**EDDIE: The Affective Autism Therapy Robot**

Jon Mrowczynski, Christine Colson, Robert Selkowitz, Jonathan Rogers,

*Department of Physics, Institute for Autism Research, Canisius College, Buffalo New York 14208*

EDDIE (Emotion Demonstration, Decoding, Interpretation, and Encoding) is an interactive AI system for the use of children with High Functioning Autism Spectrum Disorders (HFASD). EDDIE can interactively help teach HFASD children how to display and interpret emotions as well as give constructive feedback on the subjects attempt to replicate these cues as defined by Ekman’s Facial Action Units (FAUs). The demonstration of these emotions is done using a robotic, puppet like face while the interpretation and feedback is given using a Kinect.

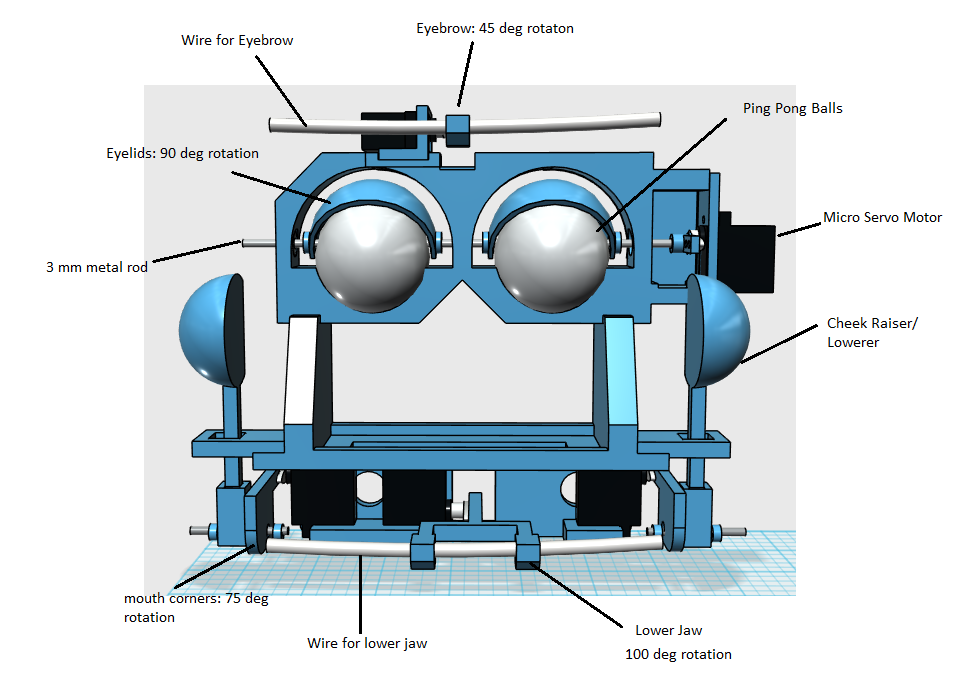
**I. INTRODUCTION**

Autism is a group of complex disorders of brain development that is becoming a more prevalent issue. Autism Spectrum Disorders (ASD) can range from a child being completely nonverbal and non-interactive to having minimal deviation from normal development; but one common deficit is the inability to recognize and appropriately express emotion.1

It has been shown that six emotions are cross culturally expressed identically: Happy, Sad, Angry, Surprised, Scared, and Disgusted.2 Paul Ekman came up with a system of discretized actions, Facial Action Units (FAUs), which compose the expression of these emotions.3

**II. THE MECHANICAL SETUP**

The skeleton of the robot consists of three primary 3D printed parts to mount the motors and several printed moving parts (as seen in Figure 1 below). Other components are three segments of a metal rod, flexible plastic tubing used for the eyebrows and the lower lip, and five micro servo motors.



