vkm)[**5**](#_3dy6vkm)

[**Fuzzy Logic**](#_1t3h5sf)[**6**](#_1t3h5sf)

[**Kinect**](#_4d34og8)[**6**](#_4d34og8)

[**Flow chart**](#_sf0sec3kuzsb)[**9**](#_sf0sec3kuzsb)

[**Servo Controller Management**](#_qt3w8mo1o5zs)[**10**](#_qt3w8mo1o5zs)

[**Issues and concerns**](#_9qiw2jg9gmdz)[**11**](#_lnxbz9)

[**Future improvements and Development**](#_lnxbz9)[**11**](#_lnxbz9)

[**Conclusion**](#_vjh8tqsk6avc) **12**

[**Arduino Motor Control Code**](#_2cr056qefzxr) **12**

[**Fuzzy Logic Code**](#_b7y8v4trgogu)[**27**](#_b7y8v4trgogu)

[**Genetic Algorithm Code**](#_1q2wf62uz6b1)[**30**](#_1q2wf62uz6b1)

[**Gesture Recognition Code**](#_t21y00xzpz1z)[**39**](#_t21y00xzpz1z)

[**Speech Recognition Code**](#_iwkj82td7b2b)[**50**](#_iwkj82td7b2b)

[**Youtube Links**](#_3a54938)[**5**](#_3a54938)**8**

# **Abstract:**

The purpose of the project was to resurrect Frankenstein for the Portland State Robot Theater. The Theater has transformed in recent years to accept commands from the Kinect device via opencv. Through motion and speech recognition we will control the actions of our robot. Once the gesture information and voice commands from Kinect are passed via a serial communication to the Arduino controlling the robot, the Arduino program parses the information and implements the developed libraries to control the robot motions. Through the development of our created libraries and possible future work to further the range and types of motions, we feel that Frankenstein will find his place in the Theater. This report details the process of building, repairing, and integrating the robot as well as addressing issues and improvements for the robot.

# **Introduction:**

Frankenstein was originally Sonbi in the Portland State University robot theater. The documentation we were given was from a final report in 2006. In this report they stressed the importance of good documentation in order for future work to continue. Needless to say the robot has transformed greatly since then and having rebuilt this robot we share in this sentiment and the hope that this documentation will provide a follow up team with a strong starting point from which to build on. Through the course of years and a transformation into Frankenstein a lot was lost. This was our first glimpse of our Frankenstein.



Figure 1. Frankenstein day 1

We were asked to create a copy of his leg, connect Frankenstein to the Kinect link, and develop and improve his waving motion. After looking at what we had to start with and losing a team member before the second week, we were a bit dismayed. There was no leg to copy, all we found was a metal plate on the left side of the wooden box. There was no controller that we could discern besides the four motor controllers for the two degrees of freedom in the shoulders. There were no hands and no software for a waving motion. Most of the servos that we found had been cut and dismounted from the skeleton.

In our project description we were given our specific tasks. Ram took on opencv and the interface to Kinect. Thuan took over the general cleanup and the upper body. Randon went to work on the legs. Although we mainly focused on our areas, the robot homeworks and integration allowed us to work more as a team than individuals. This project would not have been possible without the tireless assistance of Melih the TA.

# **Leg Development:**

With no leg to match we were forced into recreating the legs as a pair. The original ideas came from just looking at the selection of Tetrix pieces in the lab. This very quickly forced a deeper understanding of how they are supposed to be connected. While some joints might be able to be supported by the servo horn, we knew the legs would weigh more and require a better support structure. A quick tour of the internet turned up the two relevant documents that we used in our leg creation. The first was “Getting Started with TETRIX” a .pdf from the Carnegie Mellon Robotics Academy. This illustrated the bracing required and the mounting techniques for what we were to use in our knee joint. This was further elucidated by the “TETRIX by Pitsco Creator’s Guide” by PJ Graham, Paul Uttley and Tony Allen. While having a lot of the Tetrix brackets, we did find a shortage of pivot bearings. In order to make an identical pair of legs we needed a pair of all the parts. After a furtive search of the lab we were able to find a second pivot bearing for the second knee. We were unable to find more to use in the hip joint and so we created our own bearings to support the other side of the bracket. The second problem that was quickly evident was that there were two types of servos in the lab and they had different horns and hardware. This required some not perfect connections or simply bending of the brackets to get our created legs.

Once our legs were built we tested each degree of freedom with a simple arduino sweep program. We found that a lot of the servos we had salvaged were somewhat damaged and therefore were not capable of giving us the required range of motion in our leg. We went through five knee servos and four hip servos until we had full range of motion. This change also forced horn replacements as we found ourselves working with the black gear servos. The next obstacle was simply the weight of our box and the inability for the legs to support the robot frame. This led us to digging out the pendulum frame on which we could mount Frankenstein. Our homework two was then implemented using our newly created legs to explore the fundamentals of Genetic Algorithm.

# **Genetic Algorithm:**

We chose to develop the leg motion for kicking a drum as our genetic algorithm. This facilitated a better understanding of our newly created legs while also satisfying the requirement that our robot “play” a musical instrument. Our robot is controlled through an Arduino Mega and we wanted to develop automatic feedback to control the development process. We started with the assumption that a “better” kick would produce a louder sound on the drum. We had created our legs with two degrees of freedom, one in the hip and one in the knee. Our robot was outfitted with shoes for a better contact point and avoiding the sharp metal edges of the feet from damaging the drum itself. Through the principles learned in class, we were able to have our robot teach himself to kick the drum better.

This was a process in understanding the limitations of our hardware. The original design for the GA took both joints in the process. The first parents were coded from our interpretation of natural leg and kicking motion. The knee goes back, the hip goes back slightly, and then the hip swings forward as we quickly bring the knee back to its starting position. This proved too much for our plastic geared servos and our first tests shredded a servo as illustrated below.



Figure 2. Gears of our knee servo

The linked videos display this first kick as well as the alterations we found necessary for the project. The recoil of the impact with the drum itself proved that we needed a gentler approach. The solution was to just apply the genetic algorithm with the hip joint. In just a few generations we had achieved the maximum value on our sound sensor and called it a success. If we want further development, we could move the sound sensor away from the drum and make the algorithm work harder for success. We however feared shredding more servos even without the knee swing if we actually were successful in making it that much stronger.

# **Upper body development:**

The initial portion of the upper body development consisted of testing and cleaning. We cleaned off the no longer connected servos on the head which left us with three degrees of freedom in the head. One servo controlled the eyes and directed them right or left. One servo controlled the eyelids, while the last opened the mouth by controlling the jaw. The shoulder degrees of freedom were tested and in the process one of the left shoulder controllers shorted out and fried. This was corrected by installing a dual full bridge driver. Later during integration one of the right shoulder motor controllers also stopped functioning. We were able to use the second channel for this shoulder. The shoulders protrude from the box chest cavity in such a manner that force was applied presumably while lying down that required redoing the shoulder gear box.

In our search for the missing leg we did come across a usable hand that we were able to attach to the forearm. This gave us a degree of freedom in the wrist and one in the gloved motion of the hand closing. Still missing a hand we found one that we could 3D print with Melihs assistance. This was more in the shape of a claw and then was also attached to the arm with another degree of freedom as a wrist motion. The upper body is where we focused our efforts for the fuzzy logic homework.

# **Fuzzy Logic:**

Fuzzy logic is a powerful method to integrate smooth and gradual motion, compared to having on or off motion. It's a must in the robotics world because robots cannot be on and off only, it needs to have in between motion between the two extremes to create realistic and smooth motion. For Homework 1, we implemented a fuzzy logic system to smooth Frankenstein’s eye movements. We used the ultrasonic sensors on his shoulders as a tracking mechanism. This was implemented through the use of a fuzzy logic library found in python library called scikit-fuzzy. We based our design on the tipping tutorial. This library allowed us to program our inputs and outputs as fuzzy logic antecedents and consequences. Once we had those variables set up we created the membership functions and defined their shape. We then created the logic by defining the rules that would be implemented. By the python code on our laptop with our microcontroller, the arduino, we were able to get Frankenstein's eyes to track in a smoother manner.

This functionality was not implemented in our final robot mostly because the actual processing in python was happening external to the Arduino on a laptop that was serially connected by the Pololu Wixel. The Wixel presented its own challenges and appeared to work better after we used the shield with the Arduino.

