UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Tenori-ON virtu-AL

Apellidos, nombre	**Agustí i Melchor, Manuel** (magusti@disca.upv.es)
Departamento	**Departamento de Informática de Sistemas y Computadores**
Centro	**Escola Tècnica Superior d'Enginyeria Informàtica** Universitat Politècnica de València

1 Resumen de las ideas clave

Viendo el *Doodle*[1] de homenaje a *Johann Sebastian Bach* estábamos disfrutando con la interacción que proporciona permitiendo la composición musical de una manera muy visual: se van escribiendo las notas directamente en el pentagrama, al tiempo que se oyen sonar. Además, al acabar, la aplicación tiene una opción para "armonizar" la composición con las del maestro *Bach* e incluso versionearla al ritmo de *Rock'nRoll*. En esta ambientación, cambian también los músicos y los instrumentos con que se interpreta la melodía, véanse las imágenes de la izquierda de la Figura 1. Al ver el músico de la derecha (lo vemos ampliado en la imagen de la derecha de la Figura 1), reparamos en que utiliza un sintetizador y un instrumento que recuerda al *TENORION* ... y no se pudo evitar, explicando lo que era empezamos a pensar en cómo hacer uno. Bueno, pues aquí dejamos las ideas y algo de código, a ver si quieres unirte a la diversión y convertirte en un *lutier*[2] virtual, con perdón de los artesanos del mundo real.

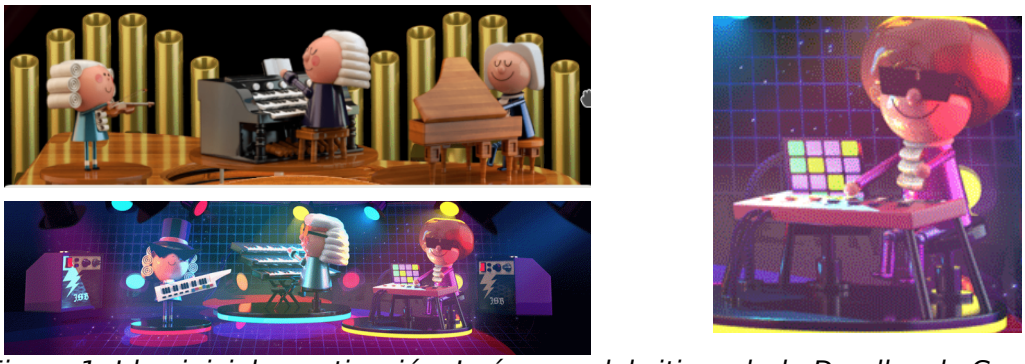


Figura 1: Idea inicial y motivación. Imágenes del sitio web de Doodles de Google.

No prometemos un instrumento con toda la funcionalidad, ni con el mismo modo de funcionamiento del real, pero sí que **mostraremos un ejemplo de diseño** que, utilizando una cámara web y unos gráficos sencillos, permitirá experimentar con estrategias básicas de visión por computador y síntesis de sonido, para proponer un divertimento basado en el *TENORI-ON* real, en forma virtual. Esperamos que te animes a probar, te contaremos cómo se ha hecho sobre *Ubuntu Linux 18.04*, con *OpenCV* y *OpenAL*. Es completamente portable a otras plataformas. Cuento con que el lector está familiarizado con estas librerías a un nivel básico para entender los ejemplos de código.

2 Objetivos

Una vez que el lector haya leído y explorado con cierto detenimiento lo que se presenta este documento, será capaz de:

- Implementar una estrategia de detección de movimiento, utilizando una sencilla cámara web, para determinar el punto de interacción del usuario con el instrumento virtual.
- Utilizar una sencilla visualización para darle parecido al interfaz real.
- Implementar un banco de sonidos para darle similitud sonora con el instrumento real.

