Proyecto Arduino

Teclado MIDI con Arduino Leonardo Pro Micro

—

Manuel Clavero García

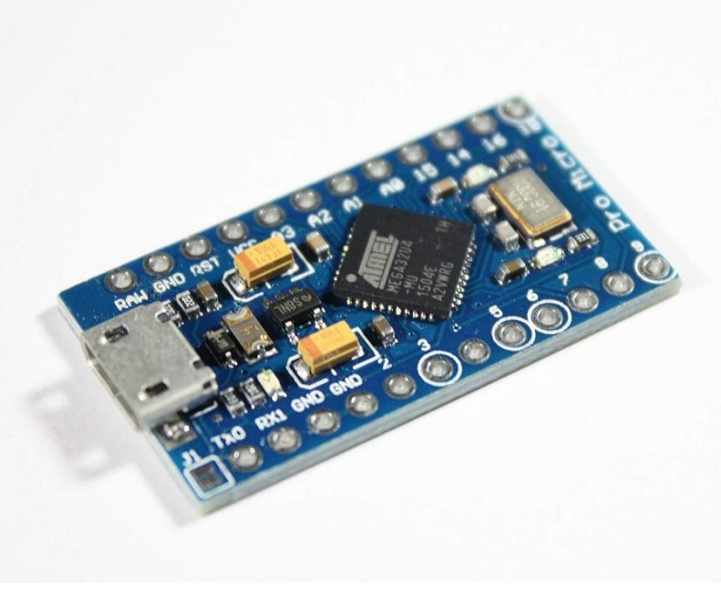
—

Sistemas Digitales

Introducción

Se ha construido desde cero un controlador MIDI utilizando un chip Arduino Leonardo Pro Micro, con el que se envían mediante MIDI la información de 12 notas musicales: la escala desde Do hasta Si incluyendo los sostenidos/bemoles.

Además, cuenta con dos potenciómetros y un sensor de distancia/ultrasonidos, que envían una señal de control, permitiendo modificar cualquier parámetro dentro del programa utilizado.



EL PROCESO

Planteamiento

En primer lugar, hay que escribir un código que permita, de forma simultánea, leer los 12 pulsadores usados como teclas de piano y gestione los potenciómetros y el sensor de distancia.

Lo primero a tener en cuenta es que las notas se envíen durante todo el tiempo que se mantienen presionados los botones, y dejen de enviarse cuando se suelten. Además, tiene que ser capaz de detectar más de un pulsador presionado a la vez.

El segundo problema a estudiar es el sensor de distancia. Al igual que los potenciómetros, hay que leer el valor de entrada, esté en el rango que esté, y mapearlo o normalizarlo entre 0 y 127, valor que se enviará por MIDI y que interpretará el programa que reciba la información.

La idea del sensor de distancia está inspirada en una versión muy básica del Theremin, instrumento que se controla moviendo las manos alrededor de dos varillas metálicas. En este proyecto hace la misma función que un potenciómetro, pudiéndose asignar a cualquier parámetro del sonido.

Como se está construyendo un dispositivo para usarse en música, las pruebas se harán con el programa Ableton Live 11 [1], pero funcionará con cualquier DAW.

Programación

Para el código, se han tomado elementos de diferentes fuentes [1] [2]. Para enviar instrucciones MIDI se emplea la librería “MIDIUSB.h”. Con esta librería, se definirán las funciones para encender y apagar las notas, así como para mandar una señal de control. Este código se añade al final, después del loop, y se puede ver en la figura 1.

A continuación, se define una función para gestionar sensor de ultrasonidos HC-SR04. Esta nos devuelve la distancia en centímetros. El funcionamiento es sencillo, se emite un pulso de 10 microsegundos por el puerto trigger y se mide cuánto tarda en recibirlo el puerto echo. El sensor manda una señal infrarroja (unos 40kHz) que rebota en el objeto (en nuestro caso la mano) y vuelve. Como la velocidad del sonido es de unos 343 metros/s, su inversa es 0,00291 s/m, o 29,1 microsegundos/cm. Como la distancia recorrida es de ida y vuelta, basta por multiplicar por 2 y redondear al entero más cercano: 58. El tiempo que tarda medido en microsegundos entre 58 nos da la distancia aproximada en centímetros.