# **Kinect:**

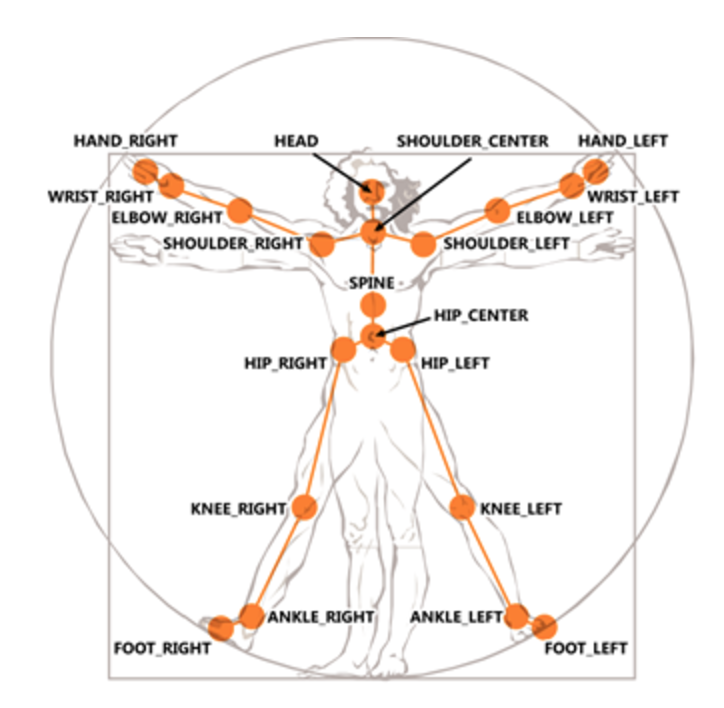
Kinect is one of the powerful cameras produced by Microsoft. Kinect has an inbuilt IR emitter, color sensor, IR depth sensor, Microphone, and Tilt motor. It is the most popular 3D camera used in robotics. Kinect can be used in various applications like object recognition, object detection, environment mapping, distance detection, voice recognition, etc. Kinect is very powerful if it is used for indoor applications, because outside you may not get the result that you are expecting. It can be really challenging to write a program to control robots using Kinect for outdoor applications because you will get so many inputs from surrounding which might hamper the performance of kinect. Actually, Kinect is just a sensor which has a camera in it and works best in an environment where there is no outside involvement. One of the best advantages of Kinect is that you can control it by writing a program. You can also control or tilt the camera according to your needs from the software. One drawback of Kinect is that it is very difficult to install the required libraries in other operating systems. Kinect was first designed to be used for the Microsoft applications. But the amazing features of the Kinect made people want to create libraries for MAC and Linux environments. Today, you can write a program in MAC and Linux to control the Kinect, but getting the libraries to install it is a really challenging task.



**Figure 3: Kinect Camera**

**Gesture Recognition:**

Kinect camera detects the movement using the 3D sensor. For gesture recognition, Microsoft has written a skeleton tracking program where you can get 20 joints from the body. You can use those joints’ information to track the movement of each joint and process that information to build motion for robots. In this project, we used couple of hand gestures like right hand up, right hand down, both hands up and both hands down to form a motion for the robot. If you need to know the angle of each joint’s live movement, you can create a 3D vector where you track the joints in X-axis, Y-axis and Z-axis (the axis that is pointing towards the camera).



**Figure 4. Skeleton Tracking(Borrowed from Microsoft Kinect Website)**

**Speech Recognition:**

Speech recognition is a powerful tool in Kinect. You can control the robot with your voice commands. The system detects the speech and processes that information and robot responds to your speech. Microsoft has created the library called Microsoft.Speech which you need to install onto your computer to use speech recognition. For other operating system platforms, you need to either write a library for it or you can download the library, but it might be a challenging job to find the required library. I would say using the library that is already available for Windows is a good start for any project. Once you have enough information about speech recognition software, you can tweak it and maybe you can write a program in different OS platforms. For this project, we have used 14 commands for the robot. Commands are:

1. Right Arm
2. Right Leg
3. Left Leg
4. Left Arm
5. Head
6. Funny
7. Kick
8. Wave
9. Walk
10. Salute
11. Badboy
12. Sprint
13. Kneel
14. Stroll

You can watch all these commands live from the video that we posted in our Youtube section. When you are giving voice commands, you have to be loud and clear. Kinect does not process the information that is not loud and clear. It is best to have only one person talking while the audio is processing. It is usually best for indoor application. You can also use speech recognition in your house to control the house environment. I was watching a video in Youtube where a guy controls his house lights, fan, tv, etc by just giving voice commands.

**Kinect gesture Lagging:**

Kinect processes the information in fraction of seconds, so processing that information lagged the communication between Kinect and Arduino. Not only that, it also delayed the skeleton movement. When we were trying to process the information from Kinect, it was processing so fast that information was passing incorrectly. We had to add a counter of 300. So, when the counter reached 300, the information was passed to Arduino.

# Flow chart:

The following flowchart shows how the Robot receives the Gesture and Voice commands from Kinect and passes that information to visual studio where the program generates the skeleton out of it and starts tracking 20 joints from the body. From the generated skeleton, the program tracks gestures like right hand up, left hand up,both hands up, and both hands down. This information is then passed to Arduino via a serial communication as a string where the program for Arduino parses the information and creates various motion for robot which will be discussed in later sections.

|  |
| --- |
|  |

**Figure 5. Kinect-Robot program flow**

# **Servo Controller Management:**

The final library of our motions are controlled from an Arduino Mega through the Pololu Maestro. The Maestro has 18 channels that we tested directly with each servo to give us our minimum and maximum frequency range. Knowing this range we were able to map these values to a standardized 0 - 180 that resembled regular servo control. The Arduino then simply writes serially to whichever channel you need to control. You can preset the speed and acceleration or can change them within the setup commands on the Arduino itself.

The servos were grouped into 5 main body parts; the head, left arm, left leg, right leg and right arm. The controller was broken down into a ten stage state machine that allowed a single second of action. With this method we can predefine a series of servo positions while also allowing independent selection of which library code we want to run for each of the five body areas. This allows all combinations to then be possible for every defined motion.

The Kinect then only has to send a serial vector that is decoded into each of the predefined categories. This in turn is parsed into the controlling the state machine for each of the body parts. We also added the functionality to define an override function that is not limited to the 1 second state machine and allows the user to define a function including all the delays present. The coded example of this is our “stroll” command.

**Vector element value**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Right arm | neutral | active |  |  |  |  |  |
| Right leg | neutral | walk | kick | sprint | kneel |  |  |
| Left arm | neutral | active |  |  |  |  |  |
| Left leg | neutral | walk | knee | sprint | kneel |  |  |
| Head | neutral | all | mouth | look left | look right | blink lids | lids shut |
| Special | Off | “stroll” |  |  |  |  |  |

**figure 6. table of commands in movement library**

# **Issues and concerns:**

The main issue is the shoulders. We were successful in controlling them with the L298N dual full bridge however the voltage required appears to make them quite jumpy and gravity has a very strong effect in their downward motion. They can also flip over if the angles are written poorly. We did redo the gearbox but its stability seems to be failing. The two shoulder joints controlled by the existing motor controllers also have experienced slipping of the potentiometer values making calibration difficult over any period of time. Due to this, most of our library code focused on the other regions of the body.

# Future improvements and Development:

1. Fix shoulders allowing better arm development.
2. Develop more integrated arm motions.
3. Robot does not have neck so add neck to make head moving.
4. Add more gesture for robot by tracking the movements from Kinect.
5. Add more voice commands.
6. Reconnect the wixel to allow the Kinect to transmit the data wirelessly.
7. Extend the state machine to 20 states or 2 seconds to allow greater flexibility.

# Conclusion:

From some rough beginnings we have produced a fully functioning robot. We have grown as a team and were able to assist each other when things didn’t quite work. We have developed a lot of software for the different aspects controlling our robot and the interconnections used in this project that will help future students to build from our progress and not repeat our mistakes. We have a functional platform in which one can easily add new body part gestures for motor control. Linked to this we have the framework that will allow those and any other combination of these motor control libraries to be represented as a voice or gesture command from the Kinect device. We have been successful in implementing Frankenstein as a lasting member of the Portland State Robot Theatre.