---

1 Disponible en la URL <https://www.google.com/doodles/celebrating-johann-sebastian-bach>.

2 La RAE lo escribe así <http://lema.rae.es/dpd/srv/search?key=lutier>.

No estamos interesados en sustituir al *TENORION* real, sino en explorar la característica de este instrumento de aunar sonido y espectáculo al ofrecer una visualización de la música. Como dice la descripción del TNR-i[3], la versión para iOS del fabricante del TENORION: "Incluso si no sabes mucho de música, puedes crear música [...] como si fueras poniendo las notas en el espacio". Bueno, si es música o no es otro tema; pero lo cierto es que la música que ves "dibujada" en la interfaz, es la que es interpretada y las notas las "pondremos con un poco de diversión" no con el ratón o el teclado, sino agitando las manos.

3 Introducción

En este apartado comentaremos cómo funciona este instrumento, que no somos los primeros en montar una versión del TENORION y lo complejo que es para entender las restricciones que nos hemos impuesto al desarrollar la idea. Después, prometemos que enseñamos el código.

3.1 El instrumento real

El *TENORION* (o *TENORI-ON*) es (más detalles en [1], [2] y [3]) un instrumento musical que cuenta con una interfaz de luz y sonido. Es el resultado de la colaboración entre Yamaha y Toshio Iwai. Este es [4] quien ideó el *Electroplankton* para *Nintendo DS*, un juego de creación musical con el que hacer sonido y animar el contenido de la pantalla con toda la interactividad que permite la consola, ¡hasta soplando!.

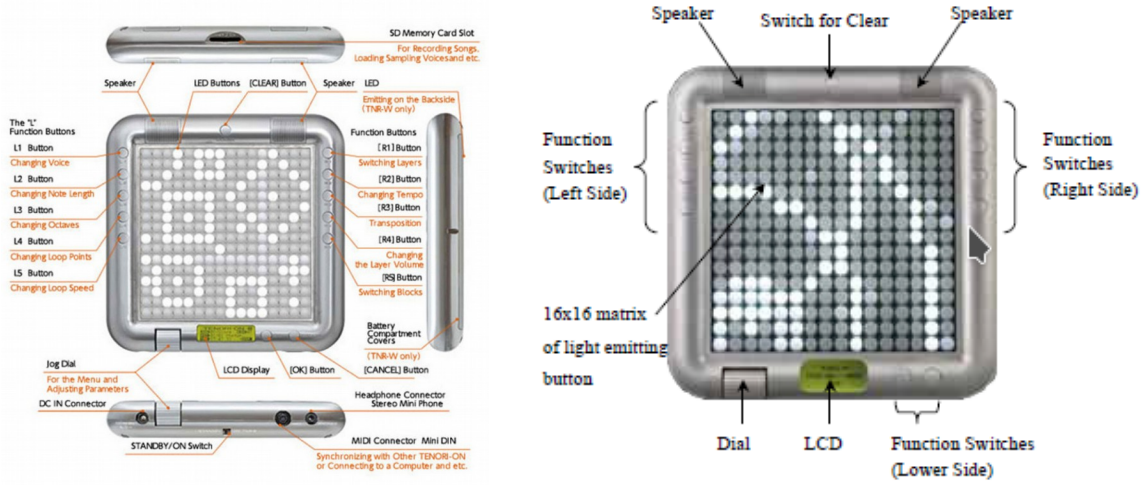


Figura 2: El interfaz del TENORI-ON físico [1].

El TENORION físico cuenta con una interfaz cuadrada, Figura 2, en la que una botonera (de 16×16 LEDs) permite combinar luz y sonido. El marco exterior tiene distribuidos una serie de controles y una pequeña pantalla LCD. En su interior [1] hay un controlador MIDI y un sintetizador, que puede utilizar hasta 16 niveles o pistas de grabación, a cada una de las cuales se le pueden asignar notas de manera independiente. Las pistas pueden reproducirse juntas o de forma aislada y, cada una de ellas, puede utilizarse en seis modos diferentes de interpretación que ofrecen formas distintas de introducir y combinar las notas: *score* (partitura), *random* (aleatorio), *draw* (dibujo), *bounce* (rebote), *push* (empuje) y *solo* (solista). Estos deciden cómo es la dinámica de la botonera en cada paso. En este trabajo solo vamos a hacer algo parecido al modo *score*, en el que el usuario activa o desactiva, a su voluntad, cada nota de la interfaz al