**Figure 1. EDDIE Mechanical Setup**

The eye plate mounts two motors for the eyebrow and the eyelids. The eyebrow motor is mounted on top of the eye plate and the motor’s horn is mounted with an arm that holds a flexible tube. The eyelid motor is mounted to the side of the eye plate and friction fit to a metal rod which holds both the eyelids and the eyes, which are made out of beer pong balls.

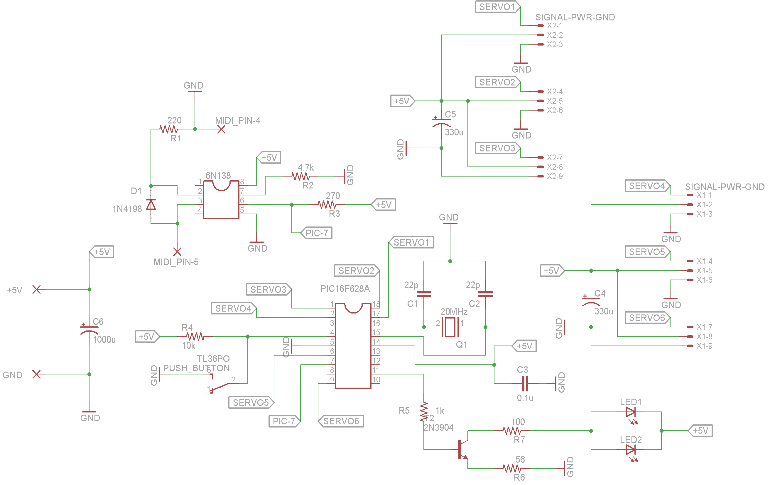
The jaw plate holds three motors: the right and left lip corners and the lower jaw. The right and left lip corners are made from arms which attach to the servo horns and extend out. These motors also control the cheeks’ rising and falling. A rod threads through the end of the lip arm and friction fits into a lollipop like structure which creates a bulge, effectively raising and lowering the cheeks in sync with the lip corner movements, mimicking facial muscle groups. The lower lip is similar in design to the eyebrow, but extends further out. It is also designed to hold flexible tubing which can move up and down to imitate talking.

The structure holds no motors and is designed primarily to maintain a stable separation between the jaw and eye plates as well as to stabilize/guide the cheek raisers.

**III. ELECTRONICS**

The entire circuit is rather simple, as shown in Figure 2, consisting of three sub-circuits, a current amplifier, a Programmable Integrated Circuit (PIC) 16F628A servo controller, and a Musical Instrument Digital Interface (MIDI) IN circuit.

The MIDI IN circuit contains an optoisolator chip which performs two functions. The first being that it isolates the MIDI IN sub-circuit from the servo controller circuit by transmitting the MIDI signals from a photodiode to a phototransistor. This prevents any voltage surges that the computer may inadvertently transmit to the microcontroller via a USB to MIDI cable from damaging the sensitive microcontroller. In addition, the optoisolator inverts the electrical signals that the computer transmits so that the voltage levels associated with those transmitted MIDI messages are converted to a proper voltage level (between 3.0-5.5V) for the PIC’s Receive Pin (RX) to be capable of reading those transmitted MIDI messages.



**Figure 2. EDDIE circuit diagram4**

The current amplifier is used to power the eye lights (LEDs) of EDDIE using the 5V, 8A power supply rather than powering them directly from the PIC. This is to avoid using the very limited amount of output power that the PIC can supply compared to the practically unlimited amount of power that can be drawn from the power supply.

The PIC servo controller circuit is a fairly standard servo controller circuit where up to six of the PIC’s output pins can send Pulse Width Modulation (PWM) signals to servo motors. However, to get greater stability and precision over the outputted PWM signals, an external 20MHz quartz crystal resonator is used in place of the microcontroller’s less stable internal 4MHz RC circuit.

