# Marie-Curie Report

# Fall 2013

In this document we describe what we started with, how the hardware is connected, and her movement and position library, what we did to modify Marie's appearance, the music composition algorithm, the new arm and the IGA for the facial expressions.

By: Kevin Bedrossian, Brett Dunscomb & Caren Zgheib

# Table of Contents

Table of Contents ..... 2
Table of Figures ..... 2
A Brief Overview ..... 3
What We Started With ..... 3
Hardware & Connections ..... 4
Degrees of Freedom ..... 4
Connections ..... 5
Movement and Position Library ..... 6
Additions ..... 7
A Second Look at Marie ..... 8
Mechanical Changes ..... 9
Robotic Hand Instructions ..... 9
Music Composition ..... 13
Interactive Genetic Algorithm ..... 17
Appendix ..... 19
Marie_Curie.h ..... 19
Marie_Curie.cpp ..... 21
Arduino code ..... 30
IGA Code (From Bohr Robot) ..... 33
Hours Spent ..... 69
References ..... 69

# Table of Figures

Figure 1- Marie-Curie (before) ..... 3
Figure 2- View of the head from the back ..... 4
Figure 3- Servos controlling the head. (Numbers do not match the pin numbers on the Pololu board) ..... 5
Figure 4- Marie Curie after modification ..... 8
Figure 5- Pololu Maestro Editor ..... 9
Figure 6- The Simple Map [1] ..... 14
Figure 7- Key for the Simple Map [1] ..... 15
Figure 8- IGA Menu Adapted from Bohr ..... 17

## A Brief Overview

The purpose of this project is to “resurrect” the robot called Marie-Curie. This process requires getting the code that previous teams used to upload to the Arduino board in the head, modify the body and fix the mechanical aspects of Marie. Also fix and improve the arms and add code to gain control of the servos in the arms as well as the rest of the servos in the head to increase facial features (not used by previous teams).

## What We Started With

Marie looked something like this:

Figure 1- Marie-Curie (before)

There was some code uploaded on the board which was able to control some servos in the head including the eyes, jaw and neck. The arms were disconnected as well as many servos inside the head.

The neck was too long, the left ear was not attached to the head properly and finally the source code was not available.

Therefore, our main task was to track down the people that worked on Marie to be able to get the source code from them and analyze it to figure out what we are dealing with. Moreover, another task was to modify the shape of the shoulders and arms to look more humanoid-like in addition to modifying the code to control all degrees of freedom.

## Hardware & Connections

An Arduino microcontroller and a Pololu SSC03A servo controller are used to control the 4 servos in the left arm. Another Pololu controller (mini maestro 18 channel) was used to control the remaining servos in her face and right arm.

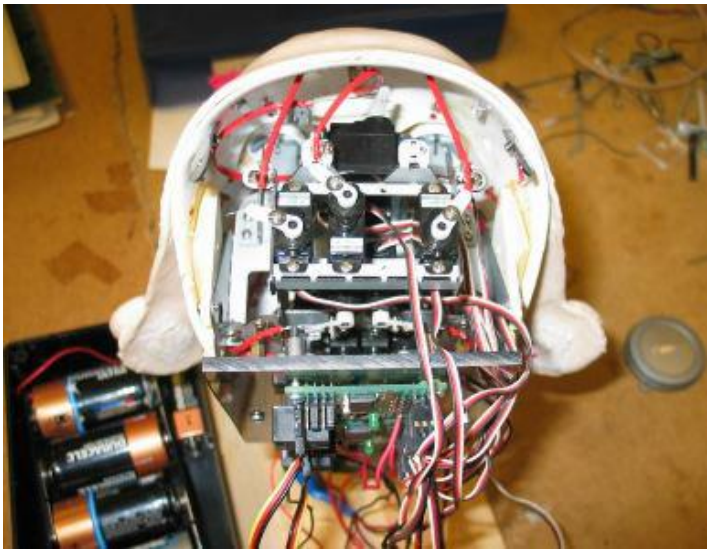


Figure 2- View of the head from the back

## Degrees of Freedom

In the head alone, there are 15 degrees of freedom (15 servos).

In addition, there are 3 degrees of freedom in each arm but some of the servos in the right arm needed to be replaced so we ended up replacing that arm with a better one described below. The right arm has additional 4 degrees of freedom to control the fingers. In addition, there is 1 degree of freedom in the waist.

Therefore, we have a total of 26 degrees of freedom.

Following is an image showing the different servos present in Marie's head. *(Note: the numbers **do not** match the current pin numbers used on the Pololu board. The connections will be discussed in the next section)*.

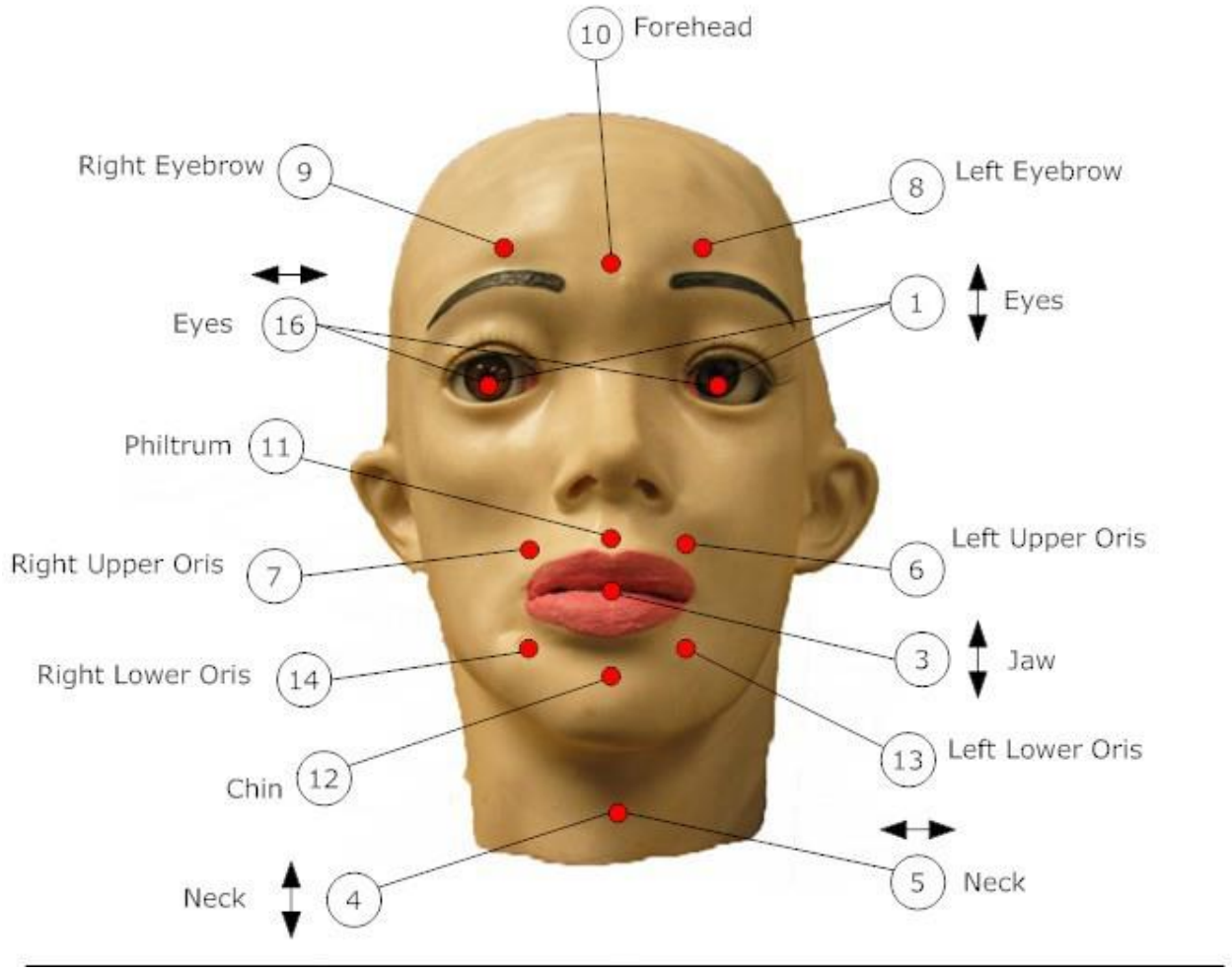



Figure 3- Servos controlling the head. (Numbers do not match the pin numbers on the Pololu board)

## Connections

The Arduino-Pololu Connections:

- Arduino GND to Pololu GND
- Arduino 5V to Pololu VIN
- Arduino Digital 7 to Pololu logic-level serial input
- Arduino Digital 5 to Pololu reset
- Arduino Digital 6 is reserved for the Arduino serial RX connection and must be left disconnected

The Pololu SSC03A -Servo Connections:

- Pololu Servo 2 to left shoulder servo
- Pololu Servo 3 to left elbow servo
- Pololu Servo 4 to left arm up/down servo
- Pololu Servo 5 to torso servo

The Pololu mini maestro 18 channel controller – servo connections:

- Pololu Servo 11 to right arm up/down servo
- Pololu Servo 12 to right elbowservo
- Pololu Servo 13 to pointer finger servo
- Pololu Servo 14 to pinky servo
- Pololu Servo 15 to middle finger servo
- Pololu Servo 16 to ring finger servo
- Pololu Servo 17 to shoulder servo
- Pololu Servo 0 lower right oris
- Pololu Servo 1 left eyebrow
- Pololu Servo 2 right eyebrow
- Pololu Servo 3 lower left oris
- Pololu Servo 4 philtrum
- Pololu Servo 5 forehead
- Pololu Servo 6 upper right oris
- Pololu Servo 7 chin
- Pololu Servo 8 upper left oris

The Pololu-Power Connections:

- Pololu servo power +/- to battery pack (or 6V power supply)

## Movement and Position Library

The base for this library has been provided by a previous team that worked on Marie-Curie. Here is what they did and our additions and modifications:

Marie's library consists of a C++ class called *marie_curie*. This class has functions that control motions and macros for the pins, positions and variables that store the position of each servo.

The class is defined in the files *marie_curie.h* and *marie_curie.cpp* that can be found in the appendix.

Each function in the code is described by its name. For example, the function "turnRight" turn Marie's head to the right. The function "openMouth" opens her mouth etc. …

In addition, in the document, the previous team explained what the other *non-trivial* functions mean: *Function names that begin with "saccade" cause her to move her eyes (a saccade is an eye movement). Saccade functions that include the letters "UD" cause her to move her eyes up or down, while saccade functions that include the letters "LR" cause her to move her eyes left or right. Function names that end with the word "to", such as turnto(), tiltto(), or saccadeUDto() allow the position to be set by either the programmer or a variable.*

## Additions

After using this library to test all the servos that it can control, we used the same code to make sure that the remaining servos are working properly. We used the code for the eyes to control the arms, the shoulders and the waist. This allowed us to identify the broken servos and add functions to control the remaining motors in a similar way as the ones in the head: rightArmUp(), rightArmDown() (those functions were used on the left arm instead), shoulderTo(byte target), elbowTo(byte target) and torsoTo(byte target) are some of those functions.

The composeMusic() and playNote(char note) functions are explained in detail below.

## A Second Look at Marie

Figure 4- Marie Curie after modification

In addition to working on the software, we made sure that we were not neglecting the hardware. We changed the shape of the shoulders to look more like human shoulders. We shortened the neck and fixed the left ear so that it's not loose anymore. We moved the servo that was supposed to control the waist and put it on her right shoulder since it can support more weight than the other servos. We rebuilt her base to give her a more stable platform to rotate around. Moreover, hair and a shirt gave Marie a human-like appearance.

Marie is supposed to play the xylophone so that she can fit in with the rest of her peers in Dr. Perkowski's robot theater. Therefore, we attached the xylophone to the base so that it stays at a known relative position from Marie. In addition, we modified her right arm to be more human-like.

## Mechanical Changes

We have ordered a new Pololu board that allows us to have simultaneous control over all of her degrees of freedom. We used the Polulu editor (shown in figure 5) to fine tune the positions of the servos and make sure that the 'tasks' that Marie should perform are actually achievable with the servos that we have. This was done using the editor by moving the servos to known positions and checking the result ourselves and making sure that what we expect is actually being implemented.

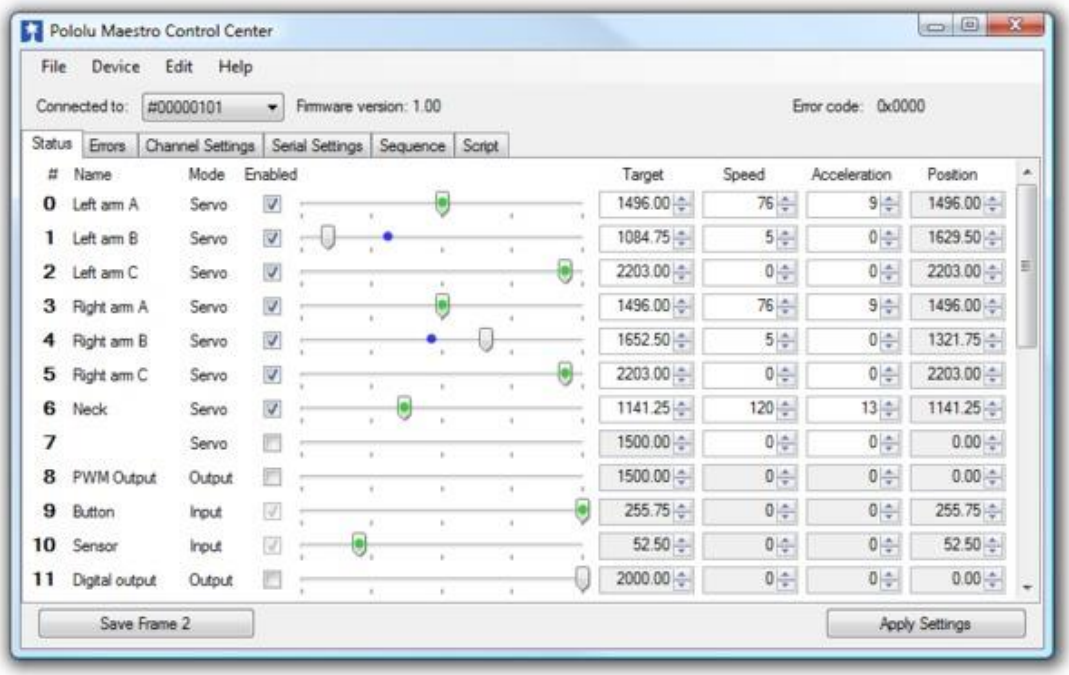



Figure 5- Pololu Maestro Editor

The xylophone that we found in the lab can also be played like a keyboard. We decided to use it in that way since it would be easier for Marie to hit the right notes. Also, in order to identify the notes on the xylophone, we used a tuner application to detect the notes and we labeled them on the keys of the instrument. This way, when Marie is composing music we can tell directly which note she is playing.

