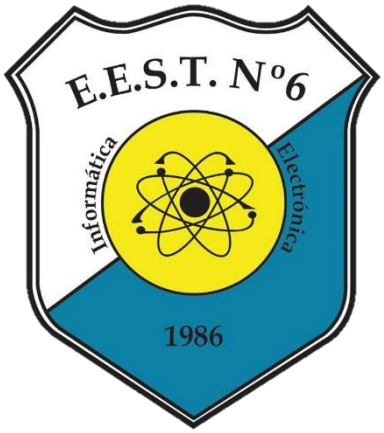


ESCUELA DE EDUCACIÓN SECUNDARIA TÉCNICA Nº 6

ELECTRONICA APLICADA

CICLO LECTIVO 2021

Curso: 7º	**División:** 1º	**Grupo:** B

Título: Transductores de medición

Escuela de Educación Secundaria Técnica Nº 6

Área: Electrónica
Materia: Sistemas de Control

Año: 7º **División:** 1º **Grupo:** B

Índice:

Trabajo Final, Afinador de guitarra automático

Objetivo:

La ejecución de este proyecto tiene como objetivo el desarrollo de un afinador de guitarra automático, es decir, capaz de realizar el ajuste necesario sobre las clavijas, todo en forma de lazo cerrado.

Funcionamiento:

La señal de la guitarra ingresa a un filtro pasa bandas con un AO (como filtro y también como adaptador de impedancia), y a dicha señal se le agrega un voltaje de offset de 2,5V para poder ser sampleada en el ADC de 10 bits del Arduino, esto debido a que la guitarra entrega un voltaje con componentes negativas y positivas.

En el programa se establece la frecuencia de muestro y se habilitan las interrupciones que leerán la señal de entrada. El código es capaz de discriminar si hubo coincidencia entre el tiempo registrado en el timer y la pendiente de la señal, o si no la hubo, y si la señal registrada posee una amplitud inferior o mayor a los preestablecidos en las constantes de tolerancias (véase el código).

Luego de la discriminación mencionada anteriormente, se calcula la frecuencia de la señal entrante y se evalúa a qué cuerda corresponde y si es inferior o superior a la frecuencia correcta.

