Universidad Nacional de Colombia
Facultad de ciencias - Departamento de física

Electrónica Digital

Informe de laboratorio
Afinador de guitarra eléctrica

Santiago Andrés Acosta Díaz saacostad@unal.edu.co

Erick Leonel Torres Otálora ertorreso@unal.edu.co

Resumen

El objetivo del proyecto final fue construir un circuito capaz de afinar una guitarra eléctrica. Para lograrlo, el arduino reside la señal de la salida auxiliar de un amplificador comercial, y la selección de cuerda de un conjunto de pulsadores. A partir de esto, arduino esta programado para realizar una transformada rápida de fourier sobre la señal, comparar su frecuencia principal con la frecuencia deseada para dicha cuerda y ordenar a un motor que accione la clavija de la guitarra. Cuando la frecuencia de la cuerda alcanza el valor deseado con un error menor a 0.8Hz, el motor deja de accionar la cuerda. Por lo anterior, se cumplieron todos los objetivos iniciales del proyecto.

Descripción del Circuito

En la figura 1 se muestra el esquema del circuito. Se observa en (1) el arduino. En este se reciben las siguientes entradas, en A5 llega la señal analogica del amplificador y por los pines digitales 10, 11 y 12 entra la señal de los pulsadores. Las salidas del arduino son las siguientes: por los pines digitales 4 al 9 sale la señal de cada uno de los 6 leds que indican la cuerda seleccionada, por los pines 1 y 2 sale la señal digital al controlador del motor. Por otro lado en (2) se encuentra el arreglo de pulsadores para la selección de cuerda. Los pulsadores izquierdo y derecho permiten navegar en ambos sentidos la selección de cuerda en pasos de uno por pulso. El pulsador central selecciona la cuerda. Por ultimo, en (3) se tiene el controlador del motor L-293D, este recibe dos señales analógicas del arduino y permite alimentar el motor con una fuente distinta al la del resto del circuito. En este caso se esta alimentando al motor con un potencial de 9V. El funcionamiento de este integrado es el siguiente. Si por los pines 10 y 15 detecta señales digitales iguales, los pines de salida 11 y 14 no tienen diferencia de potencial entre ellos, el motor se queda quieto. Si por las entradas llegan valores diferentes, el controlador pasara el voltaje de alimentación al motor entre los pines 11 y 14, si las entradas se invierten el controlador también invierte la polarización de las patas de salida. De este modo se hacer que gire el motor el cualquiera de los dos sentidos.

