Last Name	First Name
Chai	Tony
Yi	John

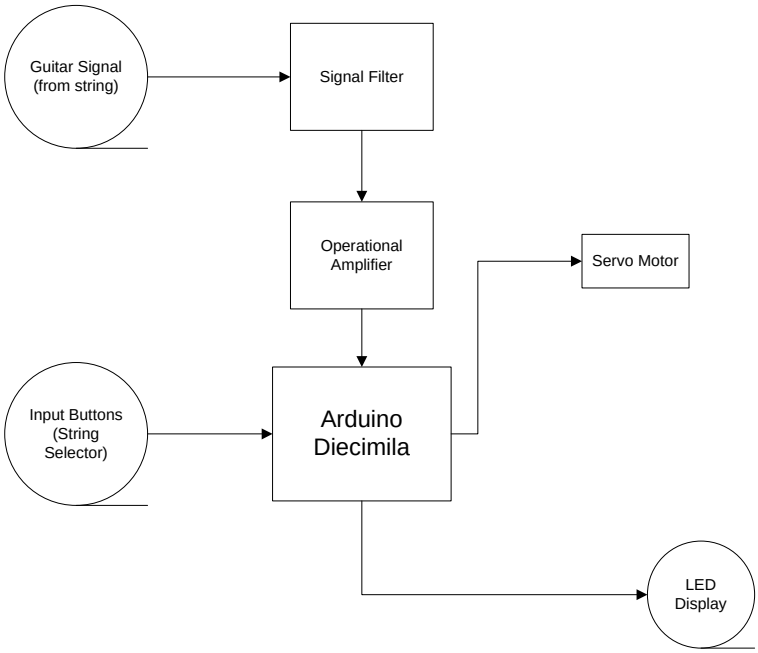
Magic Tune Guitar Tuner

Abstract

This project entails creating a full-functioning automatic guitar tuner. It will be capable of tuning guitar string pegs automatically to tune a guitar to a variety of tunings. This provides a remedy for two types of guitarists: one is for the professional, who may not be able to tune his strings by ear in a loud environment and wants his strings tuned instantly. The other is for the novice, who may not have the trained relative pitch necessary to tune the guitar himself. We also aimed to make this device affordable, so that the potential users would not be intimidated by the price point for being able to tune his guitar conveniently.

The tuner will be manually attached to the target guitar peg. The guitarist will use a button to select which string he wants to tune, and then pluck the according string for the tuner to correct the guitar tuning. If the guitar is in tune, a green LED will light up indicating that it is in tune. If it is either sharp (too high) or flat (too low), the tuner will light up the corresponding LED and twist the guitar pegs accordingly until it is in tune.

Project Block Diagram (High level)