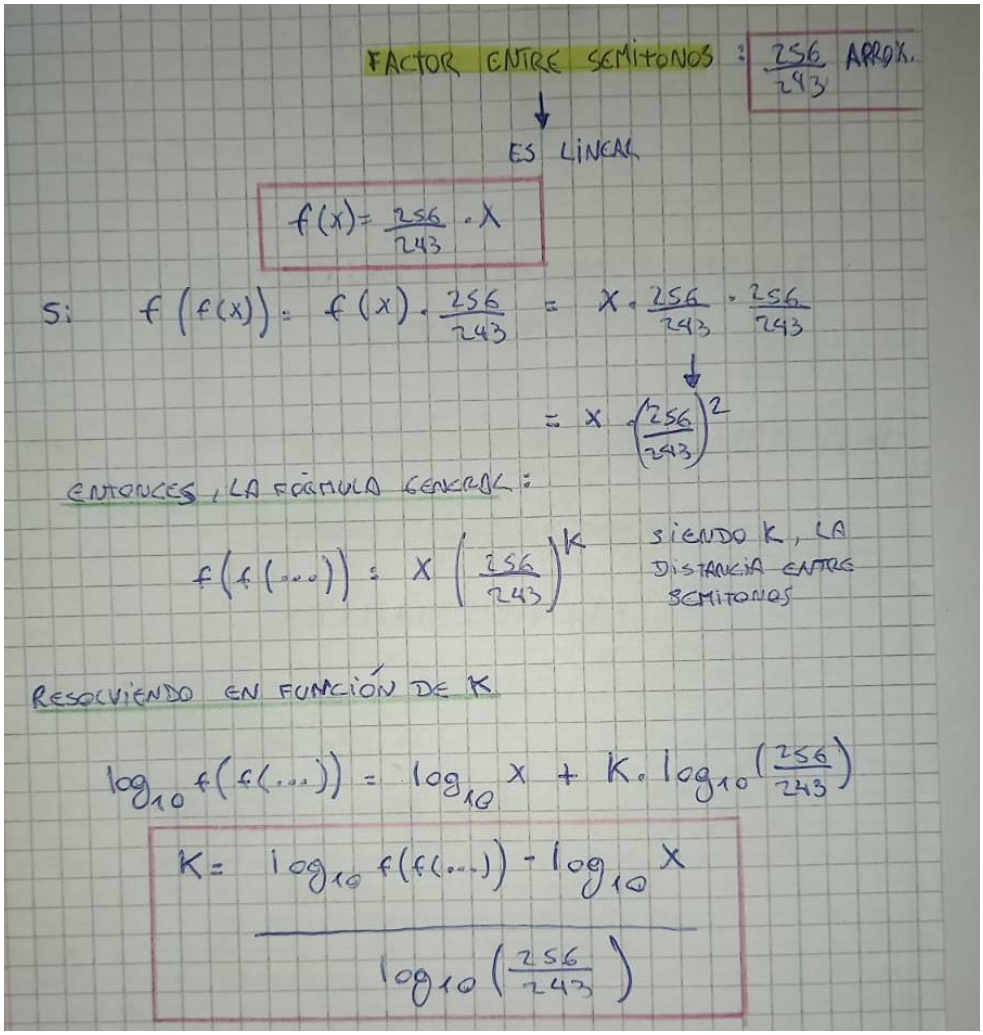
Frecuencias de cada una de las cuerdas:

E - 82.4 Hz
A - 110 Hz
D - 146.8 Hz
G - 196 Hz
B - 246.9 Hz
E - 329.6 Hz

En función de la frecuencia y la cuerda pulsada se encenderán una serie de LEDs indicadores. Seis de ellos corresponden a las cuerdas, y otros siete están destinados a visualizar si la frecuencia es superior, inferior o si coincide con el valor real que debería ser.

Para el control automático, se usó un servo motor de 48 pasos.

Ahora bien, para saber el movimiento necesario para ajustar la cuerda se utilizó el siguiente razonamiento. Entre cada semitono existe un valor lineal aproximado, 256/243, por lo tanto, se puede plantear lo siguiente "¿Cuántos semitonos hay entre la frecuencia en la que me encuentro y a la que quiero llega?". Como se puede apreciar, no se va a contar con valores enteros de semitonos, sino con coma flotante.



Con la fórmula del final se obtiene la cantidad de semitonos entre frecuencias (esta fórmula será ingresada al código).

Ahora queda saber cuántos pasos son necesarios para realizar un semitono (cabe mencionar, que este valor es muy probable que cambie entre distintas las 6 cuerdas, el calibre utilizado y el uso que tengan, sin embargo, con la fórmula anteriormente presentada en mayor o menor tiempo se podrá llegar a un valor óptimo dado que el proceso de medida y cálculo se hace de manera cíclica en forma de lazo cerrado). Con el afinador del teléfono se fue observando si se completaba un semitono a partir de un código de prueba que movía el motor la cantidad de pasos ingresados manualmente. Se pudo ver que para tensar la cuerda se necesitaban aproximadamente 270 pasos, y para destensar unos 20, también de manera aproximada. Esto permite establecer una regla de tres simple, es decir, "Si un semitono son X cantidad de pasos (270 o 20), K (semitonos dados por la fórmula), ¿Cuántos pasos serán?". Estas cuentas se pueden aplicar para cada una de las cuerdas, como se ve en el código, dentro de los voids calc () y calc2().