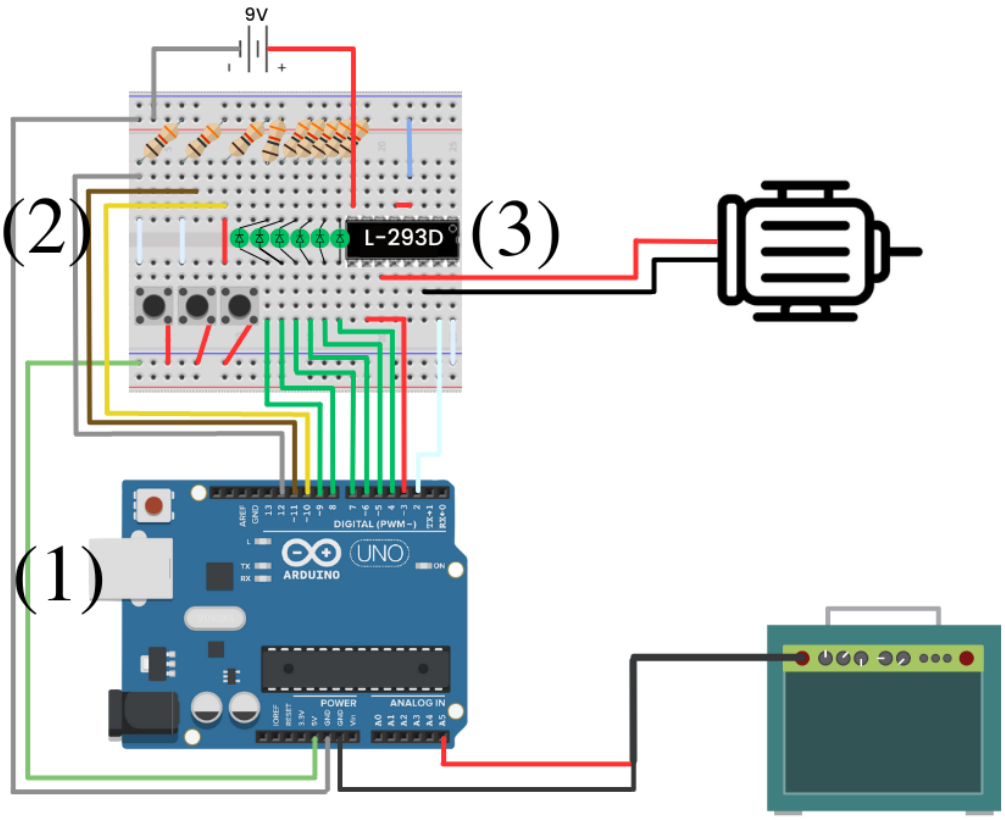

Figura 1: Diagrama del circuito

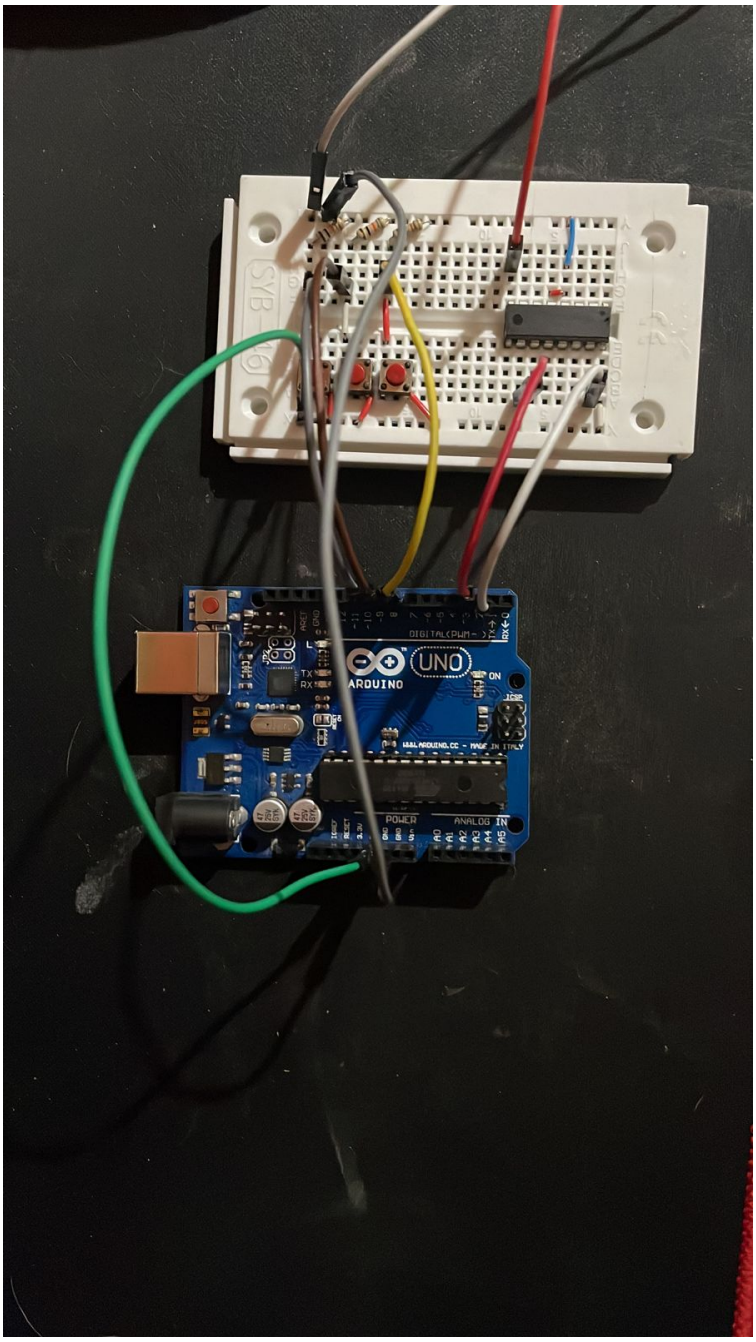

Figura 2: Foto del circuito

Código fuente

El código fue hecho en arduino, usando *ArduinoIDE* y consta de 3 secciones generales:

1. **Cargar instancias globales:** sección en donde se crean las variables globales necesarias para el funcionamiento del circuito.
2. **Funciones de arduino:** las funciones `setup` y `loop`.
3. **Funciones del proyecto:** funciones propias del proyecto.

Hay que tener en cuenta que el código real está comentado en cada línea de código, por lo que vale la pena revisarlo directamente en su repositorio de *GitHub*, el link es el siguiente:

Link al repositorio

Cargar instancias globales

```
#include "arduinoFFT.h"

#define SAMPLES 128
#define SAMPLING_FREQUENCY 3500

arduinoFFT FFT = arduinoFFT();

unsigned int sampling_period_us;
unsigned long microseconds;


double vReal[SAMPLES];
double vImag[SAMPLES];

const double tunning[6] = {330.0, 252.5, 199, 149.5, 110.5, 84.0};
short int string = 0;


double tune = 0.0;
short int state;


const short int pPin = 2;
const short int nPin = 3;


const int vol_th = 575;
const double tone_th = 0.5;

short int prev_switch = 0;
```