# Arduino Motor Control Code:

|  |
| --- |
| //Control Library for Frankenstein// fall 2106 Perkowski Robotics 1// Thuan Pham, Randon Stasney, Ram Bhattarai #include <PololuMaestro.h>  #define maestroSerial Serial2  // serial 2 is tx pin 16  MiniMaestro maestro(Serial2);  // connect motor controller pins to Arduino digital pins  int enA = 5;  int enB = 6;  int in1 = 10;  int in2 = 11;  int in3 = 12;  int in4 = 13;  int sen2 = A1;  int sen = A0; //analog pin2 for potentiometer  // Hold values from kinect  int rarm = 0;  int rleg = 0;  int larm = 0;  int lleg = 0;  int head = 0;  int Unique = 0; // enables non time slice behaviors  // Time slice state machine tracking  int State = 0;  void setup() {  // serial connection to Kinect  Serial.begin(9600);  //dc motor controller  pinMode(enA, OUTPUT);  pinMode(in1, OUTPUT);  pinMode(in2, OUTPUT);  pinMode(enB, OUTPUT);  pinMode(in3, OUTPUT);  pinMode(in4, OUTPUT);  // Set the serial port's baud rate.  Serial2.begin(9600);  maestro.setSpeed(8, 100);  maestro.setSpeed(0, 500);  maestro.setSpeed(1, 500);  maestro.setAcceleration(1, 10000);  }  void loop() {  // enables non time slice actions overrides body part inputs  if (Unique == 1){  walk();    }  // separates body into 5 areas each controlled by a 1 second state machine  // allows any combination of preprogrammed behavior  else {  RightLeg (rleg, State);  Head (head, State);  RightArm (rarm, State);  LeftArm (larm, State);  LeftLeg (lleg, State);  delay(100);  // state machine works on 1/10th second slice  if (State == 9) {  State = 0;  }  else {  State = State + 1;  }  }  }  ////////// map individual parts to maestro frequency so all uniformly controlled by 0-180 inputs  // first number is maestro connection port on the board  void jaw(int degree) {  maestro.setTarget(12, map(degree, 0, 180, 4800, 7000));  }  void brows(int degree) {  maestro.setTarget(11, map(degree, 0, 180, 2000, 4300));  }  void eyes(int degree) {  maestro.setTarget(10, map(degree, 0, 180, 6000, 8000));  }  void left\_claw(int degree) {  maestro.setTarget(9, map(degree, 0, 180, 2400, 7500));  }  void left\_shoulder(int degree) {  maestro.setTarget(8, map(degree, 0, 180, 4000, 8000));  }  void left\_hand(int degree) {  maestro.setTarget(7, map(degree, 0, 180, 3200, 9000));  }  void left\_knee(int degree) {  maestro.setTarget(6, map(degree, 0, 180, 1600, 8000));  }  void left\_hip(int degree) {  maestro.setTarget(5, map(degree, 0, 180, 4000, 8000));  }  void right\_knee(int degree) {  maestro.setTarget(4, map(degree, 0, 180, 2500, 8000));  }  void right\_hip(int degree) {  maestro.setTarget(3, map(degree, 0, 180, 2500, 8000));  }  void right\_hand(int degree) {  maestro.setTarget(2, map(degree, 0, 180, 1600, 8800));  }  void right\_claw(int degree) {  maestro.setTarget(1, map(degree, 0, 180, 2400, 6000));  }  void right\_arm(int degree) {  maestro.setTarget(0, map(degree, 0, 180, 2800, 6000));  }  // logic for the 2 dual full bridge drivers  void turn\_right(int speeds) {  digitalWrite(in1, HIGH);  digitalWrite(in2, LOW);  analogWrite(enA, speeds);  }  void turn\_right2(int speeds) {  digitalWrite(in3, HIGH);  digitalWrite(in4, LOW);  analogWrite(enB, speeds);  }  void turn\_left(int speeds) {  digitalWrite(in1, LOW);  digitalWrite(in2, HIGH);  analogWrite(enA, speeds);  }  void turn\_left2(int speeds) {  digitalWrite(in3, LOW);  digitalWrite(in4, HIGH);  analogWrite(enB, speeds);  }  void turn\_off() {  digitalWrite(in1, LOW);  digitalWrite(in2, LOW);  }  void turn\_off2() {  digitalWrite(in3, LOW);  digitalWrite(in4, LOW);  }  int read\_position() {  int pos = analogRead(sen);  int degree = map(pos, 360, 710, 0, 180);  return degree;  }  int read\_position2() {  int pos = analogRead(sen2);  int degree = map(pos, 27, 850, 0, 180);  return degree;  }  void left\_arm(int setA) {  while (setA != read\_position()) {  int speeds = 2000;  if ( abs(read\_position() - setA) < 5) {  speeds = 75;  }  if (setA < read\_position()) {  turn\_left(speeds);  }  else if (setA > read\_position()) {  turn\_right(speeds);  }  }  turn\_off();  }  void right\_shoulder(int setA) {  while (setA != read\_position2()) {  int speeds = 200;  if ( abs(read\_position2() - setA) < 10) {  speeds = 85;  }  if (setA < read\_position2()) {  turn\_left2(speeds);  }  if (setA >= read\_position2()) {  turn\_right2(speeds);  }  }  turn\_off2();  }  // "unique" non time sliced example  void walk() {  left\_hip(60);  right\_hip(130);  left\_knee(130);  right\_knee(180);  delay(400);  left\_hip(180);  right\_hip(30);  left\_knee(180);  right\_knee(125);  delay(400);  }  // another unique example with beckoning motion  void come\_here() {  left\_hand(180);  left\_shoulder(180);  left\_claw(0);  delay(100);  left\_claw(180);  delay(100);  }  void serialEvent() {  // get Kinect data data and parse into ints to control body parts  while (Serial.available()) {  rarm = Serial.parseInt();  rleg = Serial.parseInt();  larm = Serial.parseInt();  lleg = Serial.parseInt();  head = Serial.parseInt();  Unique = Serial.parseInt();  if (Serial.read() == '\n') {  Serial.print(rarm);  Serial.print(rleg);  Serial.print(larm);  Serial.print(lleg);  Serial.print(head);  Serial.print(Unique);  }  }  }  ///////////////////////////////New Functions  // left arm shoulder claw and hand display  void LA\_Up(int LA) {  switch (LA) {  case 0:  left\_shoulder(20);  left\_claw(30);  left\_hand(100);  break;  case 1:  left\_shoulder(40);  left\_claw(60);  left\_hand(120);  break;  case 2:  left\_shoulder(60);  left\_claw(90);  left\_hand(140);  break;  case 3:  left\_shoulder(90);  left\_claw(120);  left\_hand(1500);  break;  case 4:  left\_shoulder(120);  left\_claw(1500);  left\_hand(160);  break;  case 5:  left\_shoulder(150);  left\_claw(1500);  left\_hand(1700);  break;  case 6:  left\_shoulder(150);  left\_claw(1200);  left\_hand(150);  break;  case 7:  left\_shoulder(110);  left\_claw(80);  left\_hand(130);  break;  case 8:  left\_shoulder(70);  left\_claw(40);  left\_hand(110);  break;  case 9:  left\_shoulder(40);  left\_claw(20);  left\_hand(100);  break;  }  }  // Neutral position  void LA\_Nut() {  left\_shoulder(0);  left\_claw(0);  left\_hand(90);  }  // right arm claw and hand display  void RA\_Up(int RA) {  switch (RA) {  case 0:  right\_arm(20);  right\_claw(60);  right\_hand(90);  break;  case 1:  right\_arm(60);  right\_claw(120);  right\_hand(120);  break;  case 