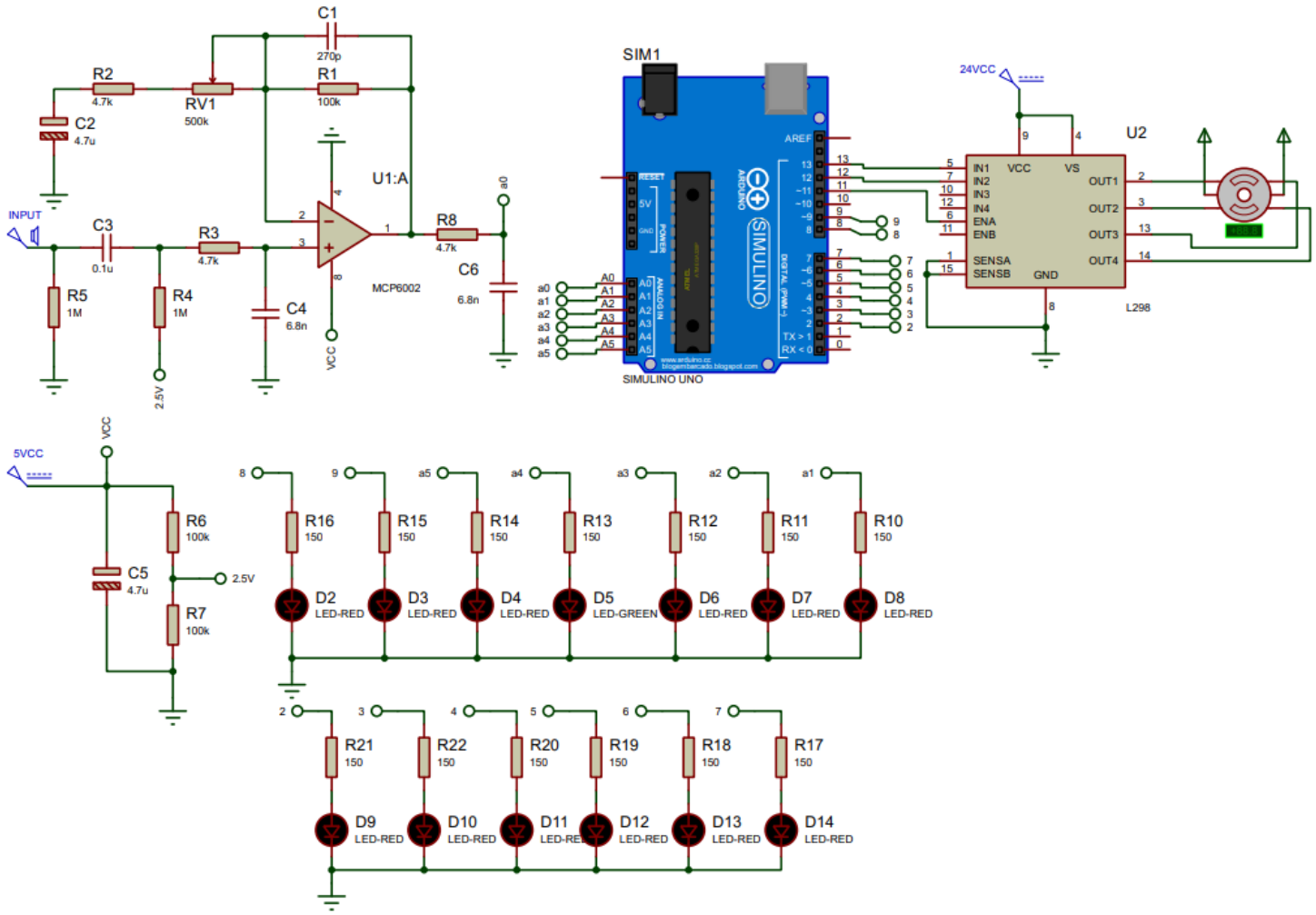
Escuela de Educación Secundaria Técnica Nº 6