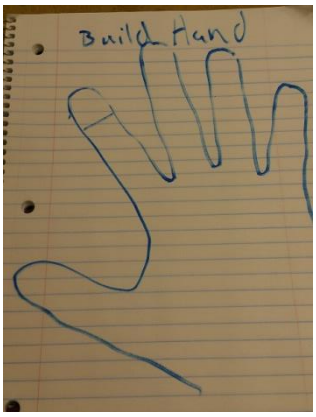
## Robotic Hand Instructions

These are the instructions to build a cheap robotic hand with hand like characteristics. It's a good alternative to more expensive options. To build this robot you need very view materials. (cf. ref. [3])

**Parts List:**

- Tubing around 4 feet.
- Fishing line or nylon twine.
- Utility knife or tool for cutting
- Electrical tape for the fingers

1. To build the hand you want to use a trace outline of your hand.




2. Draw lines where your finger joints are


3. Lay the tub over your drawing and mark on the tube where your joints are

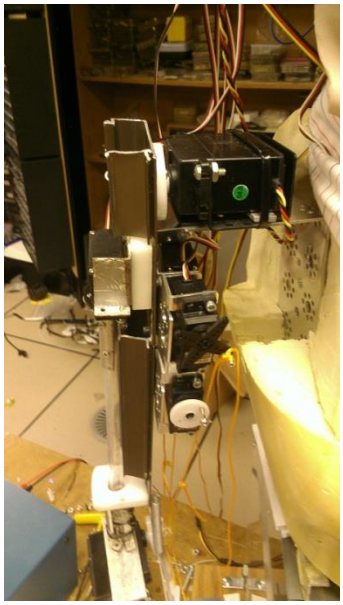
4. Cut 45 degree angles at the locations where you marked your fingers at

5. Repeat this for the other fingers.

6. Attach the string to the end of the finger by either tying it at the end or use tape on the end and run it through the finger.

7. Create a brace for the fingers and insert each finger into the whole.

8. Attach the strings to the servos you want to control.



9. What the final product should look like when done.

## *Goal*

The goal of building this hand was to increase the realism of Marie Curie. There were many constraints to consider when building the hand. The biggest factor was cost but we also wanted to make sure we considered a simple solution that would be easy to construct. Brett was tasked to find a cheap way to build a functional hand which can mimic human movements.

He found some suggestions online and which gave him some ideas on the approach.

## *Purchasing parts*

The total cost of materials came to about $20 dollars. These included nylon string, electrical tape, and plastic tubing around 5/8 inch in diameter. (At a local Home Depot)

## *Design Considerations*

When designing the hand there were many considerations to keep. Realism was the key, and to meet that, if each finger could move individually it would help facilitate the feel of a real hand. To accomplish this we would need a servo for each finger. Another large design consideration was the weight of each servo to create a hand. The placement of the servos also has a large impact on the design. If they are too far down the arm they can increase the strain on the shoulder servo.

To assist with the design Brett mounted the servos in various locations with tape and moved the arm to see how well it performed. If the servos were mounted on the forearm both the elbow and the shoulder had difficulty moving. When close to the shoulder the elbow was able to move and the shoulder could still move. The movement of the shoulder was limited however. Because of this and room on the shoulder he decided to try and limit the amount of servos to use and settled on four servos to meet this requirement. This is an acceptable tradeoff because the thumb and the pointer would be used together which would only require one servo.

The servos were now mounted on the shoulder. This caused another issue to be solved: how to get them to control the hands on the end of the arm. The use of a pulley system to transfer the servos movement to control the hands seemed the right way to go. With the pulleys mounted on the arms pivot point when the arm is moved it doesn't affect the tension on the string keeping consistent tension and not moving the hands.

## *Final thoughts*

Designing and building the hand was a challenging aspect not only construction but also the techniques to implement a 5 finger hand with the ability to hold something. The results of the design and final version work well and give a great effect to simulate a human hand. There are some aspects that do need to be further improved. The recoil of the fingers is a little slow. Adding some elastic to the top of the fingers to help them recoil would help. A shortcoming of the hand is that it isn't able to hold much wait. The hand currently can only hold very light objects and pick up crumpled up papers. This would be an area as well for further improvement.

## Music Composition

We created two new functions playNote(char note) and composeMusic() which give Marie to play user specified notes as well as compose her own melodies!

Here is how each of those functions work:

I- playNote(char note) takes a note character ('A', 'B', 'C', 'D', 'E', 'F' and 'G') each character corresponds to a note on the xylophone keyboard. This function is basically a switch case function that depends on that input note. For each note, a certain number of servos is activated and moved to the correct position and eventually plays the note. In addition, there is an extra "note" which is basically not really a note but a "silence" (the character used as input is 'S') that takes Marie to her initial position without playing any note.

Silences are actually used by musicians to keep the rhythm and flow of the melodies that they compose and play.

We used the Pololu Maestro editor to find the servo positions for each note. However, the drawback of using the editor was that the servo positions were given in µs and we needed them to be in the byte range: 0-255.

Therefore, we used the following conversion equation to convert the positions from µs to byte:

$$byteTarget = \frac{(\mu sTarget - 500) \times 255}{2000}$$

This equation gave us positions that are very close to what we needed but they were still off by a little bit. In order to fine tune, we had to go over each position again and change the servo target values by trial and error.

Here is a cheat sheet that can help in future fine tuning:

- shoulder servo:
  - increase value → go forward
  - decrease value → go backward
- elbow servo:
  - increase value → move arm further to the right
  - decrease value → move arm further to the left
- torso sevo:
  - increase value → turn right
  - decrease value → turn left

II- composeMusic()

This function takes no input. It allows Marie to compose her own melodies as long as they conform to the "Simple Map"[1]. Figure 6 below shows the map that we are using:

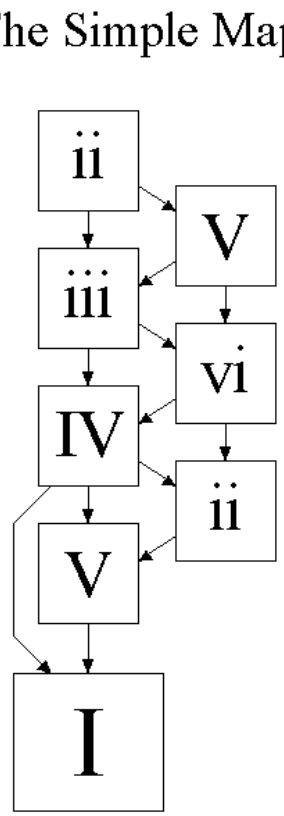



Figure 6- The Simple Map [1]

This map is used by song writers to allow them to compose music in a certain mood by selecting the chords that they use in a pseudo-random way that keeps the flow and mood of the melody.

In our project, we use this map for the notes instead of the chords due to limitations in the instrument and in Marie's arms. We can only play one note at a time. Therefore, when analyzing this map, we will be using the letters as notes NOT chords.

Here is the key that is used with the map in order to understand it better:

| KEY | I | ii | iii | IV | V | vi |
|---|---|---|---|---|---|---|
| C | C | Dm | Em | F | G | Am |
| Db | Db | Ebm | Fm | Gb | Ab | Bbm |
| D | D | Em | F#m | G | A | Bm |
| Eb | Eb | Fm | Gm | Ab | Bb | Cm |
| E | E | F#m | G#m | A | B | C#m |
| F | F | Gm | Am | Bb | C | Dm |
| Gb | Gb | Abm | Bbm | Cb | Db | Ebm |
| G | G | Am | Bm | C | D | Em |
| Ab | Ab | Bbm | Cm | Db | Eb | Fm |
| A | A | Bm | C#m | D | E | F#m |
| Bb | Bb | Cm | Dm | Eb | F | Gm |
| B | B | C#m | D#m | E | F# | G#m |

Figure 7- Key for the Simple Map [1]

First of all, we need to pick a Key. For this project, we are using C as our key. Therefore, the numbers in the map will correspond to the note letters in the following way:
I = C, ii = D, iii = E, IV = F, V = G and vi = A.
Notice that we do not use the note B to keep the mood of the melody. Also the "capital vs. non-capital numbers" is irrelevant in our case since we are using notes not chords to compose our music.

Now, here is how this searching for the right note happens in our code: (cf. ref [2])
From 'C' Marie can play any note on the map. But, once that second note is played, there are rules which are the edges on the map. For example, if she plays 'E', she can only play either 'F' or 'A' or a silent (but the silent doesn't change the fact that the last noted played was an 'E').

To keep track of all those rules, we created an 8x9 matrix that we call "adjacency matrix".
If there is an edge connecting note[i] with note[j] the value is 1. Otherwise it is 0.

```
void marie_curie::composeMusic(){
	bool adjacency [8][9] =
	{
		{1,1,1,1,1,1,1,1,1},
		{0,1,1,1,0,0,0,0,0},
		{0,0,1,1,1,0,0,0,1},
		{0,0,0,1,1,1,0,0,1},
		{0,0,0,0,1,1,1,0,1},
```

```
		{1,0,0,0,0,1,1,1,1},
		{0,0,0,0,0,0,1,1,1},
		{1,0,0,0,0,0,0,1,1}
	};

	playNote('C');
	int index = 0;
	int temp_index = 0;

	for (int i=0; i<20; i++)
	{
		// generate random number between 0 and 8
		temp_index = rand() % 9;
		if(adjacency[index][temp_index] == 1)
		{
			switch(temp_index)
			{
			case 0: // C
				playNote('C');
				index = 0;
				break;
			case 1: // D after a C
				playNote('D');
				index = 1;
				break;
			case 2: // G after only a D
				playNote('G');
				index = 2;
				break;
			case 3: // E
				playNote('E');
				index = 3;
				break;
			case 4: // A
				playNote('A');
				index = 4;
				break;
			case 5: // F
				playNote('F');
				index = 5;
				break;
			case 6: // D after A or F
				playNote('D');
				index = 6;
				break;
			case 7: // G after D or F
				playNote('G');
				index = 7;
				break;
			case 8: // Silent note. Stay on current index.
				playNote('S');
				break;
			default:
				break;
			}
		}
	}
}
```

*For future teams working on Marie, the composeMusic() function is not currently in use because the arduino is not reading the for loop correctly. We recommend using a sensor to fine tune the note playing function and use that in the composition algorithm. We have verified that the code works in theory but did not have enough time to test it.*

## Interactive Genetic Algorithm

The IGA has been adapted from the Bohr robot. The most major change we made to adapt this code to work for Marie-Curie was to use serial port communication through a mini USB cable to the Pololu Maestro 18. The code was originally designed to use a USB to RS232 cable to communicate with a Pololu device.

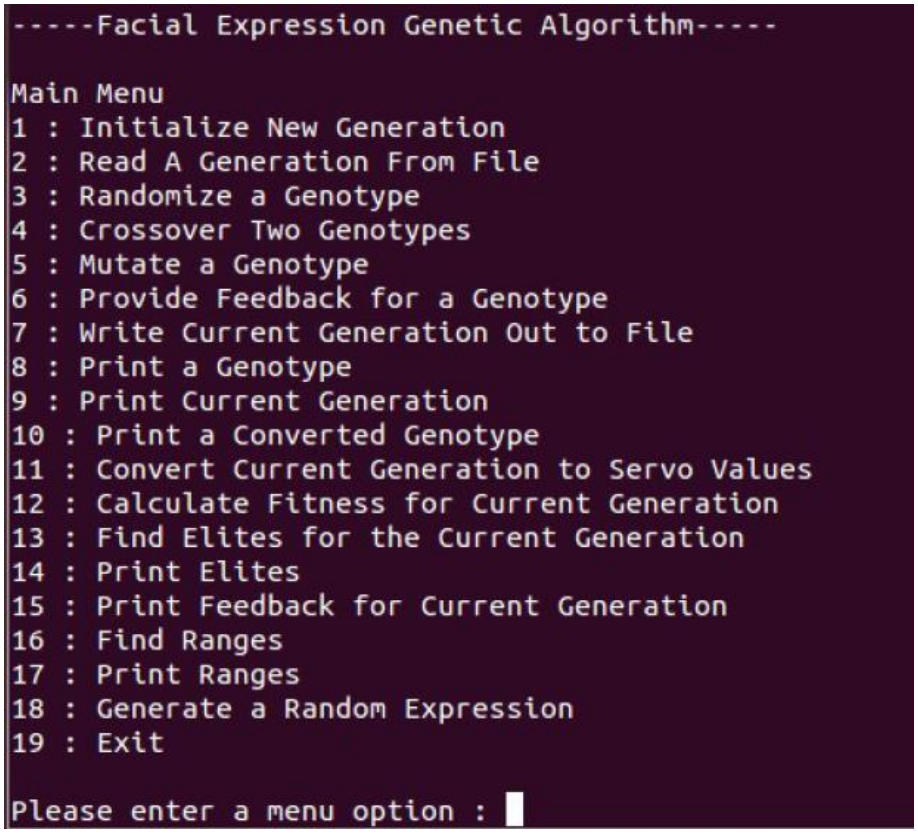



Figure 8- IGA Menu Adapted from Bohr

Figure 8 shows the main menu for the IGA. Here is how we operate the IGA:

First, initialize a new generation. Then, get the servo ranges from a saved file: “servorange.txt”. After these two steps are done, the next step is to send a genotype to the robot. We currently have it set up to use 8 genotypes (sets of servo locations) per generation with 9 genes (servos).

The program is currently capable of performing a crossover of two genes, mutating a genotype, randomizing a genotype. The ‘feedback’ and ‘find elites’ functionality is somewhat limited as evolving genotypes with this method takes an unreasonably long period of time to make progress toward evolving a specific type of emotion on the face.