2:  right\_arm(100);  right\_claw(180);  right\_hand(150);  break;  case 3:  right\_arm(140);  right\_claw(180);  right\_hand(180);  break;  case 4:  right\_arm(170);  right\_claw(180);  right\_hand(90);  break;  case 5:  right\_arm(170);  right\_claw(180);  right\_hand(90);  break;  case 6:  right\_arm(140);  right\_claw(180);  right\_hand(20);  break;  case 7:  right\_arm(110);  right\_claw(120);  right\_hand(20);  break;  case 8:  right\_arm(70);  right\_claw(60);  right\_hand(20);  break;  case 9:  right\_arm(40);  right\_claw(0);  right\_hand(90);  break;  }  }  // Neutral position  void RA\_Nut() {  right\_arm(0);  right\_claw(180);  right\_hand(90);  }  // walking offset from left leg  void R\_Walk (int RL) {  switch (RL) {  case 0:  right\_hip(110);  right\_knee(180);  break;  case 1:  right\_hip(110);  right\_knee(170);  break;  case 2:  right\_hip(90);  right\_knee(160);  break;  case 3:  right\_hip(70);  right\_knee(150);  break;  case 4:  right\_hip(50);  right\_knee(150);  break;  case 5:  right\_hip(30);  right\_knee(130);  break;  case 6:  right\_hip(50);  right\_knee(130);  break;  case 7:  right\_hip(70);  right\_knee(130);  break;  case 8:  right\_hip(90);  right\_knee(130);  break;  case 9:  right\_hip(110);  right\_knee(130);  break;  }  }  // simplified kicking the drum motion only using hip  void R\_Kick (int RL) {  switch (RL) {  case 0:  right\_hip(120);  break;  case 1:  right\_hip(110);  break;  case 2:  right\_hip(100);  break;  case 3:  right\_hip(90);  break;  case 4:  right\_hip(80);  break;  case 5:  right\_hip(60);  break;  case 6:  right\_hip(50);  break;  case 7:  right\_hip(40);  break;  case 8:  right\_hip(30);  break;  case 9:  right\_hip(20);  break;  }  }  // Neutral position  void RL\_Nut() {  right\_hip(100);  right\_knee(180);  }  // leg back and knee extended opposite left sprint  void R\_Sprint() {  right\_hip(180);  right\_knee(100);  }  // kneeling motion on right leg  void R\_Kneel() {  right\_hip(80);  right\_knee(20);  }  // time slice to rotate knee  void L\_Knee (int LL) {  switch (LL) {  case 0:  left\_hip(120);  left\_knee(170);  break;  case 1:  left\_hip(120);  left\_knee(150);  break;  case 2:  left\_hip(120);  left\_knee(150);  break;  case 3:  left\_hip(120);  left\_knee(130);  break;  case 4:  left\_hip(120);  left\_knee(110);  break;  case 5:  left\_hip(130);  left\_knee(80);  break;  case 6:  left\_hip(130);  left\_knee(50);  break;  case 7:  left\_hip(140);  left\_knee(30);  break;  case 8:  left\_hip(130);  left\_knee(100);  break;  case 9:  left\_hip(130);  left\_knee(160);  break;  }  }  // rotate hip and knee offset to other leg  void L\_Walk (int LL) {  switch (LL) {  case 0:  left\_hip(60);  left\_knee(130);  break;  case 1:  left\_hip(75);  left\_knee(160);  break;  case 2:  left\_hip(90);  left\_knee(135);  break;  case 3:  left\_hip(105);  left\_knee(140);  break;  case 4:  left\_hip(120);  left\_knee(145);  break;  case 5:  left\_hip(135);  left\_knee(160);  break;  case 6:  left\_hip(150);  left\_knee(165);  break;  case 7:  left\_hip(165);  left\_knee(170);  break;  case 8:  left\_hip(170);  left\_knee(175);  break;  case 9:  left\_hip(180);  left\_knee(180);  break;  }  }  // Neutral position  void LL\_Nut() {  left\_hip(100);  left\_knee(180);  }  // left leg forward and extended  void L\_Sprint() {  left\_hip(0);  left\_knee(160);  }  // left leg resting on knees  void L\_Kneel() {  left\_hip(130);  left\_knee(60);  }  // mostly move jaw as if to talk  void Talk (int H) {  switch (H) {  case 0:  eyes(90);  brows(0);  jaw(20);  break;  case 1:  eyes(90);  brows(0);  jaw(60);  break;  case 2:  eyes(80);  brows(0);  jaw(80);  break;  case 3:  eyes(80);  brows(10);  jaw(120);  break;  case 4:  eyes(90);  brows(10);  jaw(160);  break;  case 5:  eyes(90);  brows(10);  jaw(160);  break;  case 6:  eyes(100);  brows(10);  jaw(120);  break;  case 7:  eyes(100);  brows(0);  jaw(80);  break;  case 8:  eyes(90);  brows(0);  jaw(50);  break;  case 9:  eyes(90);  brows(25);  jaw(20);  break;  }  }  // animate brows eyes and talk  void Look (int H) {  switch (H) {  case 0:  eyes(90);  brows(0);  jaw(20);  break;  case 1:  eyes(110);  brows(20);  jaw(60);  break;  case 2:  eyes(120);  brows(40);  jaw(80);  break;  case 3:  eyes(130);  brows(70);  jaw(120);  break;  case 4:  eyes(140);  brows(100);  jaw(160);  break;  case 5:  eyes(150);  brows(130);  jaw(160);  break;  case 6:  eyes(160);  brows(100);  jaw(120);  break;  case 7:  eyes(140);  brows(90);  jaw(80);  break;  case 8:  eyes(120);  brows(50);  jaw(500);  break;  case 9:  eyes(100);  brows(25);  jaw(20);  break;  }  }  // flash eyelids  void Bashful (int H) {  eyes(90);  jaw(0);  switch (H) {  case 0:  brows(40);  break;  case 1:  brows(80);  break;  case 2:  brows(120);  break;  case 3:  brows(160);  break;  case 4:  brows(180);  break;  case 5:  brows(160);  break;  case 6:  brows(120);  break;  case 7:  brows(80);  break;  case 8:  brows(40);  break;  case 9:  brows(0);  break;  }  }  // look to the left  void LookL () {  brows(0);  jaw(0);  eyes(180);  }  // look to the right  void LookR () {  brows(0);  jaw(0);  eyes(0);  }  // eyelids down  void LidDown () {  brows(180);  jaw(0);  eyes(90);  }  // Neutral position  void H\_Nut() {  eyes(90);  brows(0);  jaw(0);  }  ///////////////////////CODE FOR KINECT BITS  // head programmed functions  void Head (int str, int state) {  switch (str) {  case 0:  H\_Nut();  break;  case 1:  Look(state);  break;  case 2:  Talk(state);  break;  case 3:  LookL();  break;  case 4:  LookR();  break;  case 5:  Bashful(state);  break;  case 6:  LidDown();  break;  }  }  // left arm functions  void LeftArm (int str, int state) {  switch (str) {  case 0:  LA\_Nut();  break;  case 1:  LA\_Up(state);  break;  }  }  // right arm functions  void RightArm (int str, int state) {  switch (str) {  case 0:  RA\_Nut();  break;  case 1:  RA\_Up(state);  break;  }  }  // left leg functions  void LeftLeg (int str, int state) {  switch (str) {  case 0:  LL\_Nut();  break;  case 1:  L\_Walk(state);  break;  case 2:  L\_Knee(state);  break;  case 3:  L\_Sprint();  break;  case 4:  L\_Kneel();  break;  }  }  // right leg functions  void RightLeg (int str, int state) {  switch (str) {  case 0:  RL\_Nut();  break;  case 1:  R\_Walk(state);  break;  case 2:  R\_Kick(state);  break;  case 3:  R\_Sprint();  break;  case 4:  R\_Kneel();  break;  }  } |