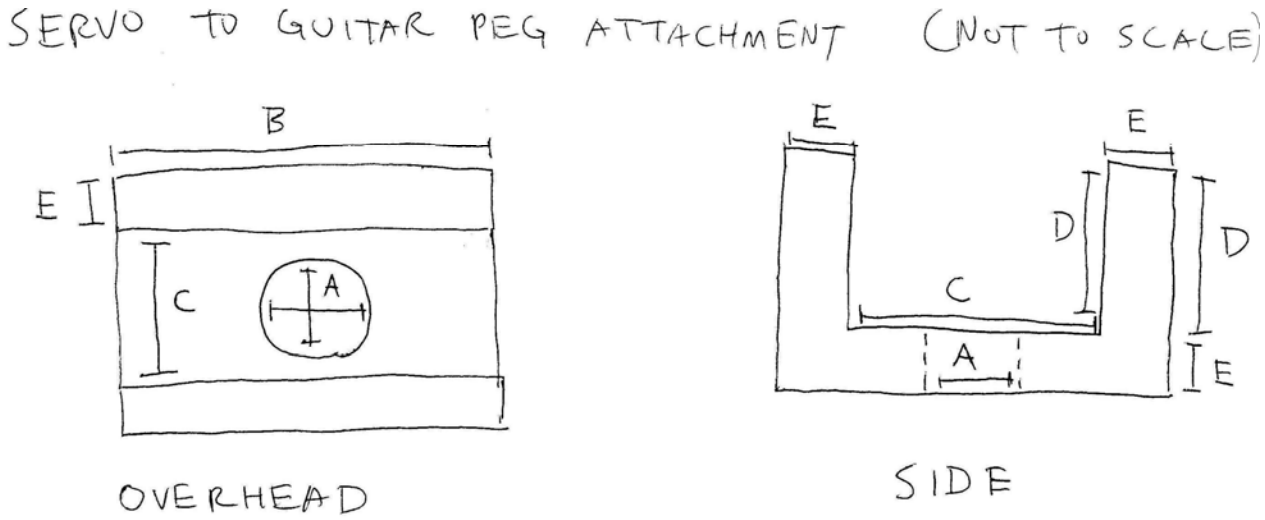


Project Components

• *Mechanical engineering*
We are using a continuous servo motor to wind the guitar pegs. The DC motor provides enough torque to wind a tuning peg if all the power from the Arduino is used for it. Thus, while the motor is turning, the rest of the circuit is temporarily disabled because the code is only focused on turning the motor in the loop. If we had the entire circuit working simultaneously with the motor turning, the Arduino wouldn't have enough power to turn the guitar pegs with the appropriate torque as the USB port provides only 5 volts.

We designed a servo motor attachment to fit on the guitar pegs which would attach onto the existing plastic turning piece on the servo. Our attachment was a sort of "shell" for the smaller piece, as the smaller piece fit into our attachment perfectly but would not be able to turn the peg by itself since it was too small.

Diagram of our servo motor attachment:



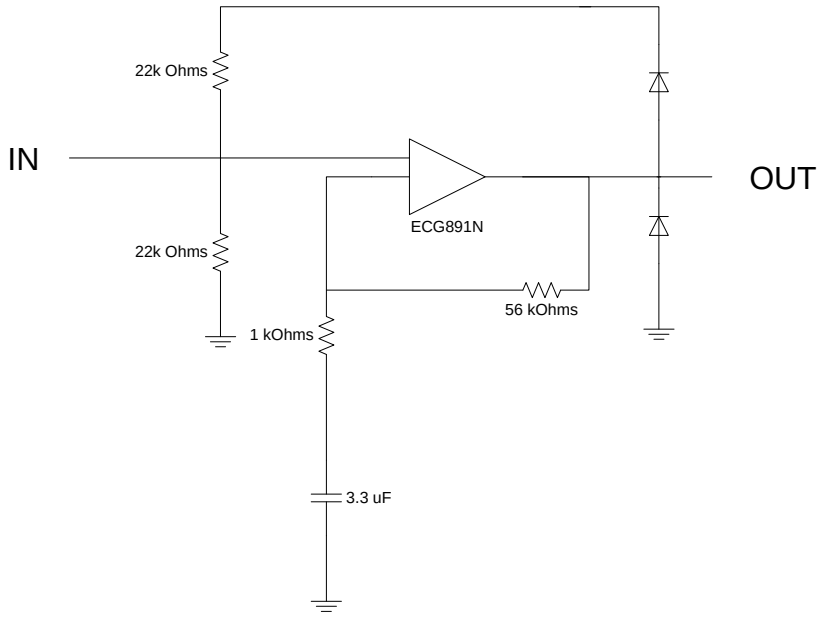
KEY:

A: Hole diameter for original servo piece:	3/8 inches
B: Length of attachment:	11/8 inches
C: Width of space for peg:	7/16 inches
D: Height of attachment inside:	5/8 inches
E: Thickness of attachment:	1/16 inches

• *Electrical*
We are sending a direct signal from an electric guitar's sound jack, amplifying it through an op-amp with a signal filter, and sending it to the ATmega128. The microcontroller is connected to a servo motor and will rotate the motor depending on the user selection using a button and 6 LEDs (each LED resembles its corresponding string). As you can see in our op-amp circuit, we have a

gain of 57 [1 + (56k/1k)] to bring up the voltage to around 5V for the Arduino to read properly. We also added diodes as a fail-safe for the circuit.

Op-Amp Circuit:



• *Software (Programs)*
We are translating the analog signal into a digital signal that will control the rate and direction of each motor using our Arduino board. Within our Arduino code, we are filtering out the overtones of each string using a Butterworth filter to isolate the fundamental frequency of the guitar signal. We count how many times the sinusoidal signal crosses a voltage threshold on the y-axis for a period of time on the x-axis in a function. We call this function with the Atmega's internal timer to get a consistent period for sampling.

After the string input has been idle (meaning the cross counts have been below threshold for a period of time), we then take the average of a few values for cross counts that were within the boundaries of the string (eliminating further possibility of higher frequencies altering our averages) to check if the string is in tune. We found the "in tune" averages and stored them in the Arduino. The microprocessor compares the computed average from the string input to the "in tune" average for each string to determine if it is in pitch.