Estos sensores no son perfectos, les influye la temperatura ambiente, la humedad y los materiales en los que reflejan, pero a cambio son baratos y efectivos hasta un poco

más de 3 metros. Aquí se usarán en un rango entre 10 y 40 cm. El código en la figura 2.

```
// ENCENDER NOTA MIDI
void noteOn(byte channel, byte pitch, byte velocity) {
midiEventPacket_t noteOn = {0x09, 0x90 | channel, pitch, velocity};
MidiUSB.sendMIDI(noteOn);
}
// APAGAR NOTA MIDI
void noteOff(byte channel, byte pitch, byte velocity) {
midiEventPacket_t noteOff = {0x08, 0x80 | channel, pitch, velocity};
MidiUSB.sendMIDI(noteOff);
}
// ENVIAR CAMBIO DE CONTROL
void controlChange(byte channel, byte control, byte value) {
midiEventPacket_t event = {0x0B, 0xB0 | channel, control, value};
MidiUSB.sendMIDI(event);
}
```

Figura 1

```
// MEDIR DISTANCIA AL SENSOR (cm)
int medir(){
 int dist;
 //digitalWrite(trigger, LOW);  //para generar un pulso limpio ponemos a LOW 4us
 //delayMicroseconds(2);
 digitalWrite(trigger, HIGH);          // Enviamos pulso de 10 microsegundos
 delayMicroseconds(10);
 digitalWrite(trigger, LOW);
 tiempoRespuesta = pulseIn(echo, HIGH);  // Y esperamos pulso de vuelta
 dist = tiempoRespuesta/58;              // Calculo de distancia en cm
 return dist;
}
```

Figura 2

En la cabecera se añaden las librerías y se definen todas las variables que se van a utilizar. Se han definido dos variables para el número de teclas/pulsadores y otro para el número de potenciómetros. Puede modificarse y tener más de ambos sin casi necesidad de modificaciones, pero en el chip que estamos usando solo tenemos 2 pines libres, así que habría que buscar otras alternativas como multiplexar o cambiar de micro.

En el vector “nota” se definen las notas desde Do hasta Si de la escala central del piano, donde el Do se corresponde al número 48, y un incremento de 1 es equivalente a un semitono. En pines_botones se define dónde se van a conectar los pulsadores. Las pares de variables valorLeido/Anterior y analogLeido/Anterior se usarán para gestionar las pulsaciones y los potenciómetros. También se define un filtro de mediana móvil, los pines del sensor infrarrojo y se inicializan a cero las variables necesarias. Código en la figura 3.

```
#include "MedianFilterLib.h"
#include "MIDIUSB.h"

const int numBotones=12;
const int numPotes=2;

int nota[]={48,49,50,51,52,53,54,55,56,57,58,59}; // Array con las 12 notas
int pines_botones[]={0,2,3,4,5,6,7,8,9,10,1, 19};
int valorLeido[numBotones];    // Array con los valores leidos de cada botón
int valorAnterior[numBotones]; // Array con el valor anterior leido

int analogLeido[numPotes];     // Array con los valores leidos de cada Potenciómetro
int analogAnterior[numPotes];  // Array con el valor anterior leido Potenciómetros
MedianFilter<int> medianFilter(20); // Inicia el filtro de mediana con una ventana de 20

int echo = 20;    // Pin para echo
int trigger = 21; // Pin para trigger

int control = 0;
int tiempoRespuesta;
int distancia=0;
int distanciaAnterior=0;
```

Figura 3

En el apartado de setup, presente en la figura 4. definimos los pines como entrada o salida, digitales o analógicas. También mandamos por MIDI todas las notas apagadas.

```
void setup() {
 //Serial.begin(9600);
 for(int i=0; i<numBotones; i++){
   pinMode(pines_botones[i], INPUT_PULLUP);// Recorremos cada pin y lo ponemos
como entrada
   valorAnterior[i]=1;       // Inicializamos el valor anterior a 1 (no pulsado)
 }
  pinMode(echo, INPUT_PULLUP);
  pinMode(trigger, OUTPUT);

 for(int i=0; i<=numBotones-1; i++){
   midiEventPacket_t noteOff = {0x08, 0x80 | 1, nota[i], 64};
   MidiUSB.sendMIDI(noteOff);
   MidiUSB.flush();
 }
}
```

Figura 4

Por último, hay que programar el loop, que se repetirá de forma indefinida y que debe gestionar pulsadores, potenciómetros y sensor infrarrojo. Para ello definimos tres bucles *for* diferentes, uno para cada cosa, como se puede observar en la figura 5.