# 

# **Fuzzy Logic Code:**

|  |
| --- |
| # Fuzzy Logic for Frankenstein in Python  # Homework 1 fall 2106 Perkowski Robotics 1  # Thuan Pham, Randon Stasney, Ram Bhattarai  # Based on the tipping tutorial for scikit-fuzzy  import numpy as np  import skfuzzy as fuzz  import matplotlib.pyplot as plt  from skfuzzy import control as ctrl  import time  import serial  #Consequents  eye\_movement = ctrl.Consequent(np.arange(90,175,1), 'eye\_movement')  #Antecedents  sensorR = ctrl.Antecedent(np.arange(0,61,1), 'sensorR')  sensorL = ctrl.Antecedent(np.arange(0,61,1), 'sensorL')  #membership for eye balls  eye\_movement['HL'] = fuzz.trapmf(eye\_movement.universe, [90,90,104,118])  eye\_movement['ML'] = fuzz.trimf(eye\_movement.universe, [104,118,132])  eye\_movement['M'] = fuzz.trimf(eye\_movement.universe, [118,132,146])  eye\_movement['MR'] = fuzz.trimf(eye\_movement.universe, [132,146,160])  eye\_movement['HR'] = fuzz.trapmf(eye\_movement.universe, [146,160,175,175])  #membership for right ultrasonic sensor  sensorR['vfar'] = fuzz.trapmf(sensorR.universe,[30,40,60,60])  sensorR['far'] = fuzz.trimf(sensorR.universe,[20,30,40])  sensorR['medium'] = fuzz.trimf(sensorR.universe,[15,20,30])  sensorR['close'] = fuzz.trimf(sensorR.universe,[10,15,20])  sensorR['vclose'] = fuzz.trapmf(sensorR.universe,[0,0,10,15])  #membership for left ultrasonic sensor  sensorL['vfar'] = fuzz.trapmf(sensorL.universe,[30,40,60,60])  sensorL['far'] = fuzz.trimf(sensorL.universe,[20,30,40])  sensorL['medium'] = fuzz.trimf(sensorL.universe,[15,20,30])  sensorL['close'] = fuzz.trimf(sensorL.universe,[10,15,20])  sensorL['vclose'] = fuzz.trapmf(sensorL.universe,[0,0,10,15])  #Rules for output  rule\_m = ctrl.Rule( (sensorR['vfar'] & sensorL['vfar']) | \  (sensorR['far'] & sensorL['far']) | \  (sensorR['medium'] & sensorL['medium']) | \  (sensorR['close'] & sensorL['close']) | \  (sensorR['vclose'] & sensorL['vclose']), \  eye\_movement['M'])  rule\_hl = ctrl.Rule( (sensorR['medium'] & sensorL['vclose']) | \  (sensorR['far'] & sensorL['vclose']) | \  (sensorR['vfar'] & sensorL['vclose']) | \  (sensorR['far'] & sensorL['close']) | \  (sensorR['vfar'] & sensorL['close']) | \  (sensorR['vfar'] & sensorL['medium']), \  eye\_movement['HL'])  rule\_ml = ctrl.Rule( (sensorR['close'] & sensorL['vclose']) | \  (sensorR['medium'] & sensorL['close']) | \  (sensorR['far'] & sensorL['medium']) | \  (sensorR['vfar'] & sensorL['far']), \  eye\_movement['ML'])  rule\_mr = ctrl.Rule( (sensorR['vclose'] & sensorL['close']) | \  (sensorR['close'] & sensorL['medium']) | \  (sensorR['medium'] & sensorL['far']) | \  (sensorR['far'] & sensorL['vfar']), \  eye\_movement['MR'])  rule\_hr = ctrl.Rule( (sensorR['vclose'] & sensorL['medium']) | \  (sensorR['vclose'] & sensorL['far']) | \  (sensorR['close'] & sensorL['far']) | \  (sensorR['vclose'] & sensorL['vfar']) | \  (sensorR['close'] & sensorL['vfar']) | \  (sensorR['medium'] & sensorL['vfar']), \  eye\_movement['HR'])    #combining rules  eye\_fuzzy = ctrl.ControlSystem([rule\_hl, rule\_hr, rule\_mr, rule\_ml, rule\_m])  #set up testing simulation for fuzzy logic  test = ctrl.ControlSystemSimulation(eye\_fuzzy)  # for MAC  #ser = serial.Serial('/dev/cu.usbmodem1411')  #print (ser.name)  #ser.baudrate = 9600  # for Windows  ser = serial.Serial('com8', 9600)  print (ser.name)  ser.baudrate = 9600  time.sleep (2)  ser.flushInput()  ser.flushOutput()  while True:  #read serial value of sensor  #time.sleep(.5)  e = int(ser.readline())  print(e)  #time.sleep(.5)  e2 = int(ser.readline())  print(e2)  #input values for sensors  test.input['sensorR'] = e  test.input['sensorL'] = e2  #run test  test.compute()  #Print out defuzzfied values  print (test.output['eye\_movement'])  #get defuzzfied value  cat = str(round(test.output['eye\_movement']))    #write value to motor  print("Writing to motor servo")  ser.write(b'eye:' + bytes(cat.encode()) + '\n'.encode()) |