If the string is in tune, the Arduino will only light up the corresponding LED. If it is out of tune, the Arduino will not only light up the corresponding LED, it will send a PWM signal to the servo motor to turn it either direction depending on if the string is too high or low.

The code also considers the user input from the button, changing the variables for pitch detection depending on the selected string (and lighting up the corresponding display LED)

```
/*
 * Magic_Tune
 *
 * This program will receive an amplified analog signal, compare the pitch, and control the servo motor while displaying output LEDs
*/


//Libraries required for servo control and usage of the Atmega's internal timer
#include <avr/interrupt.h>
#include "WProgram.h"

//Global variables to be used
int x;
int y;
int lastx;
int lasty;
long timer;
int idle_timer;
int threshold;
int cross_count;
int in_tune;
int average_val;
int pitch_diff;

int upper_bound;
int lower_bound;
int avg_cross;
int avg_counter;
int avg_upper;
int avg_lower;
float a, b;

int timer_divide;
int divide_by;

int string_select;
int select_pin_val;

int analogPin    = 0;   // analog to digital pin for signal input

int led_high     = 13;  // this LED will show user if a string's pitch is too high
int led_ok       = 12;  // this LED will show user if a string's pitch is correct
int led_low      = 11;  // this LED will show user if a string's pitch is too low

// these pins will light up to show which string the Atmega is comparing values for
int led_e4       = 10;
int led_b3       = 9;
int led_g3       = 8;
int led_d3       = 7;
int led_a2       = 6;
int led_e2       = 5;

int button_pin   = 3;    // input pin for user using a button to switch strings
int servoPin     = 2;    // control pin for servo motor
```

```
void setup()
{
  // Set up timer 1 to generate an interrupt every 1 microsecond
  TCCR1A = 0x00;
  TCCR1B = (_BV(WGM12) | _BV(CS12));
  OCR1A = .071;
  TIMSK1 = _BV(OCIE1A);

  x = 0;
  lastx = 0;
  y = 0;
  lasty = 0;

  timer = 0;
  cross_count = 0;
  avg_cross = 0;
  avg_counter = 0;
  string_select = 0;

  //Set the input and output pins
  pinMode(button_pin, INPUT);
  pinMode(servoPin, OUTPUT);
  pinMode(led_high, OUTPUT);
  pinMode(led_ok, OUTPUT);
  pinMode(led_low, OUTPUT);
  pinMode(led_e4, OUTPUT);
  pinMode(led_b3, OUTPUT);
  pinMode(led_g3, OUTPUT);
  pinMode(led_d3, OUTPUT);
  pinMode(led_a2, OUTPUT);
  pinMode(led_e2, OUTPUT);

  Serial.begin(9600);               // Opens serial port, sets data rate to 9600 bps
}

// Nothing is done in the Arduino loop, since timing is off.
void loop()
{
}

// Timer function running every microsecond
ISR(TIMER1_COMPA_vect)
{
    timer++;
    idle_timer++;

    // Read button press to determine which string is to be detected
    if (timer % 100 == 0)
    {
      select_pin_val = digitalRead(button_pin);
      if (select_pin_val == HIGH)
      {
         string_select = ((string_select + 1) % 6);
         Serial.print("string: ");
         Serial.println(string_select);
       }
    }


    // Depending on which string is selected, the proper variables are set
    switch (string_select)
```

```
{
 case 0:
    digitalWrite(led_e4, LOW);   // sets the proper LED on, all else off
    digitalWrite(led_b3, LOW);
    digitalWrite(led_g3, LOW);
    digitalWrite(led_d3, LOW);
    digitalWrite(led_a2, LOW);
    digitalWrite(led_e2, HIGH);