3 Cita de "TNR-i - Overview – Apps – Synthetizers & Mucsic Production tools". Disponible en <https://uk.yamaha.com/en/products/music_production/apps/tnr-i/>.

pulsar el botón correspondiente. En teoría, cualquiera puede tocar un *TENORION*, ayudado por las pistas visuales que ofrece la interfaz y su propia intuición … pero el funcionamiento de este instrumento va más allá del compás (o *loop*) que se ve en un momento en pantalla al permitir (véase Figura 3) el uso de pistas (layers) y agrupaciones de estas (*blocks*) que pueden funcionar en cualquiera de los seis modos disponibles. Lo que proponemos aquí no tiene tanta sofisticación.

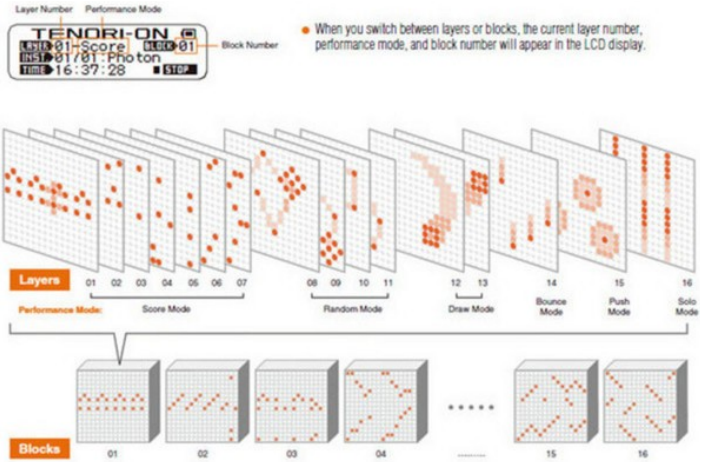


Figura 3: El uso de "layers" y bloques aumenta las posibilidades y la dificultad de uso del TENORI-ON. Imagen extraída de [5].

3.2 ¿Cómo puedo hacer mi propia versión de este instrumento?

Otros instrumentos han sido creados con la idea de una interfaz tangible, véase la Figura 4, como el *Reactable*, los *Audiocubes o el LaunchPad;* versiones software como el *Tyonar* [10] para *Android*, el *Tonenatrix de AudioTools* y prototipos educativos como el *Tone Matrix Music Box [7]*. Con sus propias particularidades, todos incluyen un sintetizador, una interfaz visual y diferentes modos de funcionamiento para dar posibilidades al instrumento.

a) *b)* *c)* *d)* *e)*

Figura 4: Productos similares al TENORION: Reactable (a), Audiocubes (b) , LaunchPad (c) DJ SoundBoard (d) o Tone Matrix Music Box (e). Imágenes extraídas de [7] y de la Wikipedia.

El modo de funcionamiento del TENORION es a través de una referencia temporal que va avanzando por las columnas, haciendo sonar las notas correspondientes a los botones que están iluminados en ese momento, véase la Figura 5a, donde una referencia temporal (indicador de bucle o *loop indicator*) indica la columna de notas cuyo estado lanzará o detendrá la reproducción del sonido asociado a los botones activos (*Active note*) y también decidirá si están apagados o encendidos. En cada fila la nota es la misma y en las columnas se distribuyen verticalmente las notas del banco de sonido que se haya asignado. Básicamente, como muestra la Figura 5b, la dirección horizontal de este instrumento representa el eje del tiempo y la dirección vertical la frecuencia.