# Appendix

## Marie_Curie.h

Following is the code for the header file: marie_curie.h

*Note: Some of those functions are not used / commented out in our final code because we use the head in the IGA, the left arm in the music playing and the right arm with the editor.*

```
#ifndef marie_h
#define marie_h

#include "WProgram.h"
#include "SoftwareSerial.h"

#define TXPIN 7
#define RXPIN 6

#define RSHOULDERFBPIN      2
#define RELBOWRLPIN         3
#define RARMUDPIN           4
#define TORSOPIN            5
#define JAWPIN                      7 // not used
//#define HEADTURNPIN 15
//#define TILTPIN 14
// NOD


//#define EYEUDPIN 3
//#define EYELRPIN 2



#define MAXTURNRIGHT        65
#define MAXTURNLEFT                 225
#define TURNFORWARD                 135
#define MAXNODDOWN                  254
#define MAXNODUP                    15
#define NODAHEAD                    175
#define MAXTILTRIGHT        30
#define MAXTILTLEFT                 6
#define TILTLEVEL                   20
#define MOUTHOPEN                   45
#define MOUTHCLOSE                  175
#define MAXEYERIGHT                 200
#define MAXEYELEFT                  50
#define EYEFORWARD                  135
#define MAXEYEUP                    10
#define MAXEYEDOWN                  240
#define EYEAHEAD                    175
#define STRAIGHT                    112
#define PLAYINGPOS                  86
#define MAXARMUP                    86
#define MAXARMDOWN                  48
```

```
#define ELBOWSTRAIGHT             108
#define TORSOSTRAIGHT             130


class marie_curie {

        public:
                marie_curie();
                void turnRight();
                void turnLeft();
                void turnForward();
                void turnto(byte);
                void nodUp();
                void nodDown();
                void nodAhead();
                void nodto(byte);
                void tiltRight();
                void tiltLeft();
                void tiltLevel();
                void tiltto(byte);
                void openMouth();
                void closeMouth();
                void saccadeRight();
                void saccadeLeft();
                void saccadeForward();
                void saccadeLRto(byte);
                void saccadeUp();
                void saccadeDown();
                void saccadeAhead();
                void saccadeUDto(byte);
                void turnEyesRight();
                void turnEyesLeft();
                void rightArmUp();
                void rightArmDown();
                void shoulderTo(byte target);
                void elbowTo(byte target);
                void torsoTo(byte target);

                // Music Composition
                void playNote(char note);
                void composeMusic();

        private:
                //SoftwareSerial tellMarie;
                byte turnPos;
                byte tiltPos;
                byte shouldPos;
                byte nodPos;
                byte jawPos;
                byte eyeUDPos;
                byte eyeLRPos;
                byte armPos;
                byte elbowPos;
                byte torsoPos;

byte convert(byte target)
{
        byte result;
```

```
        result = ((target - 500) * 255 ) / 2000;
        return result;
}
};

#endif
```

## Marie_Curie.cpp

Here is source code for marie_curie.cpp

Note from the previous team: *In order to reduce the jerkiness of Marie's movements, many of these functions utilize a gradual motion loop that slowly moves a servo from its current position to the desired position. The rate of this motion is controlled by a macro called SPEED, which is defined in the marie_curie.cpp file. We have found that a value of 5 produces a smooth, but rapid speed for most motions, but this can be easily changed if future groups find it to be too slow or too fast for their purposes. It would also be possible to add a few speed macros and use different speeds in different functions. Neither the eye movements nor the jaw movements use gradual motion since humans tend to make these motions rather quickly.*

*Another note: most of the functions are commented out because they are not used in the music composition and playing part. The functions are tested and known to work. For future teams to enable this functionality simply uncomment the desired functions and set the correct pin values and call those functions in arduino.*

```
#include "WProgram.h"
#include "marie_curie.h"
#include "SoftwareSerial.h"

#define SPEED 5
#define RESET 5

SoftwareSerial tellMarie(RXPIN,TXPIN);

marie_curie::marie_curie(){

        //Initialize Positions
        turnPos             = TURNFORWARD;
        tiltPos             = TILTLEVEL;
        shouldPos     = convert(STRAIGHT);
        nodPos        = NODAHEAD;
        jawPos        = MOUTHOPEN;
        eyeUDPos      = EYEAHEAD;
        eyeLRPos      = EYEFORWARD;
        armPos        = convert(PLAYINGPOS);
        elbowPos      = convert(ELBOWSTRAIGHT);
        torsoPos      = convert(TORSOSTRAIGHT);

        Serial.println("Finished Setup");

        // Create Serial interface
        //const SoftwareSerial tM(RXPIN,TXPIN);
        //tellMarie = &tM;
```

```
	pinMode(TXPIN,OUTPUT);
	pinMode(RXPIN,INPUT);	// Not useful since polulo doesn't respond, but need
for soft ser.
	tellMarie.begin(9600);
	Serial.begin(9600);
	/*
	// Initialize Marie's positions;
	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(TURNPIN,BYTE);			// Servo address
	tellMarie.println(turnPos,BYTE);	// Servo position

	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(NODPIN,BYTE);			// Servo address
	tellMarie.println(nodPos,BYTE);	// Servo position

	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(TILTPIN,BYTE);			// Servo address
	tellMarie.println(tiltPos,BYTE);	// Servo position

	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(JAWPIN,BYTE);			// Servo address
	tellMarie.println(jawPos,BYTE);		// Servo position

	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(EYELRPIN,BYTE);			// Servo address
	tellMarie.println(eyeLRPos,BYTE);	// Servo position
	*/
	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(RARMUDPIN,BYTE);	// Servo address
	tellMarie.println(armPos,BYTE);			// Servo position

	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(RSHOULDERFBPIN,BYTE);	// Servo address
	tellMarie.println(shouldPos,BYTE);	// Servo position

	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(RELBOWRLPIN,BYTE);	// Servo address
	tellMarie.println(elbowPos,BYTE);	// Servo position

	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(TORSOPIN,BYTE);		// Servo address
	tellMarie.println(torsoPos,BYTE);	// Servo position

	/*
	tellMarie.print(255,BYTE);			// Start byte
	tellMarie.print(EYEUDPIN,BYTE);			// Servo address
	tellMarie.println(eyeUDPos,BYTE);	// Servo position
	*/
	Serial.println("Finished Constructor");
}

void marie_curie::turnRight(){
/*	Serial.println((int)turnPos);
	while(turnPos>MAXTURNRIGHT){
		turnPos -= SPEED;
		tellMarie.print(255,BYTE);			// Start byte
		tellMarie.print(TURNPIN,BYTE);			// Servo address
		tellMarie.println(turnPos,BYTE);	// Servo position
```

```
                Serial.println((int)turnPos);
        }
        Serial.println("looking right");*/
}

void marie_curie::turnLeft(){
/*      Serial.println(turnPos);
        while(turnPos<MAXTURNLEFT){
                turnPos += SPEED;
                tellMarie.print(255,BYTE);                      // Start byte
                tellMarie.print(TURNPIN,BYTE);                  // Servo address
                tellMarie.println(turnPos,BYTE);   // Servo position
                Serial.println("Trying to look left");
                Serial.println((int)turnPos);
        }
        Serial.println("looking left");*/
}

void marie_curie::turnForward(){
/*      while(turnPos>TURNFORWARD){
                turnPos -= SPEED;
                tellMarie.print(255,BYTE);                      // Start byte
                tellMarie.print(TURNPIN,BYTE);                  // Servo address
                tellMarie.println(turnPos,BYTE);   // Servo position
                Serial.println("Trying to look center (turning Right)");
        }
        while(turnPos<TURNFORWARD){
                turnPos += SPEED;
                tellMarie.print(255,BYTE);                      // Start byte
                tellMarie.print(TURNPIN,BYTE);                  // Servo address
                tellMarie.println(turnPos,BYTE);   // Servo position
                Serial.println("Trying to look center (turning left)");
        }*/
}

void marie_curie::nodUp(){
/*      while(nodPos>MAXNODUP){
                nodPos -= SPEED;
                tellMarie.print(255,BYTE);                      // Start byte
                tellMarie.print(NODPIN,BYTE);                   // Servo address
                tellMarie.println(nodPos,BYTE);    // Servo position
                Serial.println("Trying to look up");
                Serial.println((int)nodPos);
        }*/
}

void marie_curie::nodDown(){
/*      while(nodPos<MAXNODDOWN){
                nodPos += SPEED;
                tellMarie.print(255,BYTE);                      // Start byte
                tellMarie.print(NODPIN,BYTE);                   // Servo address
                tellMarie.println(nodPos,BYTE);    // Servo position
                Serial.println("Trying to look down");
                Serial.println((int)nodPos);
        }*/

}
```

```
void marie_curie::nodAhead(){
/*     while(nodPos<NODAHEAD){
              nodPos += SPEED;
              tellMarie.print(255,BYTE);                    // Start byte
              tellMarie.print(NODPIN,BYTE);                 // Servo address
              tellMarie.println(nodPos,BYTE);    // Servo position
              Serial.println("Trying to look ahead");
              Serial.println((int)nodPos);
       }
       while(nodPos>NODAHEAD){
              nodPos -= SPEED;
              tellMarie.print(255,BYTE);                    // Start byte
              tellMarie.print(NODPIN,BYTE);                 // Servo address
              tellMarie.println(nodPos,BYTE);    // Servo position
              Serial.println("Trying to look ahead");
              Serial.println((int)nodPos);
       }
       Serial.println("in lookAhead");*/
}

void marie_curie::tiltRight(){
/*     while(tiltPos<MAXTILTRIGHT){
              tiltPos += SPEED;
              tellMarie.print(255,BYTE);                    // Start byte
              tellMarie.print(TILTPIN,BYTE);                // Servo address
              tellMarie.println(tiltPos,BYTE);   // Servo position
              Serial.println("Trying to tilt right");
              Serial.println((int)tiltPos);
       }*/
}

void marie_curie::tiltLeft(){
/*     while(tiltPos>MAXTILTLEFT){
              tiltPos -= SPEED;
              tellMarie.print(255,BYTE);                    // Start byte
              tellMarie.print(TILTPIN,BYTE);                // Servo address
              tellMarie.println(tiltPos,BYTE);   // Servo position
              Serial.println("Trying to tilt left");
              Serial.println((int)tiltPos);
       }*/

}

void marie_curie::tiltLevel(){
/*     while(tiltPos<TILTLEVEL){
              tiltPos += SPEED;
              tellMarie.print(255,BYTE);                    // Start byte
              tellMarie.print(TILTPIN,BYTE);                // Servo address
              tellMarie.println(tiltPos,BYTE);   // Servo position
              Serial.println("Trying to level my head");
              Serial.println((int)tiltPos);
       }
       while(tiltPos>TILTLEVEL){
              tiltPos -= SPEED;
              tellMarie.print(255,BYTE);                    // Start byte
              tellMarie.print(TILTPIN,BYTE);                // Servo address
              tellMarie.println(tiltPos,BYTE);   // Servo position
              Serial.println("Trying to level my head");
```

```
            Serial.println((int)tiltPos);
      }

      Serial.println("in tiltLevel");*/
}

void marie_curie::openMouth(){
      //tellMarie.print(255,BYTE);                        // Start byte
      //tellMarie.print(JAWPIN,BYTE);                 // Servo address
      //tellMarie.println(MOUTHOPEN,BYTE);            // Servo position

}

void marie_curie::closeMouth(){
      //tellMarie.print(255,BYTE);                        // Start byte
      //tellMarie.print(JAWPIN,BYTE);                 // Servo address
      //tellMarie.println(MOUTHCLOSE,BYTE);           // Servo position
}

void marie_curie::saccadeRight(){
      //tellMarie.print(255,BYTE);                        // Start byte
      //tellMarie.print(EYELRPIN,BYTE);               // Servo address
      //tellMarie.println(MAXEYERIGHT,BYTE);    // Servo position
}

void marie_curie::saccadeLeft(){
      //tellMarie.print(255,BYTE);                        // Start byte
      //tellMarie.print(EYELRPIN,BYTE);               // Servo address
      //tellMarie.println(MAXEYELEFT,BYTE);           // Servo position
}

void marie_curie::saccadeForward(){
      //tellMarie.print(255,BYTE);                        // Start byte
      //tellMarie.print(EYELRPIN,BYTE);               // Servo address
      //tellMarie.println(EYEFORWARD,BYTE);           // Servo position
}

void marie_curie::saccadeUp(){
      //tellMarie.print(255,BYTE);                        // Start byte
      //tellMarie.print(EYEUDPIN,BYTE);               // Servo address
      //tellMarie.println(MAXEYEUP,BYTE);             // Servo position
}

void marie_curie::saccadeDown(){
      //tellMarie.print(255,BYTE);                        // Start byte
      //tellMarie.print(EYEUDPIN,BYTE);               // Servo address
      //tellMarie.println(MAXEYEDOWN,BYTE);           // Servo position
}

void marie_curie::saccadeAhead(){
      //tellMarie.print(255,BYTE);                        // Start byte
      //tellMarie.print(EYEUDPIN,BYTE);               // Servo address
      //tellMarie.println(EYEAHEAD,BYTE);             // Servo position
}

void marie_curie::turnto(byte pos){
      //pos = constrain(pos,MAXTURNRIGHT,MAXTURNLEFT);
      //while(turnPos>pos){
```

```
        //      turnPos -= SPEED;
        //      tellMarie.print(255,BYTE);                      // Start byte
        //      tellMarie.print(TURNPIN,BYTE);                  // Servo address
        //      tellMarie.println(turnPos,BYTE);   // Servo position
        //      Serial.println("Turning Right");
        //}
        //while(turnPos<pos){
        //      turnPos += SPEED;
        //      tellMarie.print(255,BYTE);                      // Start byte
        //      tellMarie.print(TURNPIN,BYTE);                  // Servo address
        //      tellMarie.println(turnPos,BYTE);   // Servo position
        //      Serial.println("Turning left");
        //}
}

void marie_curie::nodto(byte pos){
        //pos = constrain(pos,MAXNODUP,MAXNODDOWN);
        //while(nodPos<pos){
        //      nodPos += SPEED;
        //      tellMarie.print(255,BYTE);                      // Start byte
        //      tellMarie.print(NODPIN,BYTE);                   // Servo address
        //      tellMarie.println(nodPos,BYTE);    // Servo position
        //      Serial.println("Trying to nod toward");
        //      Serial.println((int)nodPos);
        //}
        //while(nodPos>pos){
        //      nodPos -= SPEED;
        //      tellMarie.print(255,BYTE);                      // Start byte
        //      tellMarie.print(NODPIN,BYTE);                   // Servo address
        //      tellMarie.println(nodPos,BYTE);    // Servo position
        //      Serial.println("Trying to nod toward");
        //      Serial.println((int)nodPos);
        //}
        //
}

void marie_curie::tiltto(byte pos){
        //pos = constrain(pos,MAXTILTRIGHT,MAXTILTLEFT);
        //while(tiltPos<pos){
        //      tiltPos += SPEED;
        //      tellMarie.print(255,BYTE);                      // Start byte
        //      tellMarie.print(TILTPIN,BYTE);                  // Servo address
        //      tellMarie.println(tiltPos,BYTE);          // Servo position
        //      Serial.println("Trying to tilt my head");
        //      Serial.println((int)tiltPos);
        //}
        //while(tiltPos>pos){
        //      tiltPos -= SPEED;
        //      tellMarie.print(255,BYTE);                      // Start byte
        //      tellMarie.print(TILTPIN,BYTE);                  // Servo address
        //      tellMarie.println(tiltPos,BYTE);          // Servo position
        //      Serial.println("Trying to tilt my head");
        //      Serial.println((int)tiltPos);
        //}
        //
}

void marie_curie::saccadeUDto(byte pos){
```

```
	/*pos = constrain(pos,MAXEYEUP,MAXEYEDOWN);
	tellMarie.print(255,BYTE);
	tellMarie.print(EYEUDPIN,BYTE);
	tellMarie.println(pos,BYTE);*/
}

void marie_curie::saccadeLRto(byte pos){
	/*pos = constrain(pos,MAXEYELEFT,MAXEYERIGHT);
	tellMarie.print(255,BYTE);
	tellMarie.print(EYELRPIN,BYTE);
	tellMarie.println(pos,BYTE);*/
}

void marie_curie::turnEyesRight(){
	//tellMarie.print(255,BYTE);                        // Start byte
	//tellMarie.print(EYELRPIN,BYTE);               // Servo address
	//tellMarie.println(MAXEYERIGHT,BYTE);    // Servo position
}

void marie_curie::turnEyesLeft(){
	//tellMarie.print(255,BYTE);                        // Start byte
	//tellMarie.print(EYELRPIN,BYTE);               // Servo address
	//tellMarie.println(MAXEYELEFT,BYTE);           // Servo position
}

void marie_curie::rightArmUp(){
	armPos = MAXARMUP;//convert(MAXARMUP);
	tellMarie.print(255,BYTE);                        // Start byte
	tellMarie.print(RARMUDPIN,BYTE);          // Servo address
	tellMarie.println(armPos,BYTE);           // Servo position
}

void marie_curie::rightArmDown(){
	armPos = MAXARMDOWN;//convert(MAXARMDOWN);
	tellMarie.print(255,BYTE);                        // Start byte
	tellMarie.print(RARMUDPIN,BYTE);          // Servo address
	tellMarie.println(armPos,BYTE);           // Servo position
}

void marie_curie::shoulderTo( byte target){
	shouldPos = target;//shouldPos = convert(target);
	tellMarie.print(255,BYTE);                              // Start byte
	tellMarie.print(RSHOULDERFBPIN,BYTE);           // Servo address
	tellMarie.println(shouldPos,BYTE);                      // Servo position
}

void marie_curie::elbowTo(byte target){
	elbowPos = target;//elbowPos = convert(target);
	tellMarie.print(255,BYTE);                              // Start byte
	tellMarie.print(RELBOWRLPIN,BYTE);              // Servo address
	tellMarie.println(elbowPos,BYTE);                       // Servo position
}

void marie_curie::torsoTo(byte target){
	torsoPos = target;//torsoPos = convert(target);
	tellMarie.print(255,BYTE);                              // Start byte
	tellMarie.print(TORSOPIN,BYTE);                 // Servo address
	tellMarie.println(torsoPos,BYTE);                       // Servo position
```

```
}

void marie_curie::playNote(char note){
	byte shoulder_target;
	byte elbow_target;
	byte torso_target;

	switch (note)
	{
	case 'C':
		shoulder_target = 65;//1464;
		elbow_target  = 130;//1483;
		torso_target  = 143;//1622;
		break;
	case 'D':
		shoulder_target = 65;
		elbow_target  = 120;
		torso_target  = 85;
		break;
	case 'G':
		shoulder_target = 45;
		elbow_target  = 100;
		torso_target  = 140;
		break;
	case 'E':
		shoulder_target = 70;
		elbow_target  = 128;
		torso_target  = 85;
		break;
	case 'A':
		shoulder_target = 60;
		elbow_target  = 100;
		torso_target  = 143;
		break;
	case 'F':
		shoulder_target = 70;
		elbow_target  = 145;
		torso_target  = 85;
		break;

	case 'B':
		shoulder_target = 60;
		elbow_target  = 108;
		torso_target  = 143;
		break;

	case 'S': // for silent
		rightArmUp();
		shoulder_target = 65;
		elbow_target  = 101;
		torso_target  = 90;
		break;
	}
	shoulderTo(shoulder_target);
	elbowTo(elbow_target);
	torsoTo(torso_target);
}
```

```
void marie_curie::composeMusic(){
	bool adjacency [8][9] =
	{
		{1,1,1,1,1,1,1,1,1},
		{0,1,1,1,0,0,0,0,0},
		{0,0,1,1,1,0,0,0,1},
		{0,0,0,1,1,1,0,0,1},
		{0,0,0,0,1,1,1,0,1},
		{1,0,0,0,0,1,1,1,1},
		{0,0,0,0,0,0,1,1,1},
		{1,0,0,0,0,0,0,1,1}
	};

	playNote('C');
	int index = 0;
	int temp_index = 0;

	for (int i=0; i<10; i++)
	{
		rightArmUp();
		// generate random number between 0 and 8
		temp_index = rand() % 9;
		if(adjacency[index][temp_index] == 1)
		{
			switch(temp_index)
			{
			case 0: // C
				playNote('C');
				rightArmDown();
				delay(1000);
				index = 0;
				break;
			case 1: // D after a C
				playNote('D');
				rightArmDown();
				delay(1000);
				index = 1;
				break;
			case 2: // G after only a D
				playNote('G');
				rightArmDown();
				delay(1000);
				index = 2;
				break;
			case 3: // E
				playNote('E');
				rightArmDown();
				delay(1000);
				index = 3;
				break;
			case 4: // A
				playNote('A');
				rightArmDown();
				delay(1000);
				index = 4;
				break;
			case 5: // F
				playNote('F');
```

```
                        rightArmDown();
                        delay(1000);
                        index = 5;
                        break;
                case 6: // D after A or F
                        playNote('D');
                        rightArmDown();
                        delay(1000);
                        index = 6;
                        break;
                case 7: // G after D or F
                        playNote('G');
                        rightArmDown();
                        delay(1000);
                        index = 7;
                        break;
                case 8: // Silent note. Stay on current index.
                        playNote('S');
                        delay(1000);
                        break;
                default:
                        break;
                }
        }
    }
}
```