    a = 0.045;
    b = 0.9099;
    threshold = 150;
    upper_bound = 77;
    lower_bound = 33;
    avg_upper = 57;
    in_tune = 55;                    // This is the “in tune” average of cross counts for the string.
    avg_lower = 53;
    timer_divide= 2000;
    divide_by = 3;
    break;
 case 1:
    digitalWrite(led_e4, LOW);   // sets the proper LED on, all else off
    digitalWrite(led_b3, LOW);
    digitalWrite(led_g3, LOW);
    digitalWrite(led_d3, LOW);
    digitalWrite(led_a2, HIGH);
    digitalWrite(led_e2, LOW);
    a = 0.0592;
    b = 0.8816;
    threshold = 150;
    upper_bound = 88;
    lower_bound = 44;
    avg_upper = 67;
    in_tune = 65;                    // This is the “in tune” average of cross counts for the string.
    avg_lower = 63;
    timer_divide = 2000;
    divide_by = 3;
    break;
 case 2:
    digitalWrite(led_e4, LOW);   // sets the proper LED on, all else off
    digitalWrite(led_b3, LOW);
    digitalWrite(led_g3, LOW);
    digitalWrite(led_d3, HIGH);
    digitalWrite(led_a2, LOW);
    digitalWrite(led_e2, LOW);
    a = 0.0797;
    b = 0.8406;
    threshold = 150;
    upper_bound = 117;
    lower_bound = 63;
    avg_upper = 97;
    in_tune = 95;                    // This is the “in tune” average of cross counts for the string.
    avg_lower = 93;
    timer_divide = 2000;
    divide_by = 3;
    break;
 case 3:
    digitalWrite(led_e4, LOW);   // sets the proper LED on, all else off
    digitalWrite(led_b3, LOW);
    digitalWrite(led_g3, HIGH);
    digitalWrite(led_d3, LOW);
    digitalWrite(led_a2, LOW);
    digitalWrite(led_e2, LOW);
```

```
    a = 0.0730;
    b = 0.8541;
    threshold = 130;
    upper_bound = 50;
    lower_bound = 15;
    avg_upper = 29;
    in_tune = 27;                    // This is the “in tune” average of cross counts for the string.
    avg_lower = 26;
    timer_divide = 500;
    divide_by = 4;
    break;
  case 4:
    digitalWrite(led_e4, LOW);   // sets the proper LED on, all else off
    digitalWrite(led_b3, HIGH);
    digitalWrite(led_g3, LOW);
    digitalWrite(led_d3, LOW);
    digitalWrite(led_a2, LOW);
    digitalWrite(led_e2, LOW);
    a = 0.1270;
    b = 0.7459;
    threshold = 140;
    upper_bound = 50;
    lower_bound = 15;
    avg_upper = 35;
    in_tune      = 34;               // This is the “in tune” average of cross counts for the string.
    avg_lower = 33;
    timer_divide = 500;
    divide_by = 4;
    break;
  case 5:
    digitalWrite(led_e4, HIGH);   // sets the proper LED on, all else off
    digitalWrite(led_b3, LOW);
    digitalWrite(led_g3, LOW);
    digitalWrite(led_d3, LOW);
    digitalWrite(led_a2, LOW);
    digitalWrite(led_e2, LOW);
    a = 0.1648;
    b = 0.6705;
    threshold = 150;
    upper_bound = 60;
    lower_bound = 20;
    avg_upper = 47;
    in_tune      = 45;                // This is the “in tune” average of cross counts for the string.
    avg_lower = 43;
    timer_divide = 500;
    divide_by = 4;
    break;
}

check_crossings();

// After the string input has been idle for a while, we take the average of a number of cross counts that were in bound.
if (idle_timer == 10000)
{
  Serial.println("AVG AVG LOOK HERE AVG AVG");
  average_val = avg_cross / divide_by;
  Serial.println(average_val);

  // If else statements for tuner lights
  if ((average_val < avg_lower) && (average_val > 0))
  {
     // Turn off all string display lights to conserve power