**IV. FIRMWARE**

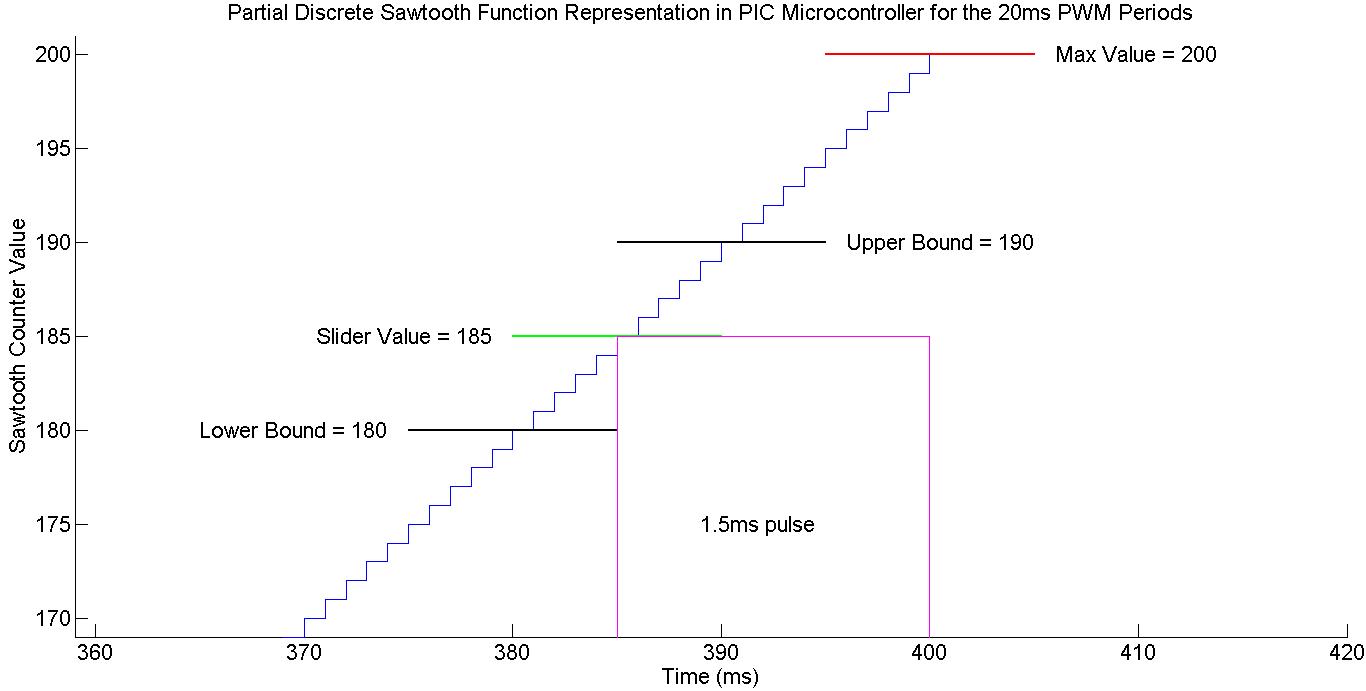
The implemented servo controller algorithm allows the PIC to control up to six servo motors based on the data contained within MIDI messages that are transmitted from a computer to the microcontroller. Each MIDI message that is sent is composed of three bytes. The first byte is the command byte, which always needs to be sent, but for our purposes it is simply disregarded. The second byte represents the pitch of the note which is used to dictate which servo motor is to be operated. The specific set of pitches or notes used does not matter as long as there is an agreement in the communication protocol between the firmware and the software. The third and final byte represents the velocity of the note which, in our case, dictates the position that the servo motors’ arms should move to. Ultimately, it is the second and third byte of the MIDI messages that tells the PIC which servo motor arm to operate.