## Arduino code

Important: Use Arduino 0023 to be able to compile and run the code.

```
#include <Marie_Curie.h>

// ********************************************************
// Marie Curie
// ECE 478/578
// Fall 2011 and 2013
//
// Arduino code to control Marie
// ********************************************************
#include <SoftwareSerial.h>
#include <marie_curie.h>

#define RESET 5

marie_curie Marie;
int posit = 0;
```

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Starting Setup");

  // Pololu board must be reset at start up
  pinMode(RESET,OUTPUT);
  digitalWrite(RESET, LOW); // Pull Reset down
  delay(100);
  pinMode(RESET,INPUT); // Let reset go
//Serial.begin(9600);
}

// Can call the setServo function and pass it whatever servo number and degree here.
// This will be our main program.
void loop()
{
  Serial.println("looping");
    Marie.playNote('C');
    delay (2000);

    Marie.rightArmUp();
    delay (2000);

    Marie.rightArmDown();
    delay (1000);

    Marie.rightArmUp();
    delay (2000);

    Marie.playNote('B');
    delay (2000);

    Marie.rightArmDown();
    delay (1000);

    Marie.rightArmUp();
    delay (2000);

    Marie.playNote('E');
    delay (2000);
```

```
Marie.rightArmDown();
delay (1000);

Marie.rightArmUp();
delay (2000);

Marie.playNote('S');
delay (2000);

Marie.playNote('D');
delay (2000);

Marie.rightArmDown();
delay (1000);

Marie.rightArmUp();
delay (2000);

Marie.playNote('A');
delay (2000);

Marie.rightArmDown();
delay (1000);

Marie.rightArmUp();
delay (2000);

Marie.playNote('G');
delay (2000);

Marie.rightArmDown();
delay (1000);

Marie.rightArmUp();
delay (2000);

Marie.playNote('F');
delay (2000);

Marie.rightArmDown();
delay (1000);
```

```
    Marie.rightArmUp();
    delay (2000);
    /*
    Marie.composeMusic();
    delay (2000);
   */
}
```

## IGA Code (From Bohr Robot)

### *Algorithm.h*

```
//#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <cstring>
//#include <string>
#include <sstream>
#include <time.h>
#include <fcntl.h>


#include "maestro.h"

using namespace std;

//#include "rs232.h"

#define NUM_GENES    9
#define NUM_GTYPES   8

//Parameters for serial communication
#define SBYTE 255 // Sync byte for MiniSSC, this never changes
//COM port number - 16 corresponds to USBtty0 in Linux on my laptop.
//This number will need to be changed if this code is compiled in windows, because a USB
to RS232 adapter is given a
//regular COM port designation.  Likewise, if a regular COM port is used, the appropriate
number needs to be entered above.
#define COMPORT "COM4"
#define BRATE 9600 // Baud rate for MiniSSC, this is only changed via a jumper pin on the
MiniSSC II chip, leave it alone.

struct ranges {
int Low;
int High;
int Abs;
int Def;
};

//Note that the class member have been shuffled and grouped somewhat, and may not appear
in Algorithm.cpp in the order listed here
class GA_Ops
{
```

```
public:
	GA_Ops(); //Constructor, no arguments.
	~GA_Ops(); //Destructor

	//Basic genetic algorithm functions
	int Init_Generation(); //Initialize a new generation with random numbers
	int Randomize(int a); //Randomize one genotype
	int Crossover(int a, int b); //Crossover two genotypes - replaces each parent with
a child
	int Mutate(int a); // Mutate one genotype

	//Fitness functions
	int Feedback(int a);  //Enter feedback for one genotype
	int Calc_Fitness(int a); //Calculate the fitness of one genotype from its feedback
	int Find_Elites(); //Find the elites for the current generation
	int Set_Expression_Name(char expr_name[]); //Sets the name of the currently
evaluated expression

	//In-program value format
	int Write_Generation(char filename[]); //Write the current generation out to a
file
	int Read_Generation(char filename[]); //Read in a generation from a file

	//Raw servo values
	int Write_Gen_Convert(char filename[]); //Convert the current generation to servo
values and write out
	int Write_Gtype_Convert(int a, char filename[]); //Convert a genotype and write
out to file

	//Serial communication with the robot
	int Set_Servo_Ranges(); //Set the range of motion for each servo
	int Write_Servo_Ranges(char filename[]); //Write the ranges of motion out to a
file
	int Read_Servo_Ranges(char filename[]); //Read the ranges of motion from a file
	int Send_Gtype(int a); //Convert a genotype and send it to the robot
	int Display_Elites(); //Convert and send the values of the elite genotypes to the
robot
	int Reset_Servos(); //Set all servos to default values


	//Random expression generation
	int Find_Exp_Ranges(int Fitness);//Find the range of values per gene for a given
fitness level
	int Gen_Expression(); //Generate a random expression from the ranges and send to
the robot

	//Print functions, prints to console
	void Print_All(); //Print the current generation in terminal
	void Print_Gtype(int a); //Print one genotype in terminal
	void Print_Elites(); //Print the elites of the current generation, if any
	void Print_Feedback(); //Print the feedback for the current generation, if any
	void Print_Exp_Ranges(); //Print the found ranges for random expression, if any
	void Print_Servo_Ranges(); //Print the servo ranges of motion, if any

	int Get_Num_Gtypes(); //Returns the NUM_GTYPES parameter
	int Get_Num_Genes(); //Returns the NUM_GENES parameter
private:
	bool Is_Elite(int a); //Returns true if the genotype is an elite
```

```
	unsigned char ** Generation; //The current generation
	unsigned char ** Feedback_Array; //Feedback for the current generation
	int Gen_ID; //The ID of the current generation (starts at 0)
	int * Elites; //The elites of the current generation
	ranges * Servo_Range; //The array holding minimum, maximum, default and absolute
ranges for each servo
	ranges * Expr_Range; //The array holding minimum, maximum, and absolute ranges for
the random expression generator
	char * Expression; //The name of the expression currently being evaluated

	//Genetic Algorithm Parameters
	//int NUM_GENES; //Number of genes per genotype
	//int NUM_GTYPES; //Number of genotypes per generation
	int NUM_MUTATE; //Maximum number of genes per genotype that can be mutated
	int NUM_CROSSOVER; //Maximum number of genes per genotype that can be crossed over
	int MUT_PERCENT; //The percentage of the maximum value of a gene (254) that a
mutation can change
	int FITNESS_BYTES; //The number of bytes in a genotype used to store the fitness
value of that genotype
	int FEEDBACK_ARRAY_BYTES;  //The number of possible feedback entries per genotype
in the feedback array
	int NUM_ELITES; //The number of elites per generation that are carried over
};
```

## *Algorithm.cpp*

```
#include "Algorithm.h"

GA_Ops::GA_Ops()
{
	srand((unsigned) time(0)); //Seeds the random number generator with the current
RTC time
	Gen_ID = 0; //Starts the generation ID at 0
	Generation = NULL; //Sets the 3 array pointers to NULL until they are initialized
	Feedback_Array = NULL;
	Elites = NULL;
	//The following are pointers for the arrays of range structs that hold the ranges
for the servos of the robots,
	//and the servo ranges of motion for the random expresion generator.
	Servo_Range = NULL;
	Expr_Range = NULL;

	//Set parameters below
	//This could be copied to a second constructor, with an array as an argument to
pass in each of these variables.