# **Genetic Algorithm Code**:

|  |
| --- |
| /\*Based on GA concept of Daniel Diaz  \* Randon Stasney, Thuan Pham, Ram Bhattarai  \* ECE 578 Robotics  \* Perkowski Fall 2016  \* Develop GA for leg movement to kick a drum  \*/  #include <Servo.h>  #include <time.h>  #include <stdlib.h>  // constants  #define lowerbound 15 // servo range min  #define upperbound 160 // servo range max  #define wait 5000 // delay  int sensorPin = A5; // select the input pin for the potentiometer for sound sensor  int sensorValue = 0; // variable to store the value coming from the sensor  int sensorValueHold = 0; // temp max value for 2 second test  int sensorP1 = 0; // hold sensor values for the 4 tests  int sensorP2 = 0;  int sensorC1 = 0;  int sensorC2 = 0;  //declare array for parent chromosome and fill with starting values of knee back then forward as we raise hip  byte P1[] = {0xF9, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF, 0xFF, 0x77, 0x77, 0x77, 0x77, 0x77, 0x77, 0x66, 0x65, 0x64, 0x63, 0x50};  byte P2[] = {0xF0, 0xF1, 0xF1, 0xF2, 0xF2, 0xF2, 0xF2, 0xF3, 0xF3, 0x73, 0x73, 0x74, 0x74, 0x75, 0x75, 0x66, 0x66, 0x67, 0x67, 0x57};  // child array  byte C1[20];  byte C2[20];  int i = 0; // random number for mutation and crossover  byte kneePos = 0; // position of servo for writing  byte hipPos = 0;  byte MaskedKneePos = 0; // result after mask of strentgh of 1/10th second  byte MaskedHipPos = 0;  byte DirKnee = 0; // result after mask 0 backward 1 forward  byte DirHip = 0;  byte KneeMaskDir = 0x80; // mask chromosome byte to determine direction  byte HipMaskDir = 0x08;  byte KneeMaskPos = 0x70; // mask chromosome byte to get value from 0-7 for time segment  byte HipMaskPos = 0x07;  //setup our servos  Servo hip, knee;  // print to serial so we can actually see values once we're done  void displayVars(byte P1[], byte P2[], byte C1[], byte C2[])  {  for (int a = 0; a < 20; a = a + 1){  Serial.print("P1 = ");  Serial.println(P1[a], HEX);  Serial.print("P2 = ");  Serial.println(P2[a], HEX);  Serial.print("C1 = ");  Serial.println(C1[a], HEX);  Serial.print("C2 = ");  Serial.println(C2[a], HEX);  }  }  // do crossover on parents to create children based on random value  void crossover(byte P1[], byte P2[], byte C1[], byte C2[], int num)  {  int cross = 0;  cross = (num % 18) + 1;  Serial.println("crossover");  Serial.println(cross, HEX);  for (int a = 0; a < 20; a = a + 1){  Serial.println(cross, HEX);  Serial.println(a, HEX);  if (a < cross){  // Serial.println("bottom cross");  C1[a] = P1[a];  C2[a] = P2[a];  }  else {  // Serial.println("top cross");  C1[a] = P2[a];  C2[a] = P1[a];  }  }  }  //mutate a chromosome to random value  void mutation(byte P1[], byte P2[], byte C1[], byte C2[], int num)  {  // Serial.println("mutation");  byte M1;  byte M2;  int Mutation = 0;  M1 = num % 256;  M2 = (num \* 2) % 256;  Mutation = (num % 19);  Serial.println(M1, HEX);  Serial.println(M2, HEX);  Serial.println(Mutation, HEX);  for (int a = 0; a < 20; a = a + 1){  Serial.println(a, HEX);  if (a == Mutation){  C1[a] = M1;  C2[a] = M2;  }  else {  C1[a] = P1[a];  C2[a] = P2[a];  }  }  }  void setup() {  Serial.begin(9600); //print to screen  srand(time(NULL));    hip.attach(5);  knee.attach(6);  }  void loop() {  while (1){  //generate random crossover points  i = rand();  //10% chance of mutation  if(i % 10 == 0){    mutation(P1, P2, C1, C2, i);  Serial.println(i, HEX);  }  else {  crossover(P1, P2, C1, C2, i);  Serial.println(i, HEX);  }  displayVars(P1, P2, C1, C2);  //P1 Test  delay(500);  kneePos = 165;  hipPos = 90;  for (hipPos > 0; hipPos <= 90; hipPos += 1) { // reset to 90 degrees  hip.write(hipPos); // tell servo to go to position in variable 'pos'  delay(15); // waits 15ms for the servo to reach the position  }  knee.write((165));  delay(100);  sensorValueHold = 0;  for (int a = 0; a < 20; a = a + 1){  DirKnee = P1[a] & KneeMaskDir;  DirHip = P1[a] & HipMaskDir;  MaskedKneePos = ((P1[a] & KneeMaskPos) >> 4);  MaskedHipPos = (P1[a] & HipMaskPos);  if (DirKnee = 0){  kneePos = kneePos + MaskedKneePos;  if (kneePos > upperbound){  kneePos = upperbound;  }  }  else {  kneePos = kneePos - MaskedKneePos;  if (kneePos < lowerbound){  kneePos = lowerbound;  }  }  if (DirHip = 0){  hipPos = hipPos + MaskedHipPos;  if (hipPos > upperbound){  hipPos = upperbound;  }  }  else {  hipPos = hipPos - MaskedHipPos;  if (hipPos < lowerbound){  hipPos = lowerbound;  }  }  hip.write(hipPos);  // knee.write((180-kneePos));  sensorValue = analogRead(sensorPin);  if (sensorValue > sensorValueHold){  sensorValueHold = sensorValue;  }  delay(100);  }  sensorP1 = sensorValueHold;  Serial.println("p1 values");  Serial.println(sensorP1, DEC);  Serial.println(hipPos, DEC);  Serial.println(kneePos, DEC);  //P2 Test  delay(500);  kneePos = 165;  hipPos = 90;  for (hipPos > 0; hipPos <= 90; hipPos += 1) { // reset to 90 degrees  hip.write(hipPos); // tell servo to go to position in variable 'pos'  delay(15); // waits 15ms for the servo to reach the position  }  knee.write((165));  delay(100);  sensorValueHold = 0;  for (int a = 0; a < 20; a = a + 1){  DirKnee = P2[a] & KneeMaskDir;  DirHip = P2[a] & HipMaskDir;  MaskedKneePos = ((P2[a] & KneeMaskPos) >> 4);  MaskedHipPos = (P2[a] & HipMaskPos);  if (DirKnee = 0){  kneePos = kneePos + MaskedKneePos;  if (kneePos > upperbound){  kneePos = upperbound;  }  }  else {  kneePos = kneePos - MaskedKneePos;  if (kneePos < lowerbound){  kneePos = lowerbound;  }  }  if (DirHip = 0){  hipPos = hipPos + MaskedHipPos;  if (hipPos > upperbound){  hipPos = upperbound;  }  }  else {  hipPos = hipPos - MaskedHipPos;  if (hipPos < lowerbound){  hipPos = lowerbound;  }  }  hip.write(hipPos);  //knee.write((180-kneePos));  sensorValue = analogRead(sensorPin);  if (sensorValue > sensorValueHold){  sensorValueHold = sensorValue;  }  delay(100);  }  sensorP2 = sensorValueHold;  Serial.println("p2 values");  Serial.println(sensorP2, DEC);  Serial.println(hipPos, DEC);  Serial.println(kneePos, DEC);  //C1 Test  delay(500);  kneePos = 165;  hipPos = 90;  for (hipPos > 0; hipPos <= 90; hipPos += 1) { // reset to 90 degrees  hip.write(hipPos); // tell servo to go to position in variable 'pos'  delay(15); // waits 15ms for the servo to reach the position  }  knee.write((165));  delay(100);  sensorValueHold = 0;  for (int a = 0; a < 20; a = a + 1){  DirKnee = C1[a] & KneeMaskDir;  DirHip = C1[a] & HipMaskDir;  MaskedKneePos = ((C1[a] & KneeMaskPos) >> 4);  MaskedHipPos = (C1[a] & HipMaskPos);  if (DirKnee = 0){  kneePos = kneePos + MaskedKneePos;  if (kneePos > upperbound){  kneePos = upperbound;  }  }  else {  kneePos = kneePos - MaskedKneePos;  if (kneePos < lowerbound){  kneePos = lowerbound;  }  }  if (DirHip = 0){  hipPos = hipPos + MaskedHipPos;  if (hipPos > upperbound){  hipPos = upperbound;  }  }  else {  hipPos = hipPos - MaskedHipPos;  if (hipPos < lowerbound){  hipPos = lowerbound;  }  }  hip.write(hipPos);  // knee.write((180-kneePos));  sensorValue = analogRead(sensorPin);  if (sensorValue > sensorValueHold){  sensorValueHold = sensorValue;  }  delay(100);  }  sensorC1 = sensorValueHold;  Serial.println("c1 values");  Serial.println(sensorC1, DEC);  Serial.println(hipPos, DEC);  Serial.println(kneePos, DEC);  //C2 Test  delay(500);  kneePos = 165;  hipPos = 90;  for (hipPos > 0; hipPos <= 90; hipPos += 1) { // reset to 90 degrees  hip.write(hipPos); // tell servo to go to position in variable 'pos'  delay(15); // waits 15ms for the servo to reach the position  }  knee.write((165));  delay(100);  sensorValueHold = 0;  for (int a = 0; a < 20; a = a + 1){  DirKnee = C2[a] & KneeMaskDir;  DirHip = C2[a] & HipMaskDir;  MaskedKneePos = ((C2[a] & KneeMaskPos) >> 4);  MaskedHipPos = (C2[a] & HipMaskPos);  if (DirKnee = 0){  kneePos = kneePos + MaskedKneePos;  if (kneePos > upperbound){  kneePos = upperbound;  }  }  else {  kneePos = kneePos - MaskedKneePos;  if (kneePos < lowerbound){  kneePos = lowerbound;  }  }  if (DirHip = 0){  hipPos = hipPos + MaskedHipPos;  if (hipPos > upperbound){  hipPos = upperbound;  }  }  else {  hipPos = hipPos - MaskedHipPos;  if (hipPos < lowerbound){  hipPos = lowerbound;  }  }  hip.write(hipPos);  // knee.write((180-kneePos));  sensorValue = analogRead(sensorPin);  if (sensorValue > sensorValueHold){  sensorValueHold = sensorValue;  }  delay(100);  }  sensorC2 = sensorValueHold;  Serial.println("c2 values");  Serial.println(sensorC2, DEC);  Serial.println(hipPos, DEC);  Serial.println(kneePos, DEC);  // next parents best of parents and child  if(sensorP1 > sensorP2){  Serial.println("P1 is better");  if (sensorC1 > sensorC2){  Serial.println("child1 is better");  for (int a = 0; a < 20; a = a + 1){  P2[a] = C1[a];  }  }  else {  Serial.println("child 2 is the snitz");  for (int a = 0; a < 20; a = a + 1){  P2[a] = C2[a];  }  }  }  else {  Serial.println("P2 is clearly better");  if (sensorC1 > sensorC2){  Serial.println("replace p1 with c1");  for (int a = 0; a < 20; a = a + 1){  P1[a] = C1[a];  }  }  else {  Serial.println("replace p1 with the snitzc2 ");  for (int a = 0; a < 20; a = a + 1){  P1[a] = C2[a];  }  }  }  hipPos = 90;  for (hipPos > 0; hipPos <= 90; hipPos += 1) { // reset to 90 degrees  hip.write(hipPos); // tell servo to go to position in variable 'pos'  delay(15); // waits 15ms for the servo to reach the position  }  delay(wait);  }  } |