However, in order for the microcontroller to successfully process the MIDI messages that it receives, the rate at which the PIC processes those received signals must be configured to match the rate at which the signals change for the sent MIDI messages. Since the Baud RATE (BRATE) for MIDI is 31250 signal changes per second, the PIC’s Serial Port Baud Rate Generator (SPBRG) register must be set to an appropriate value to receive those signal changes at that rate. The value that the SPBRG register should be set to can be determined using the following formula:

, (1)

where Fosc is the frequency of the internal/external oscillator that the PIC is implementing and our desired baud rate value is 31250. Note that this formula only applies if the microcontroller’s High Baud Rate select bit (BRGH) is set to 1, otherwise the desired baud rate should be multiplied by 64 rather than 16. Since our circuit uses a 20MHz external quartz crystal resonator, we set our PIC’s SPBRG register to 39. Also, note that for this set of values for Fosc and the desired baud rate, the baud rate error is equal to zero since the desired baud rate is equivalent to the calculated baud rate. The baud rate error will always equal zero if and only if the fraction is an integer value. The microcontroller’s frequency of oscillation should be adjusted such that this is the case.

To make the reception of the MIDI messages efficient, which is imperative for this application, the Universal Synchronous Asynchronous Receiver Transmitter (USART) ReCeive Data REGister (RCREG) interrupt flag is set such that the PIC’s interrupt service routine fires. This is significantly more efficient than the alternative option of having the PIC poll its RCREG register to see if it contains any data that needs to be processed since it prevents the PIC from continuously using its very limited computational resources. Note that the PIC’s RCREG is only a two byte First In, First Out (FIFO) buffered register, however, the MIDI messages that are sent are three bytes. The third byte shifts to the PIC’s Receive (serial) Shift Register (RSR) which eventually gets transferred to its RCREG register after the first stored byte in the FIFO gets read into program memory.



**Figure 3. Part of the second period of the PWM signal from the start of the PIC program. The digital pulse that is sent to the servo motor can be seen as the square wave which starts at the point where the sawtooth counter value equals the slider value and ends when the sawtooth counter value equals the maximum threshold value. Note that only one servo motor slider value is shown instead of all six for simplicity’s sake.**

In order to operate the servo motors, a PWM signal must be used. Since the PIC16F628A only possesses one Capture/Compare/PWM (CPP) module, an algorithm that would allow any of the PIC’s output pins to produce a PWM signal of various pulse widths had to be implemented. This can be accomplished using Timer2 match interrupts. Every fourth tick of the PIC’s clock increments the value stored in its TiMeR 2 (TMR2) register and when that value is equivalent to the value stored in the Timer2 Period register (PR2) the Timer2 interrupt flag is set, firing the PIC’s interrupt service routine. The micro servo motors that were used need a PWM signal with a period of 20ms with the pulse width lasting between 1-2ms, where a fully clockwise rotation corresponds to a pulse width of 1.0ms and a fully counterclockwise rotation corresponds to a pulse width of 2.0ms. A 0.1ms Timer2 interrupt period resolution is quite sufficient for this application, however, in order to calculate what values to set the various registers of the PIC to acquire such a resolution, the following formula must be used

(2)

where TMR2 is the value that the TMR2 register will be reset to upon a match interrupt, Tosc­ is the period of the PIC’s internal or external clock in seconds, PR2 is the value stored in the Timer2 period register, and Spre and Spost are the Prescaler and Postscaler values that are set in the Timer2 configuration register. For this application, these values were set to tinterrupt = 0.0001s, TMR2 = 0, Tosc = s, Spre = 4, and Spost = 1, which would yield a PR2 value of 124.

To adjust the PWM pulse width for each of the six servo motors, a counter is incremented every 0.1ms when a Timer2 interrupt is fired, creating a discretely changing sawtooth function. A max threshold value of 200 is set such that when the counter reaches 200, it gets reset to 0. This allows for the PWM signal to have a period of 20ms. Secondly there are two more threshold values that are set, one at 180 and another at 190. These provide a lower and upper bound for the servo motor’s slider value such that it can take on any value in between 180 and 190 inclusive. Whenever the sawtooth counter value is equal to the servo motor slider value, the output pin is set to high (+5V), and whenever the sawtooth counter value is equal to the maximum threshold value, the output pin is reset to low (+0V). The graphical representation for this part of the process during the second period of the PWM signal from the start of the program is shown in Figure 3 for a servo motor slider value of 185 resulting in a pulse width of 1.5ms.