```

```
  digitalWrite(led_e4, LOW);
  digitalWrite(led_b3, LOW);
  digitalWrite(led_g3, LOW);
  digitalWrite(led_d3, LOW);
  digitalWrite(led_a2, LOW);
  digitalWrite(led_e2, LOW);

  // Sets the proper tuning LED on, all else off
  digitalWrite(led_high, LOW);
  digitalWrite(led_ok, LOW);
  digitalWrite(led_low, HIGH);

  pitch_diff = in_tune - average_val;
  Serial.print("Pitch Difference Low: ");
  Serial.println(pitch_diff);

  // If the tuning is off by a questionably high amount, count it as an error in reading and do not turn the peg.
  // Otherwise tune the peg for a period of time. This time depends on how far off the read average is.
  if (pitch_diff < 20)
  for(long i = 0; i < pitch_diff * 36000; i++)
  {
    digitalWrite(servoPin, HIGH);   // start the pulse
    delayMicroseconds(15);  // pulse width
    digitalWrite(servoPin, LOW);    // stop the pulse
  }
}

// Don't turn the peg if the guitar is in tune.
else if ((average_val >= avg_lower && average_val <= avg_upper) || (average_val == 0))
{
  // Sets the proper tuning LED on, all else off
  digitalWrite(led_high, LOW);
  digitalWrite(led_ok, HIGH);
  digitalWrite(led_low, LOW);
}

else if (average_val > avg_upper)
{
  // Turn off all string display lights to conserve power
  digitalWrite(led_e4, LOW);
  digitalWrite(led_b3, LOW);
  digitalWrite(led_g3, LOW);
  digitalWrite(led_d3, LOW);
  digitalWrite(led_a2, LOW);
  digitalWrite(led_e2, LOW);

  // Sets the proper tuning LED on, all else off
  digitalWrite(led_high, HIGH);
  digitalWrite(led_ok, LOW);
  digitalWrite(led_low, LOW);

  pitch_diff = average_val - in_tune;
  Serial.print("Pitch Difference High: ");
  Serial.println(pitch_diff);

  // If the tuning is off by a questionably high amount, count it as an error in reading and do not turn the peg.
  // Otherwise tune the peg for a period of time. This time depends on how far off the read average is.
  if (pitch_diff < 20)
  for(long i = 0; i < pitch_diff * 270000; i++)
  {
    digitalWrite(servoPin, HIGH);   // start the pulse
    delayMicroseconds(2);  // pulse width
```

```
          digitalWrite(servoPin, LOW);    // stop the pulse
        }
       }

       // Reset all variables used for pitch detection
       cross_count = 0;
       avg_cross = 0;
       avg_counter = 0;
      }

      // We take the average of cross counts after the first value in bound. We dismiss the first value since it is usually inaccurate
      // for finding a good average.
      if (timer % timer_divide == 0){
       if (cross_count > lower_bound && cross_count < upper_bound) {
        if (avg_counter >= 1 && avg_counter < (divide_by + 1)) {
          avg_cross = avg_cross + cross_count;
          Serial.print("Runnin Avg cross sum: ");
          Serial.println(avg_cross);
        }
        avg_counter++;
       }
       Serial.print("cross_count: ");
       Serial.println(cross_count);
       cross_count = 0;
      }
}

void check_crossings()
{
   lastx = x;
   lasty = y;
   x = analogRead(analogPin);          // Read the input pin
   y = a * x + a * lastx + b * lasty;  // Apply Butterworth filter to eliminate high frequencies

   // If the string crosses it's set threshold, add it to the count. If there are no crossings, the idle timer will begin to run.
   if (lasty > threshold and y < threshold)
   {
    cross_count++;
    idle_timer = 0;
   }
}
```