En donde, primero, se incluye la librería `arduinoFFT.h`, la encargada de realizar la tranformada de Furier. Se definen `SAMPLES` y `SAMPLING_FREQUENCY`, que son la cantidad de muestras para la transformada discreta y la frecuencia con las que se tomarán. Se crea el objeto `FFT = arduinoFFT()`, que guarda la información del muestreo y realiza la transformada. Y dos variables `sampling_period_us` y `microseconds`, la primera guarda cuánto tiempo se debe esperar antes de recoger otra muestra, según `SAMPLING_FREQUENCY`, y la segunda, tomará el tiempo de ejecución de cada muestreo.

Dos arreglos `vReal` y `vImag`, los cuales guardarán los datos del muestreo de la onda. En este caso, `vImag` es un arreglo lleno de `0.0`.

El arreglo `tunning` contendrá los valores de las frecuencias esperadas para cada cuerda, *string* la cuerda a afinar en cada momento, `tune` la frecuencia dominante encontrada por la *TTF* y `state` el estado de operación.

Las variables definidas abajo son variables para calibración y selección de pines.

Funciones de arduino

```
void setup() {
```

```
  // Para la TTF
  sampling_period_us = round(1000000*(1.0/SAMPLING_FREQUENCY));
  Serial.begin(115200);
  while(!Serial);

  // Serial.println("Ready");

  // Para seleccionar los pines
  pinMode(pPin, OUTPUT);
  pinMode(nPin, OUTPUT);

  // Se inicia en el primer estado 0
  state = 0;

  // Se seleccionan los pines de entrada
  // para la selección de cuerdas
  for (short int i = 10; i <= 12; i++)
  {
    pinMode(i, INPUT);
  }

  for (short int i = 4; i <= 9; i++)
  {
    pinMode(i, OUTPUT);
  }

}
```