Efficiency cannot be stressed enough for this application. The microcontroller needs to execute all of its tasks as quickly as possible to avoid inadvertent PWM pulse width variations which would result in servo motor arm twitching. Another more fatal problem that will be encountered if the program is not optimized is RCREG overrun errors. If the microcontroller cannot read in the data that is stored in its RCREG register fast enough because it is carrying out other processes, and it receives another MIDI message, the RCREG register will overflow with data resulting in an overrun error. This causes MIDI data to be lost in addition to rendering the PIC inoperable until its Continuous Receive ENable (CREN) bit is cleared. In order to further increase efficiency, the C code is translated into assembly and it is optimized for speed before it is written to the PIC’s flash memory.

The entire MPLabX Project that contains the firmware can be found on the Bitbucket repository.4

**V. SOFTWARE**

There are two main components to the software. The first is the master control software which is written in Java and deals with everything MIDI related. The second is the Kinect v2 software which is written in C++ and C# and deals with the emotional analysis of the user based on their facial expressions.

The master control software currently performs two main functions. Firstly, it constructs all of the necessary MIDI messages that the microcontroller can understand such that they can be sent to the PIC later on when the user wants to manipulate the ruppet’s FAUs and emotional states. For example, if the user wants the ruppet to transition to its happy emotional state, a collection of already constructed *ShortMessage*s are consecutively sent the microcontroller to operate the servos such that the ruppet displays its happy emotion. This method of controlling the ruppet is currently performed using a Command Line Interface (CLI). Secondly, the master control software is also capable of encapsulating these *ShortMessage*s in *MidiEvent*s which are *MidiMessage*s that are to be played at a specified time. These *MidiEvent*s are then added to *Track*s that are then added to a *Sequence* such that the system’s default *Sequencer* can later play these temporally organized *ShortMessages* on command. This process allows the ruppet to carry out scripts that involve lower jaw movements, audio for vocals, and emotional state transitions. However, in order to run these scripts, a prerecorded audio file for the voice and text files that contain information about the lower jaw movements and the emotional state transitions need to be present.

The text files are structured in two different formats, one for the emotional transitions and the other for the lower jaw movements. The emotional transitions text file associates *float*s with *Strings*, representing the emotion that the ruppet should transition to at the specified time. The lower jaw movement text file simply contains *float*s associated with a *char*, either a ‘D’ or a ‘U’ indicating when the lower jaw should move down or up respectively. Each line of information dictates which set of *ShortMessage*s are associated with which times to construct the *MidiEvent*s that will later be played by the *Sequencer*. This process can be done either during the compilation process or runtime, allowing the ruppet to have more of an autonomous presence, especially with further development.