	//NUM_GENES = 9; //Number of genes per genotype
	//NUM_GTYPES = 4; //21; //Number of genotypes per generation
	NUM_MUTATE = 6; //Maximum number of genes per genotype that can be mutated
	NUM_CROSSOVER = 5; //Maximum number of genes per genotype that can be crossed over
	MUT_PERCENT = 20; //The percentage of the maximum value of a gene (255) that a
mutation can change
	FITNESS_BYTES = 1; //The number of bytes in a genotype used to store the fitness
value of that genotype
```

```
	FEEDBACK_ARRAY_BYTES = 5; //The number of possible feedback entries per genotype
in the feedback array
	NUM_ELITES = 2; //The number of elites per generation that are carried over
}

GA_Ops::~GA_Ops()
{
	if(Generation) //If the generation pointer has been initialized, delete both array
dimensions
	{
		for (int i = 0; i<NUM_GTYPES; i++)
		{
			delete [] Generation[i];
		}
		delete [] Generation;
	}

	if (Feedback_Array) //If the feedback pointer has been initialized, delete both
array dimensions
	{
		for (int i = 0; i<NUM_GTYPES; i++)
		{
			delete [] Feedback_Array[i];
		}
		delete [] Feedback_Array;
	}

	if (Elites) //If the elites pointer has been initialized, delete the array
	{
		delete [] Elites;
	}

	if (Servo_Range) //Same for the servo ranges
	{
		delete [] Servo_Range;
	}

	if (Expr_Range) //And the expression generator ranges
	{
		delete [] Servo_Range;
	}

}

int GA_Ops::Init_Generation() //Initializes the generation pointer and creates a new,
randomized generation
{				//of NUM_GTYPES genotypes.
	char Response;
	printf("This will randomize all genotypes in this generation. \n");
	printf("Do you wish to continue? (Y/N) \n");
	cin>>Response;
	cin.ignore();
	if(Response == 'N' || Response == 'n')
	{
		printf("Cancelling Init_Generation. \n");
		return 0;
	}
	else if(Response != 'Y' && Response != 'y')
```

```
	{
		printf("Invalid response. \n");
		return 0;
	}

	printf("Randomizing generation... \n");

	Generation = new unsigned char*[NUM_GTYPES]; //Initializes Generation as a matrix
of height NUM_GTYPES
	for (int i = 0; i<NUM_GTYPES; i++)	//and width NUM_GENES + FITNESS_BYTES
	{
		Generation[i] = new unsigned char[NUM_GENES+FITNESS_BYTES];
	}

	for (int i = 0; i<NUM_GTYPES; i++)
	{
		for (int j = 0; j<NUM_GENES; j++)
		{
			Generation[i][j] = (rand() % 255);
		}
		Generation[i][NUM_GENES] = 0;//Sets the fitness byte to 0
	}

	Feedback_Array = new unsigned char*[NUM_GTYPES]; //Creates a second, parallel two
dimensionall array
	for (int i = 0; i<NUM_GTYPES; i++)	//that holds the feedback for each genotype
	{
		Feedback_Array[i] = new unsigned char[FEEDBACK_ARRAY_BYTES];//The array
height is NUM_GTYPES,
	}							//with width
FEEDBACK_ARRAY_BYTES

	for (int i = 0; i<NUM_GTYPES; i++)
	{
		for (int j = 0; j<FEEDBACK_ARRAY_BYTES; j++)
		{
			Feedback_Array[i][j] = 0;
		}
	}
	return 0;
	Gen_ID = 0;
}

int GA_Ops::Randomize(int Gtype) //Randomizes a single genotype in the generation.  Used
to randomize any genotypes
{				//that are not either selected as elites or crossed over.
	printf("\n");
	printf("Randomizing Genotype %d. \n", Gtype);
	for (int i = 0; i<NUM_GENES; i++)
	{
		Generation[Gtype][i] = (rand() % 255);
	}
	Generation[Gtype][NUM_GENES] = 0;//Sets the fitness byte to 0
	return 0;
}

int GA_Ops::Crossover(int Gtype1, int Gtype2) //Swaps a random number of random genes
between the two parent genotypes
```

```
{                                       //GtypeA and GtypeB, replacing each parent with
the resulting child.
	if (Generation == NULL)
	{
		printf("Error! Generation has not been initialized, cannot crossover!\n");
		return -1;
	}
	else if(Gtype1 <0 || Gtype1 > (NUM_GTYPES-1) || Gtype2 <0 || Gtype2 > (NUM_GTYPES-
1))
	{
		printf("Error! One or more Selected Genotypes are outside valid range.\n");
		return -1;

	}
	else
	{
		int Num_Selected = 0;//The number of selected genes starts at 0
		unsigned char Temp;
		int Roll; //The random number generated by the selector.
		int Num_Crossed_Genes; //Number of genes that will actual be crossed over,
random between 0 and NUM_CROSSOVER-1.
		int Selected_Genes[NUM_GENES]; //Array of the same length as a genotype,
holds the flag for each
                                        //gene showing whether it has been selected.
		if(NUM_CROSSOVER > NUM_GENES)
		{
			printf("Error! The number of genes to be crossed over is greater
than the number of genes.\n");
			return -1;
		}
		Num_Crossed_Genes = rand() % NUM_CROSSOVER;//Randomly determine the number
of genes to crossover
		while (Num_Selected < Num_Crossed_Genes)//Iterate the loop until the number
of selected genes is
		{                                       //equal to the generated number.
                                        //If Num_Crossed_Genes came out to 0,
loop will not happen.
			Roll = rand() % NUM_GENES;//Random number between 0 and NUM_GENES -1
is generated
			if(Selected_Genes[Roll] != 1)//If the corresponding gene has not
been selected
			{
				Selected_Genes[Roll] = 1;//Then select it
				Num_Selected++;//and increment the Num_Selected counter.
			}
		}

		printf("\n");

		printf("Selected genes for crossover: ");

		for (int i = 0; i<NUM_GENES; i++)
		{
			if (Selected_Genes[i] == 1)
			{
				printf("%d", i);
				if (i != (Num_Selected-1))//Inserts a comma after all but the
last number
```

```
                        {
                                printf(", ");
                        }
                }
        }

        printf("\n");
        printf("Performing crossover... \n");

        for (int i = 0; i<NUM_GENES; i++)
        {
                if (Selected_Genes[i] == 1) //If this gene has been selected for
crossover,
                {
                        Temp = Generation[Gtype1][i]; //Store the gene from parent1
into the temp char
                        Generation[Gtype1][i] = Generation[Gtype2][i]; //Swap the gene
from parent2 to parent1
                        Generation[Gtype2][i] = Temp; //and swap the gene from the
temp char to parent2
                }
        }
}

        return 0;
}

int GA_Ops::Mutate(int Gtype) //Mutates a random number of random genes be a random
amount, within the percentage of
{                       //the max value (255) specified in MUT_PERCENT in the header.
        if (Generation == NULL)
        {
                printf("Error! Generation has not been initialized, cannot mutate!\n");
                return -1;
        }
        else if(Gtype <0 || Gtype > (NUM_GTYPES-1))
        {
                printf("Error! Selected Genotype is outside valid range.\n");
                return -1;

        }
        else
        {

                float Temp;
                float Random;
                float Mutation;
                int Sign;
                int Roll;
                int Selected_Genes[NUM_GENES];
                int Num_Selected = 0;
                int Num_Mutate_Genes = 0; //Randomly determined number of genes to mutate.

                if(NUM_MUTATE > NUM_GENES)
                {
                        printf("Error! The number of genes to be mutated is greater than the
number of genes.\n");
                        return -1;
```

```
		}
		Num_Mutate_Genes = rand() % NUM_MUTATE;//Randomly determine the number of
genes to mutate
		while (Num_Selected < Num_Mutate_Genes)//Iterate the loop until the number
of selected genes is
		{					//equal to the generated number.
					//If Num_Mutate_Genes came out to 0, loop
will not happen.
			Roll = rand() % NUM_GENES;//Random number between 0 and NUM_GENES -1
is generated
			if(Selected_Genes[Roll] != 1)//If the corresponding gene has not
been selected
			{
				Selected_Genes[Roll] = 1;//Then select it
				Num_Selected++;//and increment the Num_Selected counter.
			}
		}

		printf("\nSelected genes for mutation: "); //Selected genes are printed

		for (int i = 0; i<NUM_GENES; i++)
		{
			if (Selected_Genes[i] == 1)
			{
				printf("%d", i);
				if (i != (NUM_MUTATE-1)) //Inserts a comma after each number
but the last
				{
					printf(", ");
				}
			}
		}

		printf("\nMutating genes... \n");

		for (int i = 0; i<NUM_GENES; i++)//Iterates for each gene in the genotype
		{
			if (Selected_Genes[i] == 1) //If this gene has been selected for
mutation,
			{
				Random = (rand() % MUT_PERCENT) +1; //A number is generated
between 1 and MUT_PERCENT
				Mutation = (float)(Random/100)*255.0; //That number as a
percentage is multiplied by the
				Sign = (rand() % 2);		//max value, 255, to get the
mutation amount.  Depending on
				if (Sign == 1)				//whether the random
number Sign is 0 or 1, the mutation
				{				//amount is either added or
subtracted from the original value.
					Mutation = (Mutation*(-1));
				}
				printf("Mutating gene %d by %d percent. \n", i, (int)Random);
				printf("Mutation = %f \n", Mutation);
				Temp = ((float)Generation[Gtype][i] + Mutation);
				if (Temp > 255.0) //If the resulting mutated value is either
above 255 or below 0, it defaults to the maximum
				{		//or minimum, respectively.
```

```
                                Temp = 255.0;
                        }
                        else if (Temp < 0.0)
                        {
                                Temp = 0.0;
                        }

                        printf("Mutated from %d to %d. \n", Generation[Gtype][i],
(int)Temp);//Prints the old and new values
                        Generation[Gtype][i] = (int)Temp;//Sets the gene in the
genotype array to the new value
                }
        }
    }
    return 0;
}

int GA_Ops::Feedback(int Gtype) //Solicits feedback on a single genotype.  The genotype
is converted to servo values
                        //and written out to a file, and read by the servo controller.
When the user has
                        //evaluated the expression, the feedback is stored in the
genotype's data in the
                        //current generation.
{
    if(Gtype <0 || Gtype > (NUM_GTYPES-1))
    {
        printf("Error! Selected Genotype is outside valid range.\n");
        return -1;

    }
    if (Feedback_Array == NULL) //Feedback array has not been initialized, probably
because the current generation
    {                       //was loaded from a file.  The array is initialized with
zeroes is all fields.
        Feedback_Array = new unsigned char*[NUM_GTYPES]; //Creates a second,
parallel two dimensionall array
        for (int i = 0; i<NUM_GTYPES; i++)    //that holds the feedback for each
genotype
        {
                Feedback_Array[i] = new unsigned char[FEEDBACK_ARRAY_BYTES];//The
array height is NUM_GTYPES,
        }                                                       //with width
FEEDBACK_ARRAY_BYTES
        for (int i = 0; i<NUM_GTYPES; i++) //Initializes the array with all zeroes.
        {
                for (int j = 0; j<FEEDBACK_ARRAY_BYTES; j++)
                {
                        Feedback_Array[i][j] = 0;
                }
        }
    }
    else if (Feedback_Array[Gtype][FEEDBACK_ARRAY_BYTES-1] != 0)//If the row for the
selected genotype is already full
    {
        printf("Error! Feedback array for this genotype is full, cannot enter new
feedback for this genotype.\n");
        return -1;
```

```
	}

	int Response = -1;
	printf("Please rate the current robot expression with an integer on the scale of
one to ten.\n");
	printf("1----------------------------------5----------------------------------
10\n");
	printf("Inappropriate                                                    Perfect
Example\n");
	printf("Expression Being rated: %s", Expression);
	do{
		cin>>Response;
		cin.ignore(5, '\n');
		if (Response <0 || Response >10)
		{
			printf("Invalid response.  Please enter an integer between 1 and
10.\n");
			Response = -1;
		}
	}while(Response <0 || Response >10);

	char getbyte = 0;    //This routine finds the first unwritten feedback byte for the
selected
	int array_position = 0; //genotype.
	getbyte = Feedback_Array[Gtype][array_position];
	while (getbyte != 0 && array_position < FEEDBACK_ARRAY_BYTES) //This loop
increments the array position
	{                                                          //until an empty byte is
found.
		array_position++;
		getbyte = Feedback_Array[Gtype][array_position];
	}
	Feedback_Array[Gtype][array_position] = Response;
	return 0;
}

int GA_Ops::Calc_Fitness(int Gtype)
{
	if (Feedback_Array == NULL)
	{
		printf("Error! Feedback array has not been initialized, cannot calculate
fitness!\n");
		return -1;
	}
	else if (Generation == NULL)
	{
		printf("Error! Generation has not been initialized, cannot calculate
fitness!\n");
		return -1;
	}
	else if(Gtype <0 || Gtype > (NUM_GTYPES-1))
	{
		printf("Error! Selected Genotype is outside valid range.\n");
		return -1;