# **Gesture Recognition Code**:

|  |
| --- |
| //------------------------------------------------------------------------------  // <copyright file="MainWindow.xaml.cs" company="Microsoft">  // Copyright (c) Microsoft Corporation. All rights reserved.  // </copyright>  //Modified Microsoft Kinect program  //This program forms a skeleton  //Gets the gesture and sends the gesture information via arduino for serial communication  //Ram Bhattarai, Randon Stasney, Thuan Pham  //------------------------------------------------------------------------------  namespace Microsoft.Samples.Kinect.SkeletonBasics  {  using System.IO;  using System.Windows;  using System.Windows.Media;  using Microsoft.Kinect;  using System.Windows.Media.Media3D;  using System.IO.Ports;  using System.Linq;  using System.Windows.Controls;  //Main Window start of code  public partial class MainWindow : Window  {  int count = 0; //Initialization of the delay variable  int count2 = 0;  int count3 = 0;  int count4 = 0;  /// <summary>  /// Width of output drawing  /// </summary>  private const float RenderWidth = 640.0f;  /// <summary>  /// Height of our output drawing  /// </summary>  private const float RenderHeight = 480.0f;  /// <summary>  /// Thickness of drawn joint lines  /// </summary>  private const double JointThickness = 3;  /// <summary>  /// Thickness of body center ellipse  /// </summary>  private const double BodyCenterThickness = 10;  /// <summary>  /// Thickness of clip edge rectangles  /// </summary>  private const double ClipBoundsThickness = 10;  /// <summary>  /// Brush used to draw skeleton center point  /// </summary>  private readonly Brush centerPointBrush = Brushes.Blue;  /// <summary>  /// Brush used for drawing joints that are currently tracked  /// </summary>  private readonly Brush trackedJointBrush = new SolidColorBrush(Color.FromArgb(255, 68, 192, 68));  /// <summary>  /// Brush used for drawing joints that are currently inferred  /// </summary>  private readonly Brush inferredJointBrush = Brushes.Yellow;  /// <summary>  /// Pen used for drawing bones that are currently tracked  /// </summary>  private readonly Pen trackedBonePen = new Pen(Brushes.Green, 6);  /// <summary>  /// Pen used for drawing bones that are currently inferred  /// </summary>  private readonly Pen inferredBonePen = new Pen(Brushes.Gray, 1);  /// <summary>  /// Active Kinect sensor  /// </summary>  private KinectSensor sensor;  /// <summary>  /// Drawing group for skeleton rendering output  /// </summary>  private DrawingGroup drawingGroup;  /// <summary>  /// Drawing image that we will display  /// </summary>  private DrawingImage imageSource;  /// <summary>  /// Initializes a new instance of the MainWindow class.  /// </summary>  public MainWindow()  {  InitializeComponent();  }  /// <summary>  /// Draws indicators to show which edges are clipping skeleton data  /// </summary>  /// <param name="skeleton">skeleton to draw clipping information for</param>  /// <param name="drawingContext">drawing context to draw to</param>  private static void RenderClippedEdges(Skeleton skeleton, DrawingContext drawingContext)  {  if (skeleton.ClippedEdges.HasFlag(FrameEdges.Bottom))  {  drawingContext.DrawRectangle(  Brushes.Red,  null,  new Rect(0, RenderHeight - ClipBoundsThickness, RenderWidth, ClipBoundsThickness));  }  if (skeleton.ClippedEdges.HasFlag(FrameEdges.Top))  {  drawingContext.DrawRectangle(  Brushes.Red,  null,  new Rect(0, 0, RenderWidth, ClipBoundsThickness));  }  if (skeleton.ClippedEdges.HasFlag(FrameEdges.Left))  {  drawingContext.DrawRectangle(  Brushes.Red,  null,  new Rect(0, 0, ClipBoundsThickness, RenderHeight));  }  if (skeleton.ClippedEdges.HasFlag(FrameEdges.Right))  {  drawingContext.DrawRectangle(  Brushes.Red,  null,  new Rect(RenderWidth - ClipBoundsThickness, 0, ClipBoundsThickness, RenderHeight));  }  }  /// <summary>  /// Execute startup tasks  /// </summary>  /// <param name="sender">object sending the event</param>  /// <param name="e">event arguments</param>  private void WindowLoaded(object sender, RoutedEventArgs e)  {  // Create the drawing group we'll use for drawing  this.drawingGroup = new DrawingGroup();  // Create an image source that we can use in our image control  this.imageSource = new DrawingImage(this.drawingGroup);  // Display the drawing using our image control  Image.Source = this.imageSource;  // Look through all sensors and start the first connected one.  // This requires that a Kinect is connected at the time of app startup.  // To make your app robust against plug/unplug,  // it is recommended to use KinectSensorChooser provided in Microsoft.Kinect.Toolkit (See components in Toolkit Browser).  foreach (var potentialSensor in KinectSensor.KinectSensors)  {  if (potentialSensor.Status == KinectStatus.Connected)  {  this.sensor = potentialSensor;  break;  }  }  if (null != this.sensor)  {  // Turn on the skeleton stream to receive skeleton frames  this.sensor.SkeletonStream.Enable();  // Add an event handler to be called whenever there is new color frame data  this.sensor.SkeletonFrameReady += this.SensorSkeletonFrameReady;  // Start the sensor!  try  {  this.sensor.Start();  }  catch (IOException)  {  this.sensor = null;  }  }  if (null == this.sensor)  {  this.statusBarText.Text = Properties.Resources.NoKinectReady;  }  }  /// <summary>  /// Execute shutdown tasks  /// </summary>  /// <param name="sender">object sending the event</param>  /// <param name="e">event arguments</param>  private void WindowClosing(object sender, System.ComponentModel.CancelEventArgs e)  {  if (null != this.sensor)  {  this.sensor.Stop();  }  }  /// <summary>  /// Event handler for Kinect sensor's SkeletonFrameReady event  /// </summary>  /// <param name="sender">object sending the event</param>  /// <param name="e">event arguments</param>  ///  private void SensorSkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)  {  Skeleton[] skeletons = new Skeleton[0];  using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())  {  if (skeletonFrame != null)  {  skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];  skeletonFrame.CopySkeletonDataTo(skeletons);  }  }  using (DrawingContext dc = this.drawingGroup.Open())  {  // Draw a transparent background to set the render size  dc.DrawRectangle(Brushes.Black, null, new Rect(0.0, 0.0, RenderWidth, RenderHeight));  if (skeletons.Length != 0)  {  foreach (Skeleton skel in skeletons)  {  RenderClippedEdges(skel, dc);  if (skel.TrackingState == SkeletonTrackingState.Tracked)  {  this.DrawBonesAndJoints(skel, dc);  //Joint righthand=skeletons.J  }  else if (skel.TrackingState == SkeletonTrackingState.PositionOnly)  {  dc.DrawEllipse(  this.centerPointBrush,  null,  this.SkeletonPointToScreen(skel.Position),  BodyCenterThickness,  BodyCenterThickness);  }  }  }  // prevent drawing outside of our render area  this.drawingGroup.ClipGeometry = new RectangleGeometry(new Rect(0.0, 0.0, RenderWidth, RenderHeight));  }  }  /// <summary>  /// Draws a skeleton's bones and joints  /// </summary>  /// <param name="skeleton">skeleton to draw</param>  /// <param name="drawingContext">drawing context to draw to</param>  Skeleton GetFirstSkeleton(AllFramesReadyEventArgs e)  {  using (SkeletonFrame skeletonFrameData = e.OpenSkeletonFrame())  {  if (skeletonFrameData == null)  {  return null;  }  Skeleton[] allSkeletons = null;  skeletonFrameData.CopySkeletonDataTo(allSkeletons);  //get the first tracked skeleton  Skeleton first = (from s in allSkeletons  where s.TrackingState == SkeletonTrackingState.Tracked  select s).FirstOrDefault();  return first;  }  }  //Function that creates gesture from the skeleton inforamtion for Left hand and right hand  //Sends information in string to arduino  public void checkHand(Joint head, Joint rhand, Joint lhand)  {  //Assigning a serial port for the arduino communication  SerialPort send = new SerialPort("COM5", 9600);  if (rhand.Position.Y > head.Position.Y && lhand.Position.Y > head.Position.Y)  {  count3++;  System.Console.WriteLine("Both Hands are up");  if (count3 >= 300)  {  send.Open();  send.WriteLine("1:1:1:1:1:0");  send.Close();  count3 = 0;  }  }  else if (rhand.Position.Y < 0 && lhand.Position.Y < 0)  {  // System.Console.WriteLine("Both hands are down");  System.Console.WriteLine(count);  count++;  if (count >= 300)  {  send.Open();  send.WriteLine("0:0:0:0:0:1");  send.Close();  count = 0;  }  }  else  {  if (rhand.Position.Y > head.Position.Y)  {  count2++;  System.Console.WriteLine("RUP");  if (count2 >= 300)  {  send.Open();  send.WriteLine("1:0:0:0:4:0");  send.Close();  count2 = 0;  }  }  else if (lhand.Position.Y > head.Position.Y)  {  count4++;  System.Console.WriteLine("LUP");  if (count4 >= 300)  {  send.Open();  send.WriteLine("0:0:1:0:3:0");  send.Close();  count4 = 0;  }  }  }  }  private void DrawBonesAndJoints(Skeleton skeleton, DrawingContext drawingContext)  {  // Render Torso  this.DrawBone(skeleton, drawingContext, JointType.Head, JointType.ShoulderCenter);  this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter, JointType.ShoulderLeft);  this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter, JointType.ShoulderRight);  this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter, JointType.Spine);  this.DrawBone(skeleton, drawingContext, JointType.Spine, JointType.HipCenter);  this.DrawBone(skeleton, drawingContext, JointType.HipCenter, JointType.HipLeft);  this.DrawBone(skeleton, drawingContext, JointType.HipCenter, JointType.HipRight);  // Left Arm  this.DrawBone(skeleton, drawingContext, JointType.ShoulderLeft, JointType.ElbowLeft);  this.DrawBone(skeleton, drawingContext, JointType.ElbowLeft, JointType.WristLeft);  this.DrawBone(skeleton, drawingContext, JointType.WristLeft, JointType.HandLeft);  // Right Arm  this.DrawBone(skeleton, drawingContext, JointType.ShoulderRight, JointType.ElbowRight);  this.DrawBone(skeleton, drawingContext, JointType.ElbowRight, JointType.WristRight);  this.DrawBone(skeleton, drawingContext, JointType.WristRight, JointType.HandRight);  // Left Leg  this.DrawBone(skeleton, drawingContext, JointType.HipLeft, JointType.KneeLeft);  this.DrawBone(skeleton, drawingContext, JointType.KneeLeft, JointType.AnkleLeft);  this.DrawBone(skeleton, drawingContext, JointType.AnkleLeft, JointType.FootLeft);  // Right Leg  this.DrawBone(skeleton, drawingContext, JointType.HipRight, JointType.KneeRight);  this.DrawBone(skeleton, drawingContext, JointType.KneeRight, JointType.AnkleRight);  this.DrawBone(skeleton, drawingContext, JointType.AnkleRight, JointType.FootRight);  // Render Joints  foreach (Joint joint in skeleton.Joints)  {  Brush drawBrush = null;  if (joint.TrackingState == JointTrackingState.Tracked)  {  drawBrush = this.trackedJointBrush;  }  else if (joint.TrackingState == JointTrackingState.Inferred)  {  drawBrush = this.inferredJointBrush;  }  if (drawBrush != null)  {  drawingContext.DrawEllipse(drawBrush, null, this.SkeletonPointToScreen(joint.Position), JointThickness, JointThickness);  }  }  }  /// <summary>  /// Maps a SkeletonPoint to lie within our render space and converts to Point  /// </summary>  /// <param name="skelpoint">point to map</param>  /// <returns>mapped point</returns>  private Point SkeletonPointToScreen(SkeletonPoint skelpoint)  {  // Convert point to depth space.  // We are not using depth directly, but we do want the points in our 640x480 output resolution.  DepthImagePoint depthPoint = this.sensor.CoordinateMapper.MapSkeletonPointToDepthPoint(skelpoint, DepthImageFormat.Resolution640x480Fps30);  return new Point(depthPoint.X, depthPoint.Y);  }  /// <summary>  /// Draws a bone line between two joints  /// </summary>  /// <param name="skeleton">skeleton to draw bones from</param>  /// <param name="drawingContext">drawing context to draw to</param>  /// <param name="jointType0">joint to start drawing from</param>  /// <param name="jointType1">joint to end drawing at</param>  private void DrawBone(Skeleton skeleton, DrawingContext drawingContext, JointType jointType0, JointType jointType1)  {  Joint joint0 = skeleton.Joints[jointType0];  Joint joint1 = skeleton.Joints[jointType1];  // If we can't find either of these joints, exit  if (joint0.TrackingState == JointTrackingState.NotTracked ||  joint1.TrackingState == JointTrackingState.NotTracked)  {  return;  }  // Don't draw if both points are inferred  if (joint0.TrackingState == JointTrackingState.Inferred &&  joint1.TrackingState == JointTrackingState.Inferred)  {  return;  }  // We assume all drawn bones are inferred unless BOTH joints are tracked  Pen drawPen = this.inferredBonePen;  if (joint0.TrackingState == JointTrackingState.Tracked && joint1.TrackingState == JointTrackingState.Tracked)  {  drawPen = this.trackedBonePen;  }  drawingContext.DrawLine(drawPen, this.SkeletonPointToScreen(joint0.Position), this.SkeletonPointToScreen(joint1.Position));  Joint righthand = skeleton.Joints[JointType.HandRight];  Joint head = skeleton.Joints[JointType.Head];  Joint lefthand = skeleton.Joints[JointType.HandLeft];  Joint rleg = skeleton.Joints[JointType.FootRight];  Joint lleg = skeleton.Joints[JointType.FootLeft];  checkHand(head, righthand, lefthand);  }  /// Handles the checking or unchecking of the seated mode combo box  //object sending the event  //event arguments  private void CheckBoxSeatedModeChanged(object sender, RoutedEventArgs e)  {  if (null != this.sensor)  {  if (this.checkBoxSeatedMode.IsChecked.GetValueOrDefault())  {  this.sensor.SkeletonStream.TrackingMode = SkeletonTrackingMode.Seated;  }  else  {  this.sensor.SkeletonStream.TrackingMode = SkeletonTrackingMode.Default;  }  }  }    }  } |