Design Description

(1) Servo Motor - This motor will use the pitch detection code from the Arduino to wind the peg to its proper frequency.

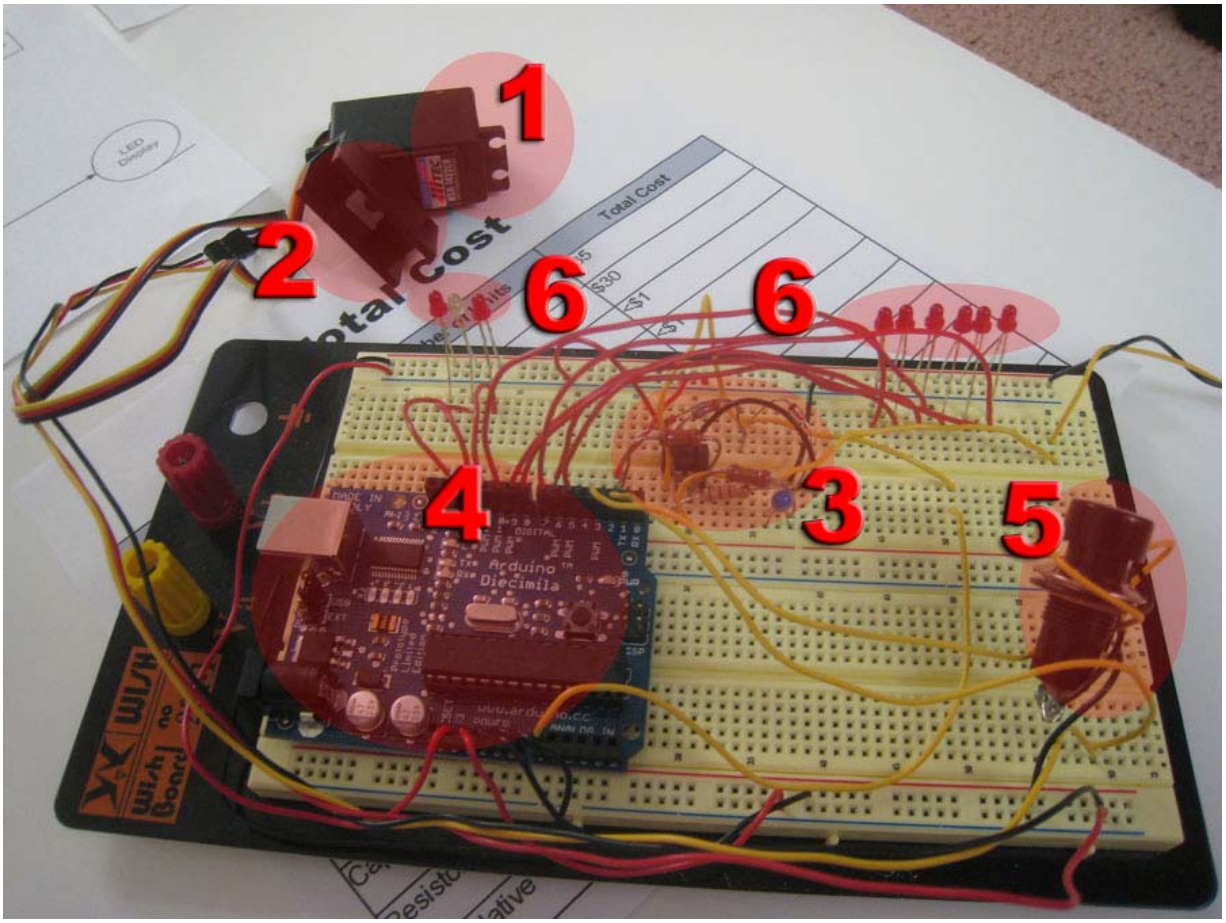
(2) Servo Motor Attachment - This is a plastic piece that attaches to the servo motor by fitting onto a smaller plastic piece already installed on the servo motor and fits on the other side to a guitar peg, so that the servo motor can effectively tune the guitar string.

(3) Op-Amp- The op-amp will amplify the guitar signal to a readable signal (around 5V) for the Arduino to determine the pitch.

(4) Pitch detector (Arduino) - The Arduino will read in the amplified signal from the op-amp circuit, then it will count how many times the signal crosses a certain threshold to determine the frequency. Depending on which string is selected, it will send a PWM signal to the servo motor to tune the peg to its proper pitch.

(5) Selector button - This will set the variables for the pitch detector, so that the user can choose which string he wants to tune. Each time the button is pressed, the next string will be chosen, shown in the LED Array.

(6) LED Array - These LED's will provide information to the user to show which string is being tuned (6 LEDs), and as to whether it is being tuned up or down by the Arduino (3 LEDs).



System Test Plan

Test Number	Description of Set-up	Input or Stimulus	Expected Behavior
1	Border conditions for each guitar string: We will input pitches that are unreasonably high or low compared to the desired pitch.	Signal from output jack of guitar with strings completely detuned or overtuned.	We expect the tuner to not run due to pitch thresholds, so that the user can tune the guitar himself to a reasonable range, to prevent possible errors from dealing with extreme pitches (such as a sudden noise when a string breaks).
2	Working conditions: Make sure that the signal being put into the tuner is only from the guitar, even with outside noise.	Signal from guitar output jack, outside noise.	We expect any outside noise not to interfere with the signal being inputted into the tuner.
3	Working conditions: Ensure that undesired electromagtnetic noise picked up by the guitar pickups does not effect the guitar tuner.	Signal from guitar output jack.	We expect minimal noise based on the noise-reducing circuit design of our op-amp.
4	Error case: The servo fails to turn.	Digital signal from Arduino to the servo.	We will add LEDs to show that the Arduino is in fact sending a signal to the servo in either direction, thus helping with troubleshooting.
5	Error case / Border Condition: If there is an error with the pitch detection algorithm, prevent the tuner from turning the servo too far in either direction	Signal from guitar output jack, or simulate an error test case in the pitch detection code.	We will implement a threshold safeguard in the code, thus preventing the tuner from breaking the string and possibly damaging the guitar.