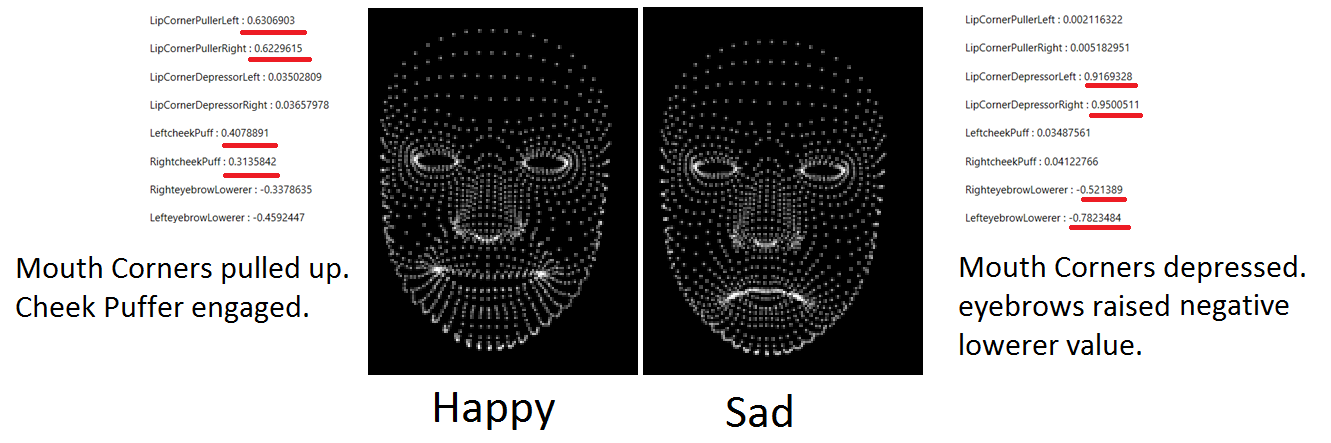
In order for the computer to successfully implement both of these functionalities, the master control software acquires both the *Receiver* of the USB to MIDI cable, which connects the computer to the microcontroller, and the computer’s default system’s *Sequencer*. The *Receiver* must be obtained in order to send MIDI messages to the microcontroller while the *Sequencer* is used to play a sequence of MIDI messages that have been constructed prior to the initiation of the played *Sequence*. However, in order to setup the *Sequencer* to have enough ticks per second to operate the ruppet without any delay and without sacrificing efficiency, one must use the formula

, (3)

where resolution is the number of pulses per the division type note, and TBPM is the Tempo in Beats Per Minute. For this application, the desired value for *ticksPerSecond* is 1,000. A division type set to *Sequencer.PPQ* (a pulse per quarter note), and with the resolution and tempo in beats per minute set to 160 pulses per quarter note and 375 respectively results in a *ticksPerSecond* value of 1,000 or 1 tick every millisecond.

The Kinect is able to capture over 1,000 High Definition facial points at 30 frames per second to collect information about the user’s face. Specifically the software focuses on the user’s *FaceShapeAnimationUnits* since there is a practically one-to-one correspondence between them and the person’s FAUs. With each *FaceShapeAnimationUnit,* such as *LipCornerPullerLeft*, there is a floating point value that is associated with it which represents how much that part of the person’s face is in that state. Most of these values range from 0.0-1.0 where 0.0 means that, in this case, the user is not pulling their left lip corner whatsoever while a value of 1.0 means that the user is significantly pulling their left lip corner.

A calibration routine is currently implemented to detect a person’s emotional state from their facial expression, where the program first averages the *FaceShapeAnimationUnit* values of the person’s neutral face over a five second period. Then, if the *FaceShapeAnimationUnit* value for that user is greater than the sum of their statistically determined neutral value and a hardcoded offset value unique to that *FaceShapeAnimationUnit*, then the threshold value is cleared and the program knows that they are displaying the corresponding FAU. This is done for various FAUs such that four basic emotions can be detected from facial expressions, specifically Happy, Sad, Angry, and Neutral. For example, if the user clears the threshold value that is associated with their *LipCornerPullerLeft* and *LipCornerPullerRight* values, then the program knows that the user must be expressing Paul Ekman’s version of happy.



**Figure 4. A visual representation of the high definition facial points along with the correlation coefficients of the *FachShapeAnimationUnit*s of interest.**

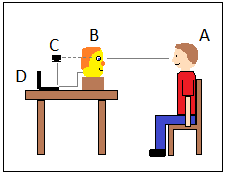
Both the Eclipse project and the Microsoft Visual Studio 2013 project can be found on the Bitbucket repository.4

**VI. AUTISM INTERVENTIONS**

The purpose of EDDIE is to help HFASD children read and express emotions. Its intended use is in addition to a therapy program under the supervision of a professional.