Área: Electrónica

Materia: Sistemas de Control

Año: 7º **División:** 1º **Grupo:** B

Circuito:



Código:

```
#include<Stepper.h> //librería para el control del motor paso a paso
#define pasos 48 //definimos la cantidad de pasos que indica el
fabricante
Stepper stepper(pasos,13,11,12,10); // se seleccionan los pines de salida
para el control del motor
// motor variables
int valor; // esta variable sirve para activar el motor cuando se detecta
una cuerda en específico
float k=0;//cantidad de semitonos
float frec=0;
```

```
int pasos_finales;//esta variable garda la cantidad de pasos que se
deberán hacer luego de haber calculado la cantidad de semitonos que
faltan para alcanzar la nota correcta
//variables de almacenamiento para la detección de la frecuencia de la
señal entrante al ADC
byte newData = 0;
byte prevData = 0;
unsigned int time = 0;//toma y guarda valores en timer[]
int timer[10];//almacenamiento para eventos de tiempo
int slope[10];//almacenamiento de los valores de la pendiente de la señal
unsigned int totalTimer;//variable usada para calcular el periodo
unsigned int period;//guarda el periodo de la señal
byte index = 0;//indice de almacenamiento
float frequency;//guarda la frecuencia de la señal
int maxSlope = 0;//se usa para calcular la pendiente máxima como
activación del almacenamiento de datos
int newSlope;//guarda la pendiente recibida

//variables para decidir si hubo una coincidencia
byte noMatch = 0;//cuenta cuantas no-coincidencias han sucedido para
luego resetear arreglos que han almacenado muchos datos
byte slopeTol = 3;//tolerancia de la pendiente (valores reajustables)
int timerTol = 10;//tolerancia del timer (valores reajustables)

//variables para la detección de la amplitud de la señal
unsigned int ampTimer = 0;
byte maxAmp = 0;
byte checkMaxAmp;
byte ampThreshold = 30;//limite de saturación de la señal de entrada
(valores reajustables)

//variables para la afinación
int correctFrequency;//frecuencia correcta según la cuerda tocada

void setup(){
  stepper.setSpeed(120);// RPM del motor paso a paso

  //Pines de los LEDs
  pinMode(7,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(2,OUTPUT);
  pinMode(A3,OUTPUT);
  pinMode(A4,OUTPUT);
  pinMode(A5,OUTPUT);
  pinMode(A1,OUTPUT);
  pinMode(A2,OUTPUT);
  pinMode(8,OUTPUT);
```

```
  pinMode(9,OUTPUT);


  //Juego de luces previa a la etapa de afinación
  digitalWrite(7,1);
  digitalWrite(6,1);
  digitalWrite(5,1);
  digitalWrite(4,1);
  digitalWrite(3,1);
  digitalWrite(2,1);
  digitalWrite(8,1);
  analogWrite(A1,255);
  delay(500);
  digitalWrite(9,1);
  analogWrite(A2,255);
  delay(500);
  digitalWrite(A5,255);
  analogWrite(A3,255);
  delay(500);
  analogWrite(A4,255);
  delay(500);



  cli();//desconexión de las interrupciones

  //se setea la frecuencia de muestreo del pin analógico A0 en 38.5kHz

  //se limpian los regiustros ADCSRA y ADCSRB
  ADCSRA = 0;
  ADCSRB = 0;

  ADMUX |= (1 << REFS0); //se setea el voltaje de referencia
  ADMUX |= (1 << ADLAR); //acá se buscan leer los 8 bits superiores del
registro ADCH

  ADCSRA |= (1 << ADPS2) | (1 << ADPS0); //se setea el prescaler del ADC
en un valor de 32- 16mHz/32=500kHz
  ADCSRA |= (1 << ADATE); //se habilita el auto trigger