Aparecen comentadas, tanto en el setup como en el loop, líneas de Serial.println. Esot sirve para ver por el monitor serie si todo funciona correctamente y no hay valores extraños o pulsadores que fallen.

```
void loop() {

 // Escaneo de PULSADORES
 for(int i=0; i<numBotones; i++){
  int pin = pines_botones[i];
  valorLeido[i] = digitalRead(pin);  // Leer pulsador
  if((valorLeido[i]==0) && (valorLeido[i]!=valorAnterior[i])){ // Solo enviamos cuando ha existido un cambio
   noteOn(1, nota[i], 64);                  // Activa nota correspondiente a cada pin
   MidiUSB.flush();
   //Serial.println("Se ha activado el pin");
   //Serial.println(pin);
  }
  else if ((valorLeido[i]==1) && (valorLeido[i]!=valorAnterior[i])){
   noteOff(1, nota[i], 64);                 // Desactiva nota correspondiente a cada pin
   MidiUSB.flush();
  }
  valorAnterior[i]=valorLeido[i];          // El valor leido pasa a ser valorAnterior para el proximo ciclo
 }

 // SENSOR DISTANCIA
 for(int i=0; i<20; i++){              // Se toman 20 muestras de la lectura de la distancia
  medianFilter.AddValue(medir());          // Se agregan los valores analogicos al filtro de mediana móvil
 }
 distancia = medianFilter.GetFiltered();    //se filtra con la mediana
                                            //discrimina para evitar errores
 if(distancia>=10 && distancia<=40 && distancia != distanciaAnterior){
  int control = map(distancia,10,40,0,127);        //escala el valor a un entero entre 0 y 127
  //Serial.println("Distancia:");
  //Serial.println(control);
  controlChange(0, 1, control); //usa la función de enviar cambios de control
  }
 MidiUSB.flush(); //Refresca comunicacion USB HID MIDI
 distanciaAnterior=distancia;

 // POTENCIÓMETROS
 // Potenciómetros conectados a partir del pin 18
 for(int i=0; i<numPotes; i++){
  // Se toman 20 muestras consecutivas de cada puerto analogico
  for(int j=0;j<20;j++){            // Se agregan los valores analogicos al filtro de mediana movil
   medianFilter.AddValue(analogRead(14+i));
  }
  analogLeido[i]= map(medianFilter.GetFiltered(), 0, 1023, 127,0); // El valor filtrado se acondiciona
  // Verificamos si hay cambio en el potenciometro
  if(analogLeido[i]!=analogAnterior[i]){
   controlChange(0, i+2, analogLeido[i]); // Envía el valor del potenciómetro que ha cambiado de estado
   //Serial.println(analogLeido[i]);
   MidiUSB.flush();
  }
  analogAnterior[i]=analogLeido[i]; // El valor analogLeido pasa a ser el analogAnterior para el próximo ciclo
 }
}
```

Figura 5

De manera resumida, se va a explicar el funcionamiento de los tres bucles. En primer lugar, para la gestión de los botones hay que asegurar que solo se manden instrucciones MIDI cuando cambia el valor de la entrada. El bucle for recorre todos los pines y, si hay un cambio, comprueba si es de encendido a apagado o de apagado a encendido. Consecuentemente, manda la instrucción de noteOn o note Off de la nota correspondiente, y actualiza el elemento en el vector valorAnterior.