**A. Setup and usage**

The figure below demonstrates the setup. The subject should be seated at eye level (A) with the Kinect (C) (which ideally would be located in EDDIE’s eyes). The robot (B) and the Kinect would be connected to a computer (D).



**Figure 5. Intended setup for robot usage**

EDDIE would talk to the child, demonstrate an expression and have the child mimic it. The Kinect would interpret the child’s response and provide constructive criticism. This criticism is based on the subject’s expression of the demonstrated FAUs.

**B. Advantages of Robotic Intervention**

Robotic intervention has many advantages over traditional one on one therapy approaches. First, most autistic children have a certain amount of anxiety regarding social situations and human interaction. Working with a robot greatly reduces this stress and allows the child to focus more on the task at hand.1

Secondly, it frees up time for the professionals who would generally be administering the therapy. Since the children are generally likely to avoid social interaction, it is a win for both parties.

Thirdly, the ability to demonstrate emotions consistently and reliably over time is helpful in recognition of emotional states, as opposed to relying on a human to make the same face repeatedly.

Finally, this robot has the potential to be used in the home providing more opportunity for practice.

**VII. FUTURE UPGRADES**

This version of EDDIE has several limitations that will need to be addressed in future models.

One of the most significant drawbacks to the current device is that it uses five low power motors which severely limit range of motion when the fabric is put on the face. To overcome this, future versions will replace the micro servo motors in the lip corners and eye brow with larger motors.

Future versions will also have the ability to express more FAUs by adding a head tilt mechanism and separating control of the inner and outer eyebrow.

There are a few alterations that can be made to the electronics that would significantly improve the quality of the device. Firstly, a MIDI OUT circuit could be incorporated such that there can be cross communication between the circuit and the computer. This, in conjunction with angular position sensors hooked up in such a way that allows for each of the positions of the movable parts of the ruppet to be detected, would allow for the ruppet to automatically calibrate itself. This would save much time and effort setting up the device since the manual calibration process is extremely tedious and time consuming.

In addition, a wireless communication system such as Bluetooth Low Energy (BTLE) can be used in place of the current USB to MIDI cable. This would help give a greater illusory quality to the ruppet as well as reduce the proximity constraints that are currently imposed, being that the ruppet needs to be within a meter of a computer for it to be operated from.

Finally, putting the entire circuit onto a printed circuit board that is integrated into the mechanical design would also help increase the ruppet’s illusory quality in addition to increasing the reliability of the electronics since wires tend to disconnect from the breadboard.

While both of the two main components of the software are independently functional, the Kinect software does not currently communicate with the master control software. Eventually they will be linked using Java’s Native Interface (JNI) such that EDDIE can use the data collected by the Kinect to successfully interact with the user. Also, from a user standpoint, the program is somewhat awkward to operate with the CLI. To help improve the clarity of the options presented and the efficiency of the operation of the ruppet, a Graphical User Interface (GUI) can be used instead.

Possibly the most important improvement that could be performed, is finding a more efficient method of putting a more functional and aesthetically pleasing face onto the ruppet where the motors are not under additional stress due to the resistance from the fleece. The current setup puts a severe limitation on which motors can be used to successfully operate the ruppet and it prevents the circuit from being practically powered by a battery since the power requirement for the servo motors is significantly higher when they are under load. If this issue can be fixed, then one could be able to successfully use only micro servo motors and, as a result, eventually power the ruppet by the use of a rechargeable battery, completing the illusion that EDDIE is an independently functioning entity.

**REFERENCES**

1Institute for Autism Research at Canisius College

2P. Ekman, *An Argument for Basic Emotions* (1992).

3P. Ekman and W.V. Friesen, *The Facial Action Coding System:* *A Technique for the Measurement of Facial Movement*. San Francisco: Consulting Psychologists Press, 1978.

4https://sleepwonder@bitbucket.org/canisiuseddie/canisiuseddie.git