En `setup`, se da el valor de `sampling_period_us`, y luego se establece la conexión serial entre el arduino y el computador. Se establecen los tipos de pines digitales y se inicia el estado en 0.

```
void loop()
{

  switch (state)
  {
    case 0:

      //Serial.println("Seleccionando cuerda");

      for (int i = 10; i <= 12; i++)
      {
        if (digitalRead(i) == HIGH)
        {

            if (prev_switch == i)
            {
              break;
            }
            else
            {

              prev_switch = i;

              switch (i)
```

```
            {
              case 10:

                if (string != 0){string--;} break;

              case 11:

                if (string != 5){string++;} break;

              case 12:

                state = 1;
                Serial.print("Cambio de estado: ");
                Serial.println(state);
                Serial.print("Cuerda elegida: ");
                Serial.println(string);

                powerOff_lights();
                return;

              default: void;
            }


          }

          delay(200);

      }
      else
      {
        prev_switch = 0;
      }

      set_light();
    }

    break;

  case 1:
    if (analogRead(A5) > vol_th)
    {
      tune = FindDominantFrequency();
      state = 2;

      Serial.print("Cambio de estado: ");
      Serial.println(state);
    }; break;

  case 2:
    if (move_string() == 1)
    {state = 0; string = 0;}
    else
    {state = 1;};
```

```
        Serial.print("Cambio de estado: ");
        Serial.println(state);
        break;

      default: void;    // por defecto, no se hace nada.
    }

}
```

La función `loop` funciona en base al estado `state` del proyecto.

estado 0:

Este estado se encarga de elegir la cuerda a afinar, usando los pines de entrada del protoboard. Tiene en cuenta el último pin activado, para evitar que se active varias veces la misma instrucción, y según el pin oprimido (el primero o el segundo), se va hacia atrás o hacia adelante en las cuerdas de la guitarra, seleccionando la cuerda con el tercer pin. Luego de seleccionar la cuerda, pasa al estado 1. A medida que se cambia la cuerda seleccionada, prende la led correspondiente a cada cuerda. Al salir de este estado, se apagan todas las led.

estado 1:

El estado uno se encarga de hacer la transformada de Furier. Éste espera hasta que el volumen de la guitarra pase cierto umbral, para evitar coger audios de ruido. Cuando el volumen pasa el umbral, llama a la función `FindDominantFrequency` y guarda su valor en `tune`, la cual se encarga de encontrar la frecuencia dominante en ese instante. Pasa al estado 2.

estado 2:

En este estado, se revisa primero que la frecuencia real esté dentro de cierto rango comparada con la frecuencia deseada (cosa que indica el retorno de la función `move_string`); si se está dentro del rango, significa que la cuerda está afinada, por lo que se devuelve al estado 0 para seleccionar nuevamente una cuerda.

En caso diferente, la función coge la diferencia entre la frecuencia real `tune` y la frecuencia deseada de `tunning`, y según ésto se mueve el motor. Luego, se devuelve al estado 2 para volver a obtener la frecuencia dominante después de haber accionado el motor.

Funciones del proyecto

```
double frec_it()
{
  for(int i=0; i<SAMPLES; i++)
    {
        microseconds = micros();
        vReal[i] = analogRead(A5);
        vImag[i] = 0;
        while(micros() < (microseconds + sampling_period_us));
    }

    /*FFT*/
    FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
    FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD);
    FFT.ComplexToMagnitude(vReal, vImag, SAMPLES);

    double peak = FFT.MajorPeak(vReal, SAMPLES, SAMPLING_FREQUENCY);
```

```
    return FFT.MajorPeak(vReal, SAMPLES, SAMPLING_FREQUENCY) * 0.990;
}
```

La función `frec_it` halla la frecuencia dominante en el instante que es llamada, haciendo uso del objeto `TTF` de la librería `arduinoTTF.h`.

```
double FindDominantFrequency()
{

  int counter = 0;

  double last_val = frec_it();

  while (counter < 3)
  {

    double inst_freq = frec_it();

    if (abs(last_val - inst_freq) < tone_th)
    {
      counter++;
      last_val = (last_val + inst_freq) / 2;
    }
    else
    {
      counter = 0;
      last_val = frec_it();
    }

  }

  return last_val;
}
```