	}
	else {
```

```
		int total = 0;
		int num_fb = 0;
		float mean;

		for (int i = 0; i < FEEDBACK_ARRAY_BYTES; i++)//This routine calculates the
mean of the feedback that has been entered
		{                                                    //for this Genotype.  Although all
of the numbers in the feedback array
                                                             //are added to the total, only the
non-zero entries in the feedback array
                                                             //are calculated as part of the
mean.  For example, if the feedback array
                                                             //held {5, 10, 7, 0, 0} for a
genotype, the mean would be calculated as
                                                             //(5 + 10 + 7 + 0 + 0) / 3.
			total = (total + Feedback_Array[Gtype][i]);
			if (Feedback_Array[Gtype][i] != 0)
			{
				num_fb++;
			}
		}
		printf("Gtype = %d, total = %d, num_fb = %d \n",Gtype, total, num_fb);
		if(total == 0 || num_fb == 0)
		{
			Generation[Gtype][NUM_GENES] = 0;
		}
		else
		{
			mean = (total/num_fb);
			Generation[Gtype][NUM_GENES] = (int)mean;
		}
	}

	return 0;
}

bool GA_Ops::Is_Elite(int Gtype) //Compares the genotype identified by Gtype against the
contents of the
{                            //array keeping track of the elites for this generation.
	if (Elites == NULL)
	{
		return false;
	}
	for (int i = 0; i < NUM_ELITES; i++)
	{
		if (Elites[i] == Gtype)
		{
			return true;
		}
	}
	return false;
}

int GA_Ops::Find_Elites()
{
	if (Generation == NULL)
	{
```

```
		printf("Error! Generation has not been initialized, cannot calculate
elites!\n");
		return -1;
	}
	else
	{
		Elites = new int[NUM_ELITES]; //Initialize elites array and set initial
elites.
		for (int i = 0; i <NUM_ELITES; i++)
		{
			Elites[i] = i;
		}

		for (int i = 0; i < NUM_ELITES; i++) //Finds the most fit genotypes in
descending order and sets them as elite.
		{
			for (int j = 0; j <NUM_GTYPES; j++)
			{
				if (Is_Elite(j) == false &&
Feedback_Array[j][FEEDBACK_ARRAY_BYTES] >
Feedback_Array[Elites[i]][FEEDBACK_ARRAY_BYTES])
				{
					Elites[i] = j;
				}
			}
		}
	}

	return 0;
}

int GA_Ops::Set_Expression_Name(char * expr_name)
{
	if (Expression == NULL)//If expression has not been initialized, then allocate it
with a length of expr_name.
	{
		Expression = new char[strlen(expr_name+1)];
	}
	else //Otherwise, delete the current Expression array and create a new one.
	{
		delete [] Expression;
		Expression = new char[strlen(expr_name+1)];
	}
	strcpy(Expression, expr_name);//Copy the contents of expr_name to Expression.
	return 1;
}
int GA_Ops::Write_Generation(char filename[])  //Writes the current generation out to a
file, which can
					//be loaded back in later.
{
	if (Generation == NULL)
	{
		printf("Error! Generation has not been initialized, cannot write to
file!\n");
		return -1;
	}
	else
	{
```

```
		ofstream Outfile;
		Outfile.open(filename);
		if (!Outfile.is_open())
		{
			printf("Error opening file to write! \n");
			return -1;
		}
		Outfile <<"Generation ";
		Outfile <<Gen_ID;
		Outfile <<"\n";

		for (int i = 0; i<NUM_GTYPES; i++)
		{
			Outfile <<i;
			Outfile <<" ";
			for (int j = 0; j<(NUM_GENES+FITNESS_BYTES); j++)
			{
				Outfile <<(int)Generation[i][j];
				Outfile <<" ";
			}
			Outfile <<"\n";
		}

		Outfile.close();
		return 0;
	}
}

int GA_Ops::Write_Gen_Convert(char filename[])  //Converts the entire current generation
to servo position values and writes it out
					//to the beginning of a file.
{
	float Temp;
	if (Generation == NULL)
	{
		printf("Error! Generation has not been initialized, cannot write to
file!\n");
		return -1;
	}
	else
	{
		ofstream Outfile;
		Outfile.open(filename);
		if (!Outfile.is_open())
		{
			printf("Error opening file to write! \n");
			return -1;
		}
		Outfile <<"Generation ";
		Outfile <<Gen_ID;
		Outfile <<"\n";

		for (int i = 0; i<NUM_GTYPES; i++)//Outputs each genotype in turn
		{
			Outfile <<i;
			Outfile <<" ";
			for (int j = 0; j<NUM_GENES; j++)//Applies the conversion formula to
each raw value and writes it out to file.
```

```
                        {
                                Temp = ((float)Generation[i][j] * (Servo_Range[j].Abs/255.0))
+ Servo_Range[j].Low;
                                Outfile <<(int)Temp;
                                Outfile <<" ";
                        }
                        Outfile <<"\n";
                }
                Outfile.close();
                return 0;
        }
        return 0;
}

int GA_Ops::Write_Gtype_Convert(int Gtype, char filename[])  //Converts one genotype to
servo position values and writes it out
                                    //to the beginning of a file.
{
        float Temp;
        if (Generation == NULL)
        {
                printf("Error! Generation has not been initialized, cannot write to
file!\n");
                return -1;
        }
        else if(Gtype <0 || Gtype > (NUM_GTYPES-1))
        {
                printf("Error! Selected Genotype is outside valid range.\n");
                return -1;

        }
        else
        {
                ofstream Outfile;
                Outfile.open(filename);
                if (!Outfile.is_open())
                {
                        printf("Error opening file to write! \n");
                        return -1;
                }
                printf("\nWriting converted genotype %d to '%s'...\n", Gtype, filename);
                Outfile <<Gtype;
                Outfile <<" ";
                for (int i = 0; i<NUM_GENES; i++) //Applies the conversion formula to each
raw value and writes it out to file.
                {
                        Temp = ((float)Generation[Gtype][i] * (Servo_Range[i].Abs/255.0)) +
Servo_Range[i].Low;
                        Outfile <<(int)Temp;
                        Outfile <<" ";
                }
                Outfile <<"\n";
                Outfile.close();
        }
        return 0;
}
```

```
int GA_Ops::Send_Gtype(int Gtype) //Converts and sends one genotype over the serial port
to the robot.
{
	unsigned char buf; //Serial data buffer. buf[0] is the sync byte, buf[1] is the
servo address, and
	unsigned char channel;
	unsigned short target;
	float Temp;		//buf[2] is the servo position byte.
	HANDLE port; // was int ret = 0
	BOOL success;

	if (Generation == NULL)
	{
		printf("Error! Generation has not been initialized, cannot write to
file!\n");
		return -1;
	}
	else if (Gtype <0 || Gtype > (NUM_GTYPES-1))
	{
		printf("Error! Selected Genotype is outside valid range.\n");
		return -1;

	}
	else if (Servo_Range == NULL)
	{
		printf("Error! Servo Ranges have not been set.\n");
		return -1;
	}
	else
	{
		buf = SBYTE; //The sync byte, always 255
		printf("Displaying converted genotype %d", Gtype);
		port = openPort(COMPORT, BRATE); // OpenComport(COMPORT, BRATE); //Attempts
to open the COM port

		if (port == INVALID_HANDLE_VALUE)
		{
			printf("Error opening ", COMPORT);
			return -1;
		}
		for (int i = 0; i<NUM_GENES; i++)//Each gene is sent in turn to the robot
		{
			Temp = ((float)Generation[Gtype][i] * (Servo_Range[i].Abs/255.0)) +
Servo_Range[i].Low; //Conversion
			channel = i; //The address of the servo is set
			target = (int)Temp;//The servo position is set to the converted
value
			//SendBuf(COMPORT, buf, 3);//Sends the buffer to the robot
			success = maestroSetTarget(port, channel, target); // Sends the
position to the target servo
			if (!success){ return -1; }
		}
	}
	CloseHandle(port); //Closes the COM port
	return 0;
}

int GA_Ops::Display_Elites()
```

```
{
	char prompt[2];
	unsigned char buf[3]; //Serial data buffer. buf[0] is the sync byte, buf[1] is the
servo address, and
	float Temp;		//buf[2] is the servo position byte.
	//int ret = 0;
	HANDLE port;
	BOOL success;

	if (Generation == NULL)
	{
		printf("Error! Generation has not been initialized, cannot write to
file!\n");
		return -1;
	}
	else if (Elites == NULL)
	{
		printf("Error! Elites have not been found for this generation.\n");
		return -1;
	}
	else if (Servo_Range == NULL)
	{
		printf("Error! Servo Ranges have not been set.\n");
		return -1;
	}
	else
	{
		buf[0] = SBYTE;//Sync byte is set
		port = openPort(COMPORT, BRATE);//Attempt to open COM port
		if (port == INVALID_HANDLE_VALUE)
		{
			printf("Error opening ", COMPORT);
			return -1;
		}
		for (int i = 0; i < NUM_ELITES; i++) //Loop runs once for each elite
		{
			for (int j = 0; j<NUM_GENES; j++)//Each gene is sent in turn to the
robot
			{
				Temp = ((float)Generation[Elites[i]][j] *
(Servo_Range[j].Abs/255.0)) + Servo_Range[j].Low; //Conversion
				buf[1] = j; //Address
				buf[2] = (int)Temp; //Converted value
				//SendBuf(COMPORT, buf, 3);//Sends the buffer to the robot
				success = maestroSetTarget(port, buf[1], buf[2]); // Sends the
position to the target servo
				if (!success){ return -1; }
			}

			printf("Displaying elite genotype %d... \n", i);

			if (i < NUM_ELITES-1)
			{
				printf("Press enter to display the next elite genotype.\n");
			}
			else
			{
				printf("Press enter to return to the main menu.\n");
```

```
                        }
                        cin.get(prompt,2,'\n'); //When enter is pressed, the loop continues.
                        cin.ignore();
                }
        }
        //CloseComport(COMPORT);//Closes the COM port
        CloseHandle(port); // Closes the COM port
        return 0;
}

int GA_Ops::Reset_Servos()
{
        unsigned char buf[3]; //Serial data buffer
        //int ret = 0;
        HANDLE port;
        BOOL success;

        if (Servo_Range == NULL)
        {
                printf("Error! Servo Ranges have not been set.\n");
                return -1;
        }
        else
        {
                buf[0] = SBYTE;
                printf("\nResetting servo positions...");
                port = openPort(COMPORT, BRATE); //Opens the COM port
                if (port == INVALID_HANDLE_VALUE)
                {
                        printf("Error opening ", COMPORT);
                        return -1;
                }
                for (int i = 0; i<NUM_GENES; i++)//Loops once for each gene/servo
                {
                        buf[1] = i;//Address
                        buf[2] = Servo_Range[i].Def;//Servo position
                        //SendBuf(COMPORT, buf, 3); //Sends the buffer to the robot
                        success = maestroSetTarget(port, buf[1], buf[2]); // Sends the
position to the target servo
                        if (!success){ return -1; }
                }
        }
        CloseHandle(port);
        return 0;
}

int GA_Ops::Read_Generation(char filename[])
{
        if (Generation == NULL)
        {
                Generation = new unsigned char*[NUM_GTYPES]; //Initializes Generation as a
matrix of height NUM_GTYPES
                for (int i = 0; i<NUM_GTYPES; i++)      //and width NUM_GENES +
FITNESS_BYTES
                {
                        Generation[i] = new unsigned char[NUM_GENES+FITNESS_BYTES];
                }
        }
```

```
	ifstream Infile;
	Infile.open(filename);
	int Result; //number which will contain the converted string values
	if (!Infile.is_open())
	{
		printf("Error opening file to read! \n");
		return -1;
	}

	string line;
	getline(Infile, line, '\n'); //Get and ignore the first line, 'Generation X'
	line.clear();
	for (int i = 0; i<NUM_GTYPES; i++)
	{
		getline(Infile, line, ' '); //Get and ignore the gtype number, because it
is equal to i
		line.clear();
		for (int j = 0; j<(NUM_GENES+FITNESS_BYTES); j++)
		{
			getline(Infile, line, ' ');

			stringstream convert(line); // stringstream used for the conversion
initialized with the contents of line

			if ( !(convert >> Result) )//give the value to Result using the
characters in the string
				Result = 0;//if that fails set Result to 0
				//Result now equal to 456
			Generation[i][j] = Result;
		}
	}
	Infile.close();
	return 0;
}

int GA_Ops::Find_Exp_Ranges(int Fitness) //Finds the servo ranges for all of the
genotypes with a fitness above int Fitness.
{
	if (Expr_Range == NULL)
	{
		Expr_Range = new ranges[NUM_GENES];
	}

	for (int i = 0; i < NUM_GENES; i++) //Initializes each range so that any new low
and high value will become the new lowest/highest.
	{
		Expr_Range[i].Low = 255;
		Expr_Range[i].High = 0;
		Expr_Range[i].Abs = 255;
	}

	for (int i = 0; i < NUM_GTYPES; i++) //The generation is searched for genotypes
with a high enough fitness
	{
		if (Generation[i][NUM_GENES] >= Fitness) //For each genotype with the
required fitness,
		{
```

```
                    for (int j = 0; j < NUM_GENES; j++) //it's genes are checked against
the current low/high ranges
                    {
                            if(Generation[i][j] < Expr_Range[i].Low) //If the gene is
lower than the current low,
                            {
                                    Expr_Range[i].Low = Generation[i][j];//it becomes the
new current low.
                            }
                            if(Generation[i][j] > Expr_Range[i].High) //Likewise for the
current high range for that gene.
                            {
                                    Expr_Range[i].High = Generation[i][j];
                            }
                    }
            }
      }//After the loop has concluded, the lowest and highest values for each gene will
have been recorded.

      for (int i = 0; i < NUM_GENES; i++)
      {
            Expr_Range[i].Abs = (Expr_Range[i].High - Expr_Range[i].Low);
      }//Finally the absolute range between the low and high for each gene is
calculated.
      //Note that this will produce bad ranges if only a single genotype with the
required fitness is found.
      //If no genotype with the required fitness are found, the absolute range will be
0.
      return 0;
}

int GA_Ops::Gen_Expression() //This function generates a random expression from the
ranges found with the Find_Exp_Ranges
{                           //function, and then converts and sends the generated
expression to the robot for display.
      unsigned char buf[3]; //Serial data buffer. buf[0] is the sync byte, buf[1] is the
servo address, and
      float Temp;           //buf[2] is the servo position byte.
      //int ret = 0;
      unsigned char Rand;
      HANDLE port;
      BOOL success;

      if (Servo_Range == NULL)
      {
            printf("Error! Servo Ranges have not been set.\n");
            return -1;
      }
      else if (Expr_Range == NULL)
      {
            printf("\nError! Ranges have not been found, cannot send. \n");
            return -1;
      }
      else
      {
            buf[0] = SBYTE; //The sync byte, always 255
            printf("\nDisplaying generated genotype. \n");
            port = openPort(COMPORT, BRATE); //Attempts to open the COM port
```

```
		if (port == INVALID_HANDLE_VALUE)
		{
			printf("Error opening ", COMPORT);
			return -1;
		}
		for (int i = 0; i<NUM_GENES; i++)//Each gene is generated and sent in turn
to the robot
		{
			Rand = (rand() % (Expr_Range[i].Abs)) + Expr_Range[i].Low; //The
random value is generated
			Temp = ((float)Rand * (Servo_Range[i].Abs/255.0)) +
Servo_Range[i].Low; //Conversion
			buf[1] = i; //The address of the servo is set
			buf[2] = (int)Temp;//The servo position is set to the converted
value
			//SendBuf(COMPORT, buf, 3);//Sends the buffer to the robot
			success = maestroSetTarget(port, buf[1], buf[2]); // Sends the
position to the target servo
			if (!success){ return -1; }
		}