  ADCSRA |= (1 << ADIE); //habilita las interrupciones cuando las medidas
se han tomado
  ADCSRA |= (1 << ADEN); //habilita el ADC
  ADCSRA |= (1 << ADSC); //setea el ADC para la lectura de datos

  sei();//se habilitan las interrupciones
}

ISR(ADC_vect) {//cuando se detectó un valor
  prevData = newData;//guarda un valor previo
  newData = ADCH;//toma el nuevo valor dede A0
```

```
  if (prevData < 127 && newData >=127){//si el valor medido supera los
127, es decir 2,5V del offset
    newSlope = newData - prevData;//se calcula la pendiente
    if (abs(newSlope-maxSlope)<slopeTol){// si el valor de la pendiente
es inferior al seteado por la tolerancia
      //se guarda la nueva data y se reinicia el timer
      slope[index] = newSlope;
      timer[index] = time;
      time = 0;
      if (index == 0){//primer valor leído
        noMatch = 0;
        index++;//incrementa index
      }
      else if (abs(timer[0]-timer[index])<timerTol && abs(slope[0]-
newSlope)<slopeTol){//si hay una coincidencia entre el timer y la
pendiente leída
        totalTimer = 0;
        for (byte i=0;i<index;i++){
          totalTimer+=timer[i];
        }
        period = totalTimer;
        timer[0] = timer[index];
        slope[0] = slope[index];
        index = 1;//set index to 1
        noMatch = 0;
      }
      else{//si no hubo coincidencia
        index++;//incrementa index
        if (index > 9){
          reset();
        }
      }
    }
    else if (newSlope>maxSlope){//si la nueva pendiente es mayor a max
slope
      maxSlope = newSlope;
      time = 0;//resetea clock
      noMatch = 0;
      index = 0;//resetea index
    }
    else{//no hubo variaciones significativas en la pendiente
      noMatch++;//incrementa el contador no-match
      if (noMatch>9){
        reset();
      }
    }
  }

  time++;//incrementa el timer con una frecuencia de clock de 38.5kHz,
coincidente con la frecuencia de las muestras tomadas
```

```
  ampTimer++;//incrementa la amplitud del timer
  if (abs(127-ADCH)>maxAmp){
    maxAmp = abs(127-ADCH);
  }
  if (ampTimer==1000){
    ampTimer = 0;
    checkMaxAmp = maxAmp;
    maxAmp = 0;
  }

}

void reset(){//limpia algunas variables
  index = 0;//resetea index
  noMatch = 0;//resetea match couner
  maxSlope = 0;//resetea slope
}

//Se apagan 5 de los 6 LEDs según la cuerda que se halla pulsada
void otherLEDsOff(int LED1, int LED2,int LED3,int LED4,int LED5){
  digitalWrite(LED1,0);
  digitalWrite(LED2,0);
  digitalWrite(LED3,0);
  digitalWrite(LED4,0);
  digitalWrite(LED5,0);
}

//Determina qué cuerda fue pulsada y enciende el LED correspondiente a
esa cuerda
void stringCheck(){
  if(frequency>70&frequency<90){
    otherLEDsOff(2,3,5,6,7);
    digitalWrite(2,1);
    correctFrequency = 82.4;
    valor=HIGH;
  }
   else if(frequency>100&frequency<120){
    otherLEDsOff(2,3,4,5,6);
    digitalWrite(3,1);
    correctFrequency = 110;
    valor=HIGH;
  }
   else if(frequency>135&frequency<155){
    otherLEDsOff(2,3,4,6,7);
    digitalWrite(4,1);
    correctFrequency = 146.8;
    valor=HIGH;
  }
  else if(frequency>186&frequency<205){
```

```
    otherLEDsOff(2,3,5,6,7);
    digitalWrite(5,1);
    correctFrequency = 196;
    valor=HIGH;
  }