Test Number	Description of Set-up	Input or Stimulus	Expected Behavior
6	Working conditions: Make sure the user is able to select the proper string to tune.	Selector switch for the user to use.	We will show which string is being tuned using representative LEDs, which will correctly show which note the tuner is using to compare pitches.

Project timeline

December 2007

- Project chosen
- Minimal subset determined
- Ordered various parts (servo motor, resistors, capacitors, USB oscilloscope)
- High-level Design determined

January 2008

- Modified servo motor to be continuous to allow maximal tuning
- Ordered every part needed
- Flowchart finalized
- Servo motor test control code written

February 2008

- Op-amp circuit design completed
- Op-amp circuit implemented
- Servo motor attachment design completed
- Servo motor attachment built (with help from Ted Edis in the Engineering Machine Shop)
- Pitch detection algorithm started with extensive testing using an oscilloscope

March 2008

- Determined the “in-pitch” values for each string and the ranges of the flat/sharp values
- Pitch detection algorithm finalized and tested on every string
- Integrated pitch detection algorithm with moving the servo motors
- Modified pitch detection code, since the power used for turning the motor affected the analog-to-digital values from the guitar.
- Added in a button for the string selecting
- Added in LEDs for indicators of in-tune, sharp (too high), or flat (too low)
- Added in LEDs for the string selecting from the button

Division of work

For the duration of the entire project, both team members worked together for the following tasks:

Purchase of equipment

High-level design

Testing non-continuous servo

Purchase of continuous motor

Coding of continuous motor

Op-amp circuit design research

Op-amp design implementation

Mechanical design and fabrication of attachment for servo motor to peg

Pitch detection code

Add LED's and user input switch

Cost

Part	Number of Units	Total Cost
Arduino Diecimila	1	$35
Servo Motor (Continuous)	1	$30
Servo Motor Attachment	1	<$1
Operational Amplifier (ECG891N)	1	<$1
LEDs	9	<$1
Pushbutton	1	<$1
USB Oscilloscope	1	$140
Capacitors (3.3uF)	1	<$1
Resistors (56k, 22k, 1k)	4	<$1
Cumulative Total Cost		**~$211**

Problems encountered and comments

Our first challenge was when we initially purchased a non-continuous servo motor to use for winding the pegs. This motor only had a range from 180 to -180 degrees, which would prevent the guitar from being tuned fully. We had the choice of either modifying the non-continuous motor, or purchase a continuous motor. We opted for the latter, and now can wind a peg continuously.

One technical challenge involving the signal input was that we could not get a signal to be picked up by a piezoelectric transducer. Our earlier outline of the project involved using a transducer to pick up the signal so that we could use any acoustic guitar, but out of 3 different piezos we purchased none could pick up a signal in a satisfactory manner. We ended up using alligator clips to pick up a signal directly from an instrument cable connected to an electric guitar.

We also ran into problems getting our signal to be amplified to be outputted by the op-amp. We initially attributed this to the op-amp itself, thinking by the op-amp. When using an op-amp we found in the lab, we were only able to get the gain voltage it was defective. However, after using several different op-amp makes and circuits we were clueless. We finally found out after research that the signal from the guitar was not grounded, thus we were only getting a voltage from the power source. We grounded the guitar signal by clamping an alligator clip to the ground shaft of the instrument cable, and thus were able to get a properly amplified signal from the op-amp.

Upon amplifying the signal and writing the code for pitch detection, we found that the native loop() function in the Arduino didn't provide us with consistent values for frequency, so we added in a timer function running every microsecond by utilizing one of the Atmega's internal timer. We used the AVR library for the Atmega168.

The servo motor was also a bit underpowered. We knew that the servo would have trouble turning the peg, but it was adequate for its purpose. We had to disable some display lights to ensure enough power was going to the servo.

We had problems determining the pitch of the higher strings, as we had to increase the sample rate to get good readings. Even so, the higher strings did not resonate as well as the lower strings, so we were unable to get enough cross count samples to determine a good average of cross counts in a period.

By the end of the project, we had created an automatic tuner that was mostly accurate for most strings and could tune a string by itself within one or two windings. We were also pleased to find that our two biggest hurdles (signal amplification and pitch detection) were solved with time to spare to allow us to focus on the other components (LEDs and servo control). We were also able to create the tuner using relatively inexpensive parts, as the tuner itself could be made for only around $70.

Project Presentation Poster