	}
	CloseHandle(port); //Closes the COM port
	return 0;
}

int GA_Ops::Set_Servo_Ranges() //Before the display or conversion functions are used, the
servo ranges need to be determined
{				//and recorded with this function.  The user can set each
servo range individually, or opt to
				//use the maximum range for each servo.
	int temp_min;
	int temp_max;
	int temp_def;
	char response;
	if (Servo_Range == NULL)
	{
		Servo_Range = new ranges[NUM_GENES];
	}
	printf("\nWould you like to use default range of motion[d/D] or set new
ranges[n/N]?");
	cin>>response;
	cin.ignore();
	if (response == 'd' || response == 'D')
	{
		printf("Using default (255) ranges.\n");
		for (int i = 0; i < NUM_GENES; i++) //Sets all servo ranges to 0-255
		{
			Servo_Range[i].Low = 0;
			Servo_Range[i].High = 254;
			Servo_Range[i].Abs = 254;
			Servo_Range[i].Def = 0;
		}
	}
	else if (response == 'n' || response == 'N')
	{
		printf("\nPlease enter the range of motion for each servo: \n");
		for (int i = 0; i < NUM_GENES; i++)
```

```
		{
			printf("Servo %d\n", i);
			printf("min: ");
			cin>>temp_min;
			cin.ignore(3,'\n');
			Servo_Range[i].Low = temp_min;
			printf("max: ");
			cin>>temp_max;
			cin.ignore(3,'\n');
			Servo_Range[i].High = temp_max;
			printf("default: ");
			cin>>temp_def;
			cin.ignore(3,'\n');
			Servo_Range[i].Def = temp_def;
			Servo_Range[i].Abs = temp_max - temp_min;
			printf("\n");
		}
	}
	else
	{
		printf("Invalid response, returning to main menu.\n");
		delete [] Servo_Range;
		return -1;
	}
	return 0;
}

int GA_Ops::Write_Servo_Ranges(char filename[]) //Allows the user to saves the current
servo ranges for later use
{
	if (Servo_Range == NULL)
	{
		printf("Error! Servo ranges have not been entered.\n");
		return -1;
	}
	else
	{
		ofstream Outfile;
		Outfile.open(filename);
		if (!Outfile.is_open())
		{
			printf("Error opening file to write! \n");
			return -1;
		}
		Outfile <<"Ranges of Motion \n";

		for (int i = 0; i<NUM_GENES; i++)
		{
			Outfile <<Servo_Range[i].Low;
			Outfile <<" ";
			Outfile <<Servo_Range[i].High;
			Outfile <<" ";
			Outfile <<Servo_Range[i].Abs;
			Outfile <<" ";
			Outfile <<Servo_Range[i].Def;
			Outfile <<" ";
		}
		Outfile <<"\n";
```

```
			Outfile.close();
		}
		return 0;
}

int GA_Ops::Read_Servo_Ranges(char filename[]) //Allows the user to read in saved servo
ranges
{
		ifstream Infile;

		if (Servo_Range == NULL)
		{
			Servo_Range = new ranges[NUM_GENES];
		}

		Infile.open(filename);
		int Result; //number which will contain the converted string values
		if (!Infile.is_open())
		{
			printf("Error opening file to read! \n");
			return -1;
		}

		string line;
		getline(Infile, line, '\n'); //Get and ignore the first line, 'Servo Ranges of
Motion'
		line.clear();
		for (int i = 0; i<NUM_GENES; i++)
		{
			//Read the low range
			getline(Infile, line, ' ');
			stringstream convert_0(line); // stringstream used for the conversion
initialized with the contents of line
			if ( !(convert_0 >> Result) )
			{
				//give the value to Result using the characters in the string
				Result = 0;//if that fails set Result to 0
			}
			Servo_Range[i].Low = Result;

			//Read the high range
			getline(Infile, line, ' ');
			stringstream convert_1(line); // stringstream used for the conversion
initialized with the contents of line
			if ( !(convert_1 >> Result) )
			{
				//give the value to Result using the characters in the string
				Result = 0;//if that fails set Result to 0
			}
			Servo_Range[i].High = Result;

			//Read the absolute range
			getline(Infile, line, ' ');
			stringstream convert_2(line); // stringstream used for the conversion
initialized with the contents of line
			if ( !(convert_2 >> Result) )
			{
				//give the value to Result using the characters in the string
```

```
                        Result = 0;//if that fails set Result to 0
                }
                Servo_Range[i].Abs = Result;

                //Read the default value
                getline(Infile, line, ' ');
                stringstream convert_3(line); // stringstream used for the conversion
initialized with the contents of line
                if ( !(convert_3 >> Result) )
                {
                        //give the value to Result using the characters in the string
                        Result = 0;//if that fails set Result to 0
                }
                Servo_Range[i].Def = Result;
        }
        Infile.close();
        return 0;
}

void GA_Ops::Print_All() //Prints the raw values of each genotype in the generation,
including the fitness, which is the last number
{
        float Temp;
        if (Generation == NULL)
        {
                printf("\nError! Generation has not been initialized, cannot print!\n");
        }
        else
        {
                for (int i = 0; i <NUM_GTYPES; i++)
                {
                        printf("Genotype %d: \n", i);
                        for (int j = 0; j<(NUM_GENES+FITNESS_BYTES); j++)
                        {
                                printf("%d ", Generation[i][j]);
                        }
                        printf("\n");
                }
        }
}

void GA_Ops::Print_Gtype(int Gtype)//Prints the gene values and fitness of a single
genotype, designated by Gtype
{
        if (Generation == NULL)
        {
                printf("\nError! Generation has not been initialized, cannot print!\n");
        }
        else if(Gtype <0 || Gtype > (NUM_GTYPES-1))
        {
                printf("Error! Selected Genotype is outside valid range.\n");

        }
        else
        {
                printf("\n");
                printf("Genotype %d: ", Gtype+1);
                for (int i = 0; i<(NUM_GENES+FITNESS_BYTES); i++)
```

```
		{
			printf("%d ", Generation[Gtype][i]);
		}
		printf("\n");
	}
}

void GA_Ops::Print_Elites()//Prints the array index of the elite genotypes in the
generation array.
{
	if (Elites == NULL)
	{
		printf("Error! Elites have not been initialized, cannot print!\n");
	}
	else
	{
		printf("Elite genotypes: ");
		for (int i = 0; i < NUM_ELITES; i++)
		{
			printf("%d", Elites[i]);
			if (i != (NUM_ELITES-1))
			{
				printf(", ");//Places a comma after all the numbers but the
last.
			}
		}
		printf("\n");
	}
}

void GA_Ops::Print_Feedback() //Prints in console the contents of the feedback array.
Un-entered array values show as zeroes.
{
	if (Feedback_Array == NULL)
	{
		printf("Error! Feedback array has not been initialized, cannot print!\n");
	}
	else
	{
		for (int i = 0; i <NUM_GTYPES; i++)
		{
			printf("Genotype %d Feedback: \n", i);
			for (int j = 0; j<FEEDBACK_ARRAY_BYTES; j++)
			{
				printf("%d ", Feedback_Array[i][j]);
			}
			printf("\n");
		}
	}
}

void GA_Ops::Print_Exp_Ranges() //Prints the gene value ranges for the random expression
generator.
{
	if (Expr_Range == NULL)//If the ranges have not been determined, then the function
will return.
	{
		printf("\nError! Ranges have not been found, cannot print. \n");
```

```
	}
	else
	{
		printf("\nLow gene values:  \n"); //The low, high and absolute values are
printed in rows, with each collumn
		for (int i = 0; i < NUM_GENES; i++) //corresponding to a gene.
		{
			printf("%d", Expr_Range[i].Low);
			if (i < NUM_GENES-1)
			{
				printf(", ");
			}
		}
		printf("\n");
		printf("\nHigh gene values: \n");
		for (int i = 0; i < NUM_GENES; i++)
		{
			printf("%d", Expr_Range[i].High);
			if (i < NUM_GENES-1)
			{
				printf(", ");
			}
		}
		printf("\n");
		printf("\nAbsolute range:   \n");
		for (int i = 0; i < NUM_GENES; i++)
		{
			printf("%d", Expr_Range[i].Abs);
			if (i < NUM_GENES-1)
			{
				printf(", ");
			}
		}
		printf("\n");

	}
}

void GA_Ops::Print_Servo_Ranges() //Prints in console the servo ranges of motion that
were set in the
{			//Set_Servo_Ranges() function.
	if (Servo_Range == NULL) //If the ranges have not been set yet, the function will
return.
	{
		printf("\nError! Servo ranges have not been entered, cannot print. \n");
	}
	else //The values are printed in the same way as the expression generator values,
in collumns of numbers for each gene.
	{
		printf("\nLow:  ");
		for (int i = 0; i < NUM_GENES; i++)
		{
			printf("%d ", Servo_Range[i].Low);
		}
		printf("\n");

		printf("High: ");
		for (int j = 0; j < NUM_GENES; j++)
```

```
		{
			printf("%d ", Servo_Range[j].High);
		}
		printf("\n");

		printf("Abs:  ");
		for (int k = 0; k < NUM_GENES; k++)
		{
			printf("%d ", Servo_Range[k].Abs);
		}
		printf("\n");

		printf("Def:  ");
		for (int l = 0; l < NUM_GENES; l++)
		{
			printf("%d ", Servo_Range[l].Def);
		}
		printf("\n");
	}
}

//These functions are meant to be called from the main function, maintaining data hiding
for the GA_Ops class.
int GA_Ops::Get_Num_Gtypes() //Returns the number of genotypes per generation
{
	return NUM_GTYPES;
}

int GA_Ops::Get_Num_Genes() //Returns the number of genes in a genotype
{
	return NUM_GENES;
}
```

### *maestro.h*

```
/* MaestroSerialExampleCWindows:
*  Example program for sending and receiving bytes from the Maestro over a serial port
*  in C on Windows.
*
*  This program reads the position of channel 0.
*  If the position is less than 6000, it sets the target to 7000.
*  If the position is 6000 or more, it sets the target to 5000.
*
*  If channel 0 is configured as a servo channel, the servo should move
*  when you run this program (except perhaps the first time you run it).
*  If channel 0 is configured as a digital output, the output should toggle
*  when you run this program.
*
*  All the Windows functions called by the program are documented on MSDN:
*  http://msdn.microsoft.com/
*
*  The error codes that this program may output are documented on MSDN:
*  http://msdn.microsoft.com/en-us/library/ms681381%28v=vs.85%29.aspx
*
*  The Maestro's serial commands are documented in the "Serial Interface"
*  section of the Maestro user's guide:
```

```
*  http://www.pololu.com/docs/0J40
*
*  For an advanced guide to serial port communication in Windows, see:
*  http://msdn.microsoft.com/en-us/library/ms810467
*
*  REQUIREMENT: The Maestro's Serial Mode must be set to "USB Dual Port"
*  or "USB Chained" for this program to work.
*/

#include <stdio.h>
#include <windows.h>

/** Opens a handle to a serial port in Windows using CreateFile.
* portName: The name of the port.
* baudRate: The baud rate in bits per second.
* Returns INVALID_HANDLE_VALUE if it fails.  Otherwise returns a handle to the port.
*   Examples: "COM4", "\\\\.\\USBSER000", "USB#VID_1FFB&PID_0089&MI_04#6&3ad40bf600004#
*/
HANDLE openPort(const char * portName, unsigned int baudRate);

/** Implements the Maestro's Get Position serial command.
* channel: Channel number from 0 to 23
* position: A pointer to the returned position value (for a servo channel, the units are
quarter-milliseconds)
* Returns 1 on success, 0 on failure.
* For more information on this command, see the "Serial Servo Commands"
* section of the Maestro User's Guide: http://www.pololu.com/docs/0J40 */
BOOL maestroGetPosition(HANDLE port, unsigned char channel, unsigned short * position);

/** Implements the Maestro's Set Target serial command.
* channel: Channel number from 0 to 23
* target: The target value (for a servo channel, the units are quarter-milliseconds)
* Returns 1 on success, 0 on failure.
* Fore more information on this command, see the "Serial Servo Commands"
* section of the Maestro User's Guide: http://www.pololu.com/docs/0J40 */
BOOL maestroSetTarget(HANDLE port, unsigned char channel, unsigned short target);
```

### *maestro.cpp*

```
/* MaestroSerialExampleCWindows:
*  Example program for sending and receiving bytes from the Maestro over a serial port
*  in C on Windows.
*
*  This program reads the position of channel 0.
*  If the position is less than 6000, it sets the target to 7000.
*  If the position is 6000 or more, it sets the target to 5000.
*
*  If channel 0 is configured as a servo channel, the servo should move
*  when you run this program (except perhaps the first time you run it).
*  If channel 0 is configured as a digital output, the output should toggle
*  when you run this program.
*
*  All the Windows functions called by the program are documented on MSDN:
*  http://msdn.microsoft.com/
*
*  The error codes that this program may output are documented on MSDN:
*  http://msdn.microsoft.com/en-us/library/ms681381%28v=vs.85%29.aspx
```

```
*
*  The Maestro's serial commands are documented in the "Serial Interface"
*  section of the Maestro user's guide:
*  http://www.pololu.com/docs/0J40
*
*  For an advanced guide to serial port communication in Windows, see:
*  http://msdn.microsoft.com/en-us/library/ms810467
*
*  REQUIREMENT: The Maestro's Serial Mode must be set to "USB Dual Port"
*  or "USB Chained" for this program to work.
*/

#include <stdio.h>
#include <windows.h>
#include "maestro.h"

/** Opens a handle to a serial port in Windows using CreateFile.
* portName: The name of the port.
* baudRate: The baud rate in bits per second.
* Returns INVALID_HANDLE_VALUE if it fails.  Otherwise returns a handle to the port.
*   Examples: "COM4", "\\\\.\\USBSER000", "USB#VID_1FFB&PID_0089&MI_04#6&3ad40bf600004#
*/
HANDLE openPort(const char * portName, unsigned int baudRate)
{
	HANDLE port;
	DCB commState;
	BOOL success;
	COMMTIMEOUTS timeouts;