Para el sensor de distancia, se realizan 20 mediciones llamando a la función *medir* y se añaden al filtro de mediana móvil (de esta manera se discriminan los errores de medición). Si el valor de distancia está dentro del rango elegido de 10 a 40 cm y es diferente a la distancia anterior, se mapea entre 0 y 127 y se envía. El valor de distancia anterior se actualiza.

Con los potenciómetros, conectados en los pines 14 y 15, se comprueban las entradas analógicas, se toman 20 medidas y se calcula la mediana, al igual que con el sensor de ultrasonidos. Este valor que está entre 0 y 1023 se mapea/normaliza entre 0 y 127. Si es distinto del valor anterior, se envía el nuevo y se actualiza el vector analogAnterior.

Montaje

Para el montaje se han utilizado los siguientes elementos:

- Arduino Leonardo Pro Micro
- 2 Potenciómetros lineales 5K
- Sensor de ultrasonidos HC-SR04
- 12 micropulsadores
- Cable micro USB
- Piezas impresas en 3D
- Mini protoboard
- Cables/jumpers
- Soldador de estaño
- Pistola de silicona

Una vez comprobado el funcionamiento de los 3 elementos principales por separado, había que diseñar una carcasa en la que montar el teclado midi. Como base se ha usado un modelo 3d de Thingiverse [2] y se ha editado en un programa de diseño 3d muy sencillo, Tinkercad [3]. La pieza de la tapa quedó como se ve en la figura 6. La base y los laterales no han sufrido modificaciones. Para segmentar las piezas y poder imprimirlas se ha usado el software Ultimaker Cura [4]. Se puede observar la interfaz en la figura 7.

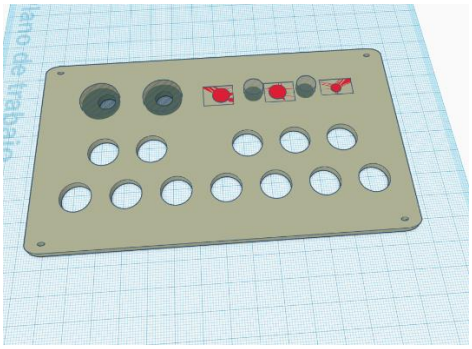


Figura 6. Modificaciones en Tinkercad

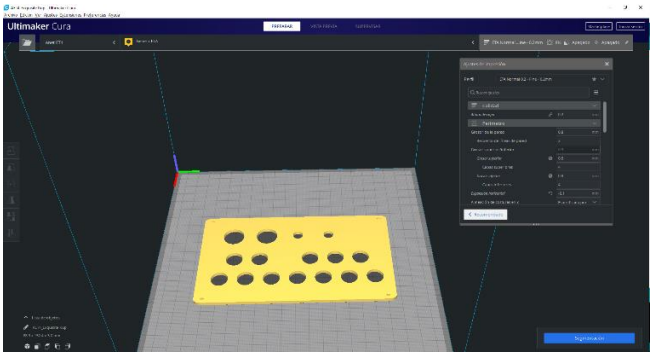


Figura 7. Pieza en Ultimaker Cura

Para la “cubierta” de los pulsadores se ha usado otro modelo de Thingiverse [5], que ha habido que redimensionar para ajustarlo a la caja. Este modelo de botón consta de 3 piezas: la cubierta, el pulsador y el soporte, que se pueden ver por separado en las figura 8 y 9. Para la similitud con un piano, se han impreso 7 en color blanco y 5 en negro.

Figura 8. Partes del botón impresas

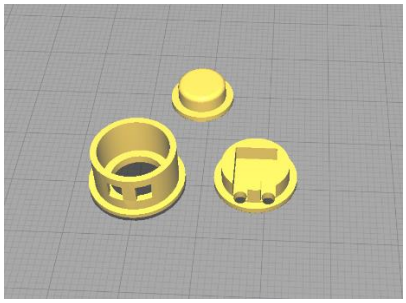


Figura 9. Modelos 3D del botón

Para hacer las piezas se ha usado una Anet ET4, impresora 3D FDM, es decir, por extrusión de material. El filamento utilizado ha sido PLA, en colores blanco, negro y plateado. La cubierta de los potenciómetros es el único elemento a la vista que no ha sido impreso.