La función `FindDominantFrequency` busca que el regreso de 3 transformadas de Furier seguidas esté dentro de un mismo rango, por lo que llama a la función `frec_it` para obtener la primera frecuencia dominante, e inicia un contador para asegurarse que las siguientes tengan un valor en ese rango. Cuando encuentra que es así, devuelve el valor del promedio de las 3 frecuencias dominantes obtenidas.

```
bool move_string()
{

  double diff = tune - tunning[string];


  if (abs(diff) > 50.0)
  {
    return 0;
  }

  if ( abs(diff) < tone_th )
  {
    return 1;
  }

  if (diff < 0)
```

```
  {
      digitalWrite(pPin, HIGH);

      delay(-1.0 * diff * 30.0);

      digitalWrite(pPin, LOW);
  }
  else
  {
    digitalWrite(nPin, HIGH);

      delay(1.0 * diff * 30.0);

      digitalWrite(nPin, LOW);
  }

  return 0;
}
```

La función `move_string` es la encargada de activar los pines que control del integrado *l293d*, el cual controla la salida al motor. Primero, revisa que la frecuencia real esté en un rango similar a la frecuencia deseada, si lo está (cuerda afinada), retorna `true` (1), por lo que el controlador de estados se devuelve al estado 0. Según el signo de la diferencia, el motor debe moverse a un lado o hacia al otro, así, se decide a qué dirección mover el motor, se activa el pin digital y se espera cierto tiempo, que varía según la diferencia entre frecuencias, para luego desactivar ese pin, dejando el motor quieto. En este caso, se retorna 0, por lo que el controlador de estados vuelve a buscar la frecuencia dominante para activar nuevamente el motor.

```
// Prende el led de la cuerda en selección
void set_light()
{
  for (short int i = 0; i < 6; i++)
  {
    if (i == (string))
    {
      digitalWrite(i + 4, HIGH);
    }
    else
    {
      digitalWrite(i + 4, LOW);
    }
  }
}


// Apaga todos los led (lo hice porque si no mis ojos se quemaban)
void powerOff_lights()
{
  for (short int i =4; i <= 9; i++)
  {
    digitalWrite(i, LOW);
  }
}
```

Por último, estas funciones se encargan de encender la led correspondiente a cada cuerda en la selección de cuerdas. Al momento de salir de este estado, se apagan todas las luces, y vuelve a prenderse el primer led,

indicando que la cuerda se afinó correctamente y que se puede volver a elegir un estado.

Dificultades y sugerencias

Elegir el motor fue la primera dificultad encontrada, puesto que el proyecto al necesitar un torque considerable, necesitaba de un motor especial capaz de ésto. Finalmente se decidió utilizar un motor-reductor simple, pues este no necesita de controladores para operar y ofrece el torque necesitado.

El siguiente problema fue cómo controlar la dirección del motor, y para esto, buscando en internet, se encontró un controlador que ofrece solución al problema, el integrado *l293d*.

El último problema fue la entrada de la señal de audio, puesto que los pines analógicos de arduino sólo reciben voltages de 0v a 5v, mientras que la señal de audio puede tomar valores negativos y valores muy pequeños, que arduino no interpretaría correctamente. Para esto, se decidió usar la un amplificador de guitarra y mandar su señal de salida al arduino. El ampificador usado fue un *Orange Crush Mini*, cuya señal de salida se encuentra directamente centrada en 2.5v, por lo que la señal que recibe el arduino de este amplificador es apropiada.

Sugerencias

Las cosas a mejorar serían la portabilidad del dispositivo, puesto que es muy débil. Una interfaz visual (sea con leds o con matrices led) que permita ver el estado y demás información del dispositivo, puesto que a partir de la selección de la cuerda (si no se están viendo los mensajes en el computador) la información interna del dispositivo queda oculta al usuario. Por último, mayor precisión en el movimiento del motor, ya que cuando se consiguen frecuencias muy parecidas a la esperada, se mueve muy poco y tarda relativamente mucho en llegar a la frecuencia correcta, cosa que se puede conseguir usando un motor-reductor a paso, por ejemplo.

Referencias

1. GitHub del proyecto arduinoTTF.
2. I built a guitar tuner with an Arduino.