	/* Open the serial port. */
	port = CreateFileA(portName, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
	if (port == INVALID_HANDLE_VALUE)
	{
		switch (GetLastError())
		{
		case ERROR_ACCESS_DENIED:
			fprintf(stderr, "Error: Access denied.  Try closing all other
programs that are using the device.\n");
			break;
		case ERROR_FILE_NOT_FOUND:
			fprintf(stderr, "Error: Serial port not found.  "
				"Make sure that \"%s\" is the right port name.  "
				"Try closing all programs using the device and unplugging the
"
				"device, or try rebooting.\n", portName);
			break;
		default:
			fprintf(stderr, "Error: Unable to open serial port.  Error code
0x%x.\n", GetLastError());
			break;
		}
		return INVALID_HANDLE_VALUE;
	}

	/* Set the timeouts. */
	success = GetCommTimeouts(port, &timeouts);
	if (!success)
```

```
	{
		fprintf(stderr, "Error: Unable to get comm timeouts.  Error code 0x%x.\n",
GetLastError());
		CloseHandle(port);
		return INVALID_HANDLE_VALUE;
	}
	timeouts.ReadIntervalTimeout = 1000;
	timeouts.ReadTotalTimeoutConstant = 1000;
	timeouts.ReadTotalTimeoutMultiplier = 0;
	timeouts.WriteTotalTimeoutConstant = 1000;
	timeouts.WriteTotalTimeoutMultiplier = 0;
	success = SetCommTimeouts(port, &timeouts);
	if (!success)
	{
		fprintf(stderr, "Error: Unable to set comm timeouts.  Error code 0x%x.\n",
GetLastError());
		CloseHandle(port);
		return INVALID_HANDLE_VALUE;
	}

	/* Set the baud rate. */
	success = GetCommState(port, &commState);
	if (!success)
	{
		fprintf(stderr, "Error: Unable to get comm state.  Error code 0x%x.\n",
GetLastError());
		CloseHandle(port);
		return INVALID_HANDLE_VALUE;
	}
	commState.BaudRate = baudRate;
	success = SetCommState(port, &commState);
	if (!success)
	{
		fprintf(stderr, "Error: Unable to set comm state.  Error code 0x%x.\n",
GetLastError());
		CloseHandle(port);
		return INVALID_HANDLE_VALUE;
	}

	/* Flush out any bytes received from the device earlier. */
	success = FlushFileBuffers(port);
	if (!success)
	{
		fprintf(stderr, "Error: Unable to flush port buffers.  Error code 0x%x.\n",
GetLastError());
		CloseHandle(port);
		return INVALID_HANDLE_VALUE;
	}

	return port;
}

/** Implements the Maestro's Get Position serial command.
* channel: Channel number from 0 to 23
* position: A pointer to the returned position value (for a servo channel, the units are
quarter-milliseconds)
* Returns 1 on success, 0 on failure.
* For more information on this command, see the "Serial Servo Commands"
```

```
* section of the Maestro User's Guide: http://www.pololu.com/docs/0J40 */
BOOL maestroGetPosition(HANDLE port, unsigned char channel, unsigned short * position)
{
	unsigned char command[2];
	unsigned char response[2];
	BOOL success;
	DWORD bytesTransferred;

	// Compose the command.
	command[0] = 0x90;
	command[1] = channel;

	// Send the command to the device.
	success = WriteFile(port, command, sizeof(command), &bytesTransferred, NULL);
	if (!success)
	{
		fprintf(stderr, "Error: Unable to write Get Position command to serial
port.  Error code 0x%x.", GetLastError());
		return 0;
	}
	if (sizeof(command) != bytesTransferred)
	{
		fprintf(stderr, "Error: Expected to write %d bytes but only wrote %d.",
sizeof(command), bytesTransferred);
		return 0;
	}

	// Read the response from the device.
	success = ReadFile(port, response, sizeof(response), &bytesTransferred, NULL);
	if (!success)
	{
		fprintf(stderr, "Error: Unable to read Get Position response from serial
port.  Error code 0x%x.", GetLastError());
		return 0;
	}
	if (sizeof(response) != bytesTransferred)
	{
		fprintf(stderr, "Error: Expected to read %d bytes but only read %d
(timeout). "
			"Make sure the Maestro's serial mode is USB Dual Port or USB
Chained.", sizeof(command), bytesTransferred);
		return 0;
	}

	// Convert the bytes received in to a position.
	*position = response[0] + 256 * response[1];

	return 1;
}

/** Implements the Maestro's Set Target serial command.
* channel: Channel number from 0 to 23
* target: The target value (for a servo channel, the units are quarter-milliseconds)
* Returns 1 on success, 0 on failure.
* Fore more information on this command, see the "Serial Servo Commands"
* section of the Maestro User's Guide: http://www.pololu.com/docs/0J40 */
BOOL maestroSetTarget(HANDLE port, unsigned char channel, unsigned short target)
{
```

```
	unsigned char command[3];
	//unsigned char command[4];
	DWORD bytesTransferred;
	BOOL success;

	// Compose the command.
	command[0] = 255;
	command[1] = channel;
	command[2] = target;
	//command[3] = (target >> 7) & 0x7F;

	// Send the command to the device.
	success = WriteFile(port, command, sizeof(command), &bytesTransferred, NULL);
	if (!success)
	{
		fprintf(stderr, "Error: Unable to write Set Target command to serial port.
Error code 0x%x.", GetLastError());
		return 0;
	}
	if (sizeof(command) != bytesTransferred)
	{
		fprintf(stderr, "Error: Expected to write %d bytes but only wrote %d.",
sizeof(command), bytesTransferred);
		return 0;
	}

	return 1;
}
```

### *Main.cpp*

```
#include "Algorithm.h"

//Prototype functions for main, implemented below the main function
void ReadGen();
int Randomize();
int Crossover();
int Mutate();
void WriteGen();
void WriteConv();
int PrintGtype();
int CalcFitness();
int Feedback();
int SendGtype();
int Write_Servo_Range();
int Read_Servo_Range();

GA_Ops Test; //Genetic algorithm operations class instantiation

int main(void)
{
	int mainmenu_choice = 0; //Int that hold the menu choice of the user

	//Debug
	printf("NUM_GTYPES = %d\n", NUM_GTYPES);
	printf("NUM_GENES = %d\n", NUM_GENES);
```

```
while (mainmenu_choice != 25)
{
	printf("\n-----Facial Expression Genetic Algorithm-----\n");
	printf("\nMain Menu\n");
	printf("1 : Initialize New Generation\n");
	printf("2 : Read A Generation From File\n");
	printf("3 : Randomize a Genotype\n");
	printf("4 : Crossover Two Genotypes\n");
	printf("5 : Mutate a Genotype\n");
	printf("6 : Provide Feedback for a Genotype\n");
	printf("7 : Write Current Generation Out to File \n");
	printf("8 : Print a Genotype \n");
	printf("9 : Print Current Generation \n");
	printf("10 : <Removed> \n");
	printf("11 : Convert Current Generation to Servo Values \n");
	printf("12 : Calculate Fitness for Current Generation\n");
	printf("13 : Find Elites for the Current Generation\n");
	printf("14 : Print Elites\n");
	printf("15 : Print Feedback for Current Generation\n");
	printf("16 : Find Random Expression Ranges \n");
	printf("17 : Print Random Expression Ranges \n");
	printf("18 : Generate a Random Expression\n");
	printf("19 : Send Genotype to Robot\n");
	printf("20 : Set Servo Range of Motion\n");
	printf("21 : Send Elite Genotypes to Robot\n");
	printf("22 : Write Servo Range of Motion to File\n");
	printf("23 : Read Servo Range of Motion from File\n");
	printf("24 : Print Servo Range of Motion\n");
	printf("25 : Exit\n");
	printf("\nPlease enter a menu option : ");
	cin>>mainmenu_choice;
	cin.ignore(5, '\n');
	if (mainmenu_choice == 1)
	{
		Test.Init_Generation();
	}
	else if (mainmenu_choice == 2)
	{
		ReadGen();
	}
	else if (mainmenu_choice == 3)
	{
		Randomize();
	}
	else if (mainmenu_choice == 4)
	{
		Crossover();
	}
	else if (mainmenu_choice == 5)
	{
		Mutate();
	}
	else if (mainmenu_choice == 6)
	{
		Feedback();
	}
	else if (mainmenu_choice == 7)
	{
```

```
        WriteGen();
}
else if (mainmenu_choice == 8)
{
        PrintGtype();
}
else if (mainmenu_choice == 9)
{
        Test.Print_All();
}
else if (mainmenu_choice == 10)
{

}
else if (mainmenu_choice == 11)
{
        WriteConv();
}
else if (mainmenu_choice == 12)
{
        CalcFitness();
}
else if (mainmenu_choice == 13)
{
        Test.Find_Elites();
}
else if (mainmenu_choice == 14)
{
        Test.Print_Elites();
}
else if (mainmenu_choice == 15)
{
        Test.Print_Feedback();
}
else if (mainmenu_choice == 16)
{
        Test.Find_Exp_Ranges(7);
}
else if (mainmenu_choice == 17)
{
        Test.Print_Exp_Ranges();
}
else if (mainmenu_choice == 18)
{
        Test.Gen_Expression();
}
else if (mainmenu_choice == 19)
{
        SendGtype();
}
else if (mainmenu_choice == 20)
{
        Test.Set_Servo_Ranges();
}
else if (mainmenu_choice == 21)
{
        Test.Display_Elites();
}
```

```
		else if (mainmenu_choice == 22)
		{
			Write_Servo_Range();
		}
		else if (mainmenu_choice == 23)
		{
			Read_Servo_Range();
		}
		else if (mainmenu_choice == 24)
		{
			Test.Print_Servo_Ranges();
		}
		else if (mainmenu_choice != 25) //If the user did not enter any valid
choice, they get the menu again.
		{
			printf("Invalid Menu Choice.  Please enter a valid option. \n");
		}
	}

	return 0;
}

//Function implementations for main

void ReadGen()
{
	char * Fname;
	char input[100];
	printf("Enter the filename of the input file: ");
	cin.getline(input, 100, '\n');
	Fname = new char[sizeof(input)];
	strcpy(Fname,input);
	Test.Read_Generation(Fname);
	delete [] Fname;
}

int Randomize()
{
	int Gtype = 0;
	printf("\nPlease enter the number of the genotype to randomize. \n");
	printf("[Integer from 0 - %d]", NUM_GTYPES-1);
	cin>>Gtype;
	cin.ignore(5, '\n');
	Test.Randomize(Gtype);
	return 0;
}

int Crossover()
{
	int GtypeA = 0;
	int GtypeB = 0;
	printf("\nPlease enter the number of the first genotype to cross over. \n");
	printf("[Integer from 0 - %d]", NUM_GTYPES-1);
	cin>>GtypeA;
	cin.ignore(5, '\n');

	printf("\nPlease enter the number of the second genotype to cross over. \n");
	printf("[Integer from 0 - %d]", NUM_GTYPES-1);
```

```
        cin>>GtypeB;
        cin.ignore(5, '\n');

        Test.Crossover(GtypeA, GtypeB);
        return 0;
}

int Mutate()
{
        int Gtype = 0;
        printf("\nPlease enter the number of the genotype to mutate. \n");
        printf("[Integer from 0 - %d]", NUM_GTYPES-1);
        cin>>Gtype;
        cin.ignore(5, '\n');
        Test.Mutate(Gtype);
        return 0;
}


void WriteGen()
{
        char * Fname;
        char input[100];
        printf("Enter the filename for the new output file: ");
        cin.getline(input, 100, '\n');
        Fname = new char[sizeof(input)];
        strcpy(Fname,input);
        Test.Write_Generation(Fname);
        delete [] Fname;
}

int PrintGtype()
{
        int Gtype = 0;
        printf("\nPlease enter the number of the genotype to print. \n");
        printf("[Integer from 0 - %d]", NUM_GTYPES-1);
        cin>>Gtype;
        cin.ignore(5, '\n');
        Test.Print_Gtype(Gtype);
        return 0;
}

void WriteConv()
{
        char * Fname;
        char input[100];
        printf("Enter the filename for the new converted output file: ");
        cin.getline(input, 100, '\n');
        Fname = new char[sizeof(input)];
        strcpy(Fname,input);
        Test.Write_Gen_Convert(Fname);
        delete [] Fname;
}

int CalcFitness()
{
        int Gtype = -1;
        printf("\nPlease enter the number of the genotype to calcuate fitness for. \n");
```

```
        printf("[Integer from 0 - %d]", NUM_GTYPES - 1);
        cin>>Gtype;
        cin.ignore(5, '\n');
        Test.Calc_Fitness(Gtype);
        return 0;
}

int Feedback()
{
        int Gtype = -1;
        printf("\nPlease enter the number of the genotype to provide feedback for. \n");
        printf("[Integer from 0 - %d]", NUM_GTYPES - 1);
        cin>>Gtype;
        cin.ignore(5, '\n');
        Test.Feedback(Gtype);
        return 0;
}

int SendGtype()
{
        int Gtype = 0;
        printf("\nPlease enter the number of the genotype to send to the robot. \n");
        printf("[Integer from 0 - %d]", NUM_GTYPES-1);
        cin>>Gtype;
        cin.ignore(5, '\n');
        Test.Send_Gtype(Gtype);
        return 0;
}

int Write_Servo_Range()
{
        char * Fname;
        char input[100];
        printf("Enter the filename to write the servo ranges to: ");
        cin.getline(input, 100, '\n');
        Fname = new char[sizeof(input)];
        strcpy(Fname,input);
        Test.Write_Servo_Ranges(Fname);
        delete [] Fname;
        return 0;
}

int Read_Servo_Range()
{
        char * Fname;
        char input[100];
        printf("Enter the filename containing the servo ranges: ");
        cin.getline(input, 100, '\n');
        Fname = new char[sizeof(input)];
        strcpy(Fname,input);
        Test.Read_Servo_Ranges(Fname);
        delete [] Fname;
        return 0;
}
```

## Hours Spent

Brett Dunscomb: 56 hours
Kevin Bedrossian: 50 hours
Caren Zgheib:  52 hours

## References


1- Stephen Mugglin, *Music Theory For Song Writers*: http://mugglinworks.com/chordmaps/
2- Zgheib, Caren, et al. "A Musical Classification and Interpretation of Abstract Art." (2013).: http://dl.acm.org/citation.cfm?id=2523429.2531848&coll=DL&dl=ACM&CFID=388098961&CFTOKEN=88480841
3- http://www.instructables.com/id/Simple-Animatronics-robotic-hand/