 else if(frequency>235&frequency<255){
    otherLEDsOff(2,4,5,6,7);
    digitalWrite(6,1);
    correctFrequency = 246.9;
    valor=HIGH;
  }
  else if(frequency>320&frequency<340){
    otherLEDsOff(3,4,5,6,7);
    digitalWrite(7,1);
    correctFrequency = 329.6;
    valor=HIGH;
  }else{valor=LOW;}
}
void calc(){//calcula la cantidad de semitonos y pasos que se deberán
hacer, desde una frecuencia más baja hacia una más alta
  frec= frequency;
 k=(log10(correctFrequency)-(log10(frec)))/(0.0226);//cálculo de la
distancia de semitonos entre frecuencias
 pasos_finales=(270*k);// 270 es un valor constante que representa 1
semitono POR DEBAJO, al cual se le aplica una regla de 3 simple para
ajustarse con la cantidad de pasos necesarios según lo inique k
 stepper.step(pasos_finales);
 delay(2000);
  }
void calc2(){//calcula la cantidad de semitonos y pasos que se deberán
hacer, desde una frecuencia más alta hacia una más baja
  frec= frequency;
 k=(log10(correctFrequency)-(log10(frec)))/(0.0226);//cálculo de la
distancia de semitonos entre frecuencias
 pasos_finales=(90*k);// 90 es un valor constante que representa 1
semitono POR ENCIMA, al cual se le aplica una regla de 3 simple para
ajustarse con la cantidad de pasos necesarios según lo inique k
 stepper.step(pasos_finales);
 delay(2000);
  }
//Compare the frequency input to the correct
//frequency and light up the appropriate LEDS
void frequencyCheck(){
  if(frequency>correctFrequency+1){
  analogWrite(A3,255);//Primer LED a la derecha del LED verde (en tono)
  if(valor==HIGH){
  calc2();
  }
  }
  if(frequency>correctFrequency+4){
```

```
  analogWrite(A2,255);//Segundo LED a la derecha del LED verde (en tono)
  if(valor==HIGH){
  calc2();
  }
  }
  if(frequency>correctFrequency+6){
  analogWrite(A1,255);//Tercer LED a la derecha del LED verde (en tono)
  if(valor==HIGH){
  calc2();
  }
  }
  if(frequency<correctFrequency-1){
  analogWrite(A5,255);//Primer LED a la izquierda del LED verde (en tono)
  if(valor==HIGH){
  calc();
  }
  }
  if(frequency<correctFrequency-4){
  digitalWrite(9,1);//Segundo LED a la izquierda del LED verde (en tono)
  if(valor==HIGH){
  calc();
  }
  }
  if(frequency<correctFrequency-6){
  digitalWrite(8,1);//Tercer LED a la izquierda del LED verde (en tono)
  if(valor==HIGH){
  calc();
  }
  }
  if(frequency>correctFrequency-1&frequency<correctFrequency+1){
  analogWrite(A4,255);//En tono
}
}

void allLEDsOff(){ //Función para el apagado de los LEDs
  digitalWrite(2,0);
  digitalWrite(3,0);
  digitalWrite(4,0);
  digitalWrite(5,0);
  digitalWrite(6,0);
  digitalWrite(7,0);
  digitalWrite(8,0);
  digitalWrite(9,0);
  analogWrite(A1,0);
  analogWrite(A2,0);
  analogWrite(A3,0);
  analogWrite(A4,0);
  analogWrite(A5,0);
}
```

```
void loop(){

  allLEDsOff();//Se apagan los LEDs

  if (checkMaxAmp>ampThreshold){//Se verifica que la señal de medida sea
superior al nivel seteado
    frequency = 38462/float(period);//Se calcula la frecuencia
  }
  stringCheck();// Se verifica la cuerda pulsada
  frequencyCheck();// Se verifica la frecuencia de la señal de entrada

  delay(100);}
```

Bibliografía:

- https://www.instructables.com/Arduino-Guitar-Tuner/
- https://www.instructables.com/Arduino-Frequency-Detection/