Pruebas

Para comprobar el correcto funcionamiento se ha conectado el teclado MIDI al programa Ableton Live. Escogiendo cualquier plugin MIDI, el programa reconoce sin problemas el controlador (drivers instalados y configurado), que envía las notas desde Do a Si e la escala central del piano.

Si se quiere usar los potenciómetros y el sensor infrarrojo se tiene que activar el mapeo MIDI y seleccionar el parámetro que se quiere controlar. Al tener 3 elementos, podemos controlar 3 efectos a la vez.

Como se busca el efecto comentado del “Theremin”, se puede asociar a los efectos de “Vibrato”, por ejemplo, con lo que da la sensación de que el sonido vibra con el movimiento de la mano. En las figuras 10 y 11 se puede ver la interfaz de la versión 11 de Ableton Live, y cómo se controlan los efectos del sonido “Organ Transistor Series”. El sensor se asigna al efecto *Vibrato*, mientras que los potenciómetros controlan *Tone* y *Space*.

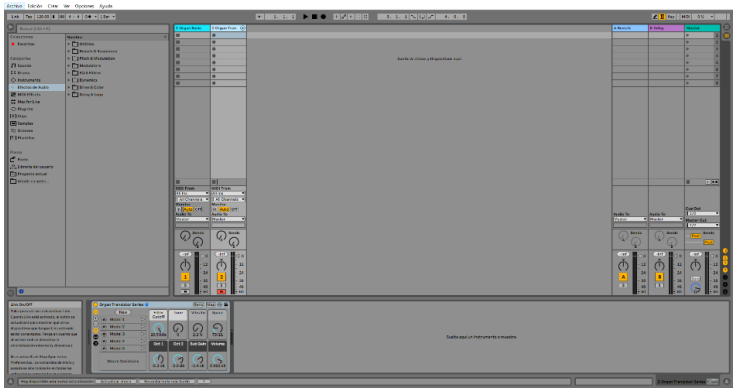


Figura 10. Interfaz Ableton Live 11

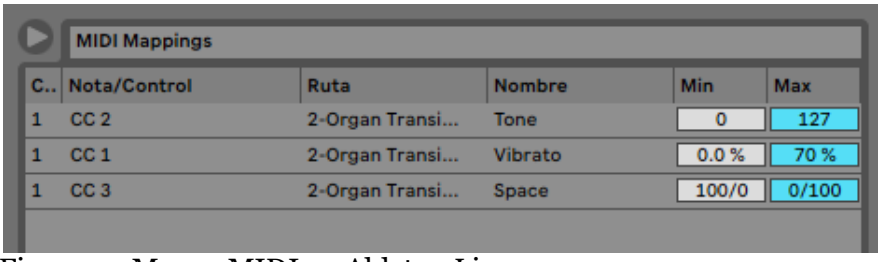


Figura 11. Mapeo MIDI en Ableton Live 11

Conclusión y futuros desarrollos

La elección de este proyecto se debe a la aplicación directa que tiene en música y producción musical. Es cierto que los botones no se acercan siquiera a la precisión de unas teclas que emulen un piano, y mucho menos hablar de sensibilidad, pero fusionarlo con 2 potenciómetros y la idea original del sensor de ultrasonidos como una especie de Theremin hacen que sea un proyecto relativamente novedoso y con posibilidad de usarlo en un entorno real de creación musical.

Posibles mejoras serían añadir más botones o potenciómetros, para lo que se necesitaría utilizar multiplexores... aunque parece más llamativo intentar registrar la sensibilidad, o añadir un botón que transporte la escala por octavas hacia arriba o hacia abajo.

Por mi parte, es un aparato que voy a utilizar, no sé si más por lo llamativo de controlarlo con la mano o porque realmente es una aplicación útil. De todas maneras, ha sido muy entretenido fusionar programación e impresión 3D para poder presentar un proyecto hecho de principio a fin por mí con cosas que tenía por casa.

Referencias

[1] aquilesvaesa
[2] YouTube
[3] Ableton Live
[4] Tapa superior, thingiverse
[5] Tinkercad
[6] Ultimaker Cura
[7] Botones, Thingiverse