# **Speech Recognition Code**:

|  |
| --- |
| //------------------------------------------------------------------------------  // <copyright file="Program.cs" company="Microsoft">  // Copyright (c) Microsoft Corporation. All rights reserved.  // </copyright>  //------------------------------------------------------------------------------  // This module provides sample code used to demonstrate the use  // of the KinectAudioSource for speech recognition  //Modifed Version of program  //Added couple of lines of commands, removed the portion of code that was not necessary  //Added the arduino code  //Modifed the sample version of code to be effective and useful.  //Ram Bhattarai, Thuan Pham, Randon Stasney  namespace Speech  {  using System;  using System.IO;  using System.IO.Ports;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading;  using Microsoft.Kinect;  using Microsoft.Speech.AudioFormat;  using Microsoft.Speech.Recognition;  public class Program  {  public static void Main(string[] args)  {  // Obtain a KinectSensor if any are available  KinectSensor sensor = (from sensorToCheck in KinectSensor.KinectSensors where sensorToCheck.Status == KinectStatus.Connected select sensorToCheck).FirstOrDefault();  if (sensor == null)  {  Console.WriteLine(  "No Kinect sensors are attached to this computer or none of the ones that are\n" +  "attached are \"Connected\".\n" +  "Attach the KinectSensor and restart this application.\n" +  "If that doesn't work run SkeletonViewer-WPF to better understand the Status of\n" +  "the Kinect sensors.\n\n" +  "Press any key to continue.\n");  // Give a chance for user to see console output before it is dismissed  Console.ReadKey(true);  return;  }  sensor.Start();  // Obtain the KinectAudioSource to do audio capture  KinectAudioSource source = sensor.AudioSource;  source.EchoCancellationMode = EchoCancellationMode.None; // No AEC for this sample  source.AutomaticGainControlEnabled = false; // Important to turn this off for speech recognition  RecognizerInfo ri = GetKinectRecognizer();  if (ri == null)  {  Console.WriteLine("Could not find Kinect speech recognizer. Please refer to the sample requirements.");  return;  }  Console.WriteLine("Using: {0}", ri.Name);  // NOTE: Need to wait 4 seconds for device to be ready right after initialization  int wait = 4;  while (wait > 0)  {  Console.Write("Device will be ready for speech recognition in {0} second(s).\r", wait--);  Thread.Sleep(1000);  }  using (var sre = new SpeechRecognitionEngine(ri.Id))  {  //Here are choices of commands to use  var commands = new Choices();  commands.Add("right arm");  commands.Add("right leg");  commands.Add("left leg");  commands.Add("left arm");  commands.Add("head");  commands.Add("funny");  commands.Add("kick");  commands.Add("wave");  commands.Add("walk");  commands.Add("salute");  commands.Add("badboy");  commands.Add("sprint");  commands.Add("kneel");  commands.Add("stroll");  var gb = new GrammarBuilder { Culture = ri.Culture };  // Specify the culture to match the recognizer in case we are running in a different culture.  gb.Append(commands);  // Create the actual Grammar instance, and then load it into the speech recognizer.  var g = new Grammar(gb);  sre.LoadGrammar(g);  sre.SpeechRecognized += SreSpeechRecognized;  sre.SpeechRecognitionRejected += SreSpeechRecognitionRejected;  using (Stream s = source.Start())  {  sre.SetInputToAudioStream(  s, new SpeechAudioFormatInfo(EncodingFormat.Pcm, 16000, 16, 1, 32000, 2, null));  //Prints out the commands that needs to be talked.  Console.WriteLine("Recognizing speech. Say:'right arm', 'right leg', 'left leg', 'left arm','head','funny','kick','Wave','walk','salute','badboy','sprint','kneel','stroll'");  sre.RecognizeAsync(RecognizeMode.Multiple);  Console.ReadLine();  Console.WriteLine("Stopping recognizer ...");  sre.RecognizeAsyncStop();  }  }  sensor.Stop();  }  private static RecognizerInfo GetKinectRecognizer()  {  Func<RecognizerInfo, bool> matchingFunc = r =>  {  string value;  r.AdditionalInfo.TryGetValue("Kinect", out value);  return "True".Equals(value, StringComparison.InvariantCultureIgnoreCase) && "en-US".Equals(r.Culture.Name, StringComparison.InvariantCultureIgnoreCase);  };  return SpeechRecognitionEngine.InstalledRecognizers().Where(matchingFunc).FirstOrDefault();  }  //If the voice is not loud enough or voice does not exist in voice commands, the program will just throw the error message.  private static void SreSpeechRecognitionRejected(object sender, SpeechRecognitionRejectedEventArgs e)  {  Console.WriteLine("\nSpeech Rejected");  if (e.Result != null)  {  Console.WriteLine("Incorrect Entry");  }  }  //Function that sends voice commands to arduino.  //Voice has to be clear when you say the commands  //VOice will not be processed if the voice command is not perfect  private static void SreSpeechRecognized(object sender, SpeechRecognizedEventArgs e)  {  SerialPort send = new SerialPort("COM5", 9600);  if (e.Result.Confidence >= 0.7)  {  Console.WriteLine("\nSpeech Recognized: \t{0}\tConfidence:\t{1}", e.Result.Text, e.Result.Confidence);  if (e.Result.Text == "right arm")  {  Console.WriteLine(0);  send.Open();  send.WriteLine("1:0:0:0:0:0");  send.Close();  }  else if (e.Result.Text == "right leg")  {  Console.WriteLine(1);  send.Open();  send.WriteLine("0:1:0:0:0:0");  send.Close();  }  else if (e.Result.Text == "left leg")  {  Console.WriteLine(2);  send.Open();  send.WriteLine("0:0:0:1:0:0");  send.Close();  }  else if (e.Result.Text == "left arm")  {  Console.WriteLine(3);  send.Open();  send.WriteLine("0:0:1:0:0:0");  send.Close();  }  else if(e.Result.Text=="funny")  {  Console.WriteLine(4);  send.Open();  send.WriteLine("1:1:1:1:1:0");  send.Close();  }  else if (e.Result.Text == "kick")  {  Console.WriteLine(6);  send.Open();  send.WriteLine("0:2:0:0:2:0");  send.Close();  }  else if (e.Result.Text == "wave")  {  Console.WriteLine(7);  send.Open();  send.WriteLine("1:0:1:0:1:0");  send.Close();  }  else if (e.Result.Text == "walk")  {  Console.WriteLine(8);  send.Open();  send.WriteLine("0:1:0:1:2:0");  send.Close();  }  else if (e.Result.Text == "salute")  {  Console.WriteLine(9);  send.Open();  send.WriteLine("1:0:0:2:2:0");  send.Close();  }  else if (e.Result.Text == "badboy")  {  Console.WriteLine(10);  send.Open();  send.WriteLine("0:0:0:0:5:0");  send.Close();  }  else if (e.Result.Text == "sprint")  {  Console.WriteLine(11);  send.Open();  send.WriteLine("0:3:0:3:0:0");  send.Close();  }  else if (e.Result.Text == "kneel")  {  Console.WriteLine(12);  send.Open();  send.WriteLine("0:4:0:4:6:0");  send.Close();  }  else if (e.Result.Text == "stroll")  {  Console.WriteLine(13);  send.Open();  send.WriteLine("0:0:0:0:0:1");  send.Close();  }  else if (e.Result.Text == "head")  {  Console.WriteLine(5);  send.Open();  send.WriteLine("0:0:0:0:1:0");  send.Close();  }  }  else  {  Console.WriteLine("\nSpeech Recognized but confidence was too low: \t{0}", e.Result.Confidence);  Console.WriteLine("Please try Again");  }  }  }  } |

# 

# 

# **Youtube Links**:

Frankenstein Final Voice Recognition

<https://www.youtube.com/watch?v=bfsBNZnKE-c>

Frankenstein Gesture:

<https://www.youtube.com/watch?v=LfHExKT95BI>

Fuzzy Logic Demo:

<https://www.youtube.com/watch?v=UKbPCnX2sfU&feature=youtu.be>

Genetic Algorithm:

<https://www.youtube.com/watch?v=yKYRfGHoIF0>

Kick that Broke the Servo:

<https://www.youtube.com/watch?v=3OwX1x9Urp8>