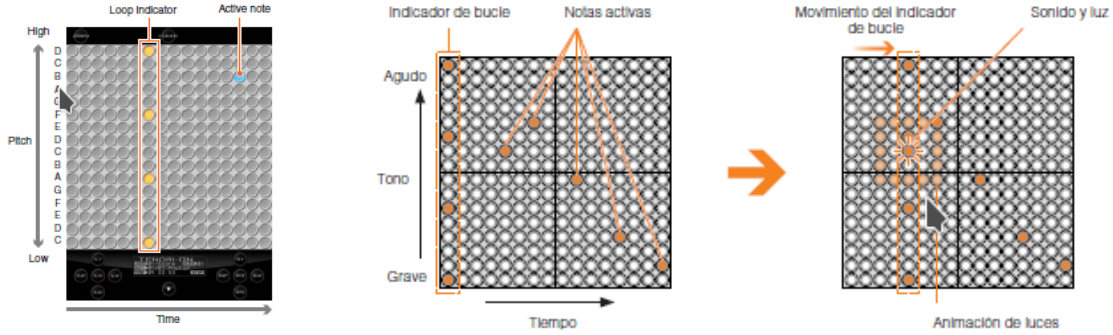


Figura 5: Descripción gráfica del TENORION para plataforma iOS [5].

La idea es llevar esa interfaz del TENORION a la pantalla del computador e interactuar con ella mediante el movimiento de las manos delante de una cámara (para lo cual haremos uso de técnicas de Visión por Computador, utilizando la librería *OpenCV*) y podemos hacer sonar el instrumento con audio 3D para ambientar la escenografía sonora (haciendo uso de *OpenAL*), lo que permitiría, en un futuro, situar espacialmente el sonido alrededor del usuario. Esta es una característica que no posee el instrumento real, así que de momento no vamos a insistir en ello, pero tenemos la base preparada para hacerlo. Como "tocar" un punto sin el uso de accesorios especiales puede ser poco preciso con esta interfaz virtual, hemos decidido que el movimiento de las manos sea el elemento que "da energía" a la botonera virtual y que, con el tiempo, se vaya perdiendo esa "carga" que se volverá a incrementar si algo se mueve encima de ella. Con estas ideas en mente, el algoritmo que vamos a desarrollar toma la forma de la Figura 6. La explicamos en el punto de Desarrollo.

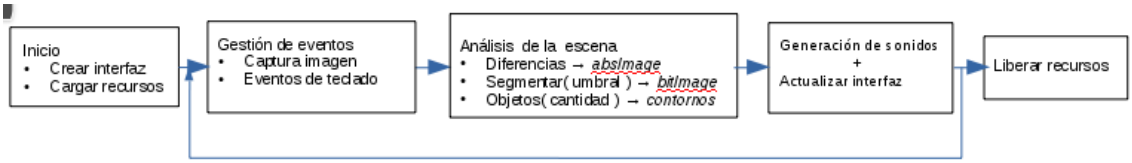


Figura 6: Revisión del diagrama básico de la aplicación.

4 Desarrollo

A la hora de iniciar el desarrollo del contenido sobre el que versa el artículo es necesario hablar de qué componentes deben estar presentes, independientemente de la plataforma. Otra cosa es que para instalarlas se utilicen procedimientos diferentes en plataformas diferentes.

Aunque la elección de *OpenAL* nos permite generar notas a partir de señales simples, para que el sonido que se reproduzca suene más al original era interesante disponer de muestras de sonido (*samples*) que sean similares al instrumento real. Para ello se puede hacer uso de bancos gratuitos como los de *Legowelt*[4] (que ocupa 266 MB), o el *TENORI-DRUMS*[5] (que ocupa 14 MB) y que es

4 *YAMAHA DX-FILES*. Disponible en <http://legowelt.org/samples/>. Contiene cerca de 300 sonidos de varios sintetizadores de la serie DX de *Yamaha*: DX100, TX81Z, DX21, DX7 y DX5. El mismo autor tiene otros similares de las marcas *Korg*, *Roland* y *Ableton*.

5 *TENORI-DRUMS Sample Pack*. Disponbile en <https://free-sample-packs.com/tenori-drums/>.

un conjunto de 14 “kits” de 16 muestras de sonidos del TENORI-ON real. De esta segunda referencia hemos extraído un solo conjunto de muestras (que ocupa 76 KB), con lo que tenemos material para jugar y, cuando estemos contentos con la mecánica de nuestro instrumento virtual, podemos hacer crecer el contenido de este directorio y disponer algunas acciones para elegir el conjunto de entre los que dispongamos. Con lo que la distribución de la aplicación final contendrá un directorio, donde poder organizar estos conjuntos por autor y nombre, con lo que queda como se muestra en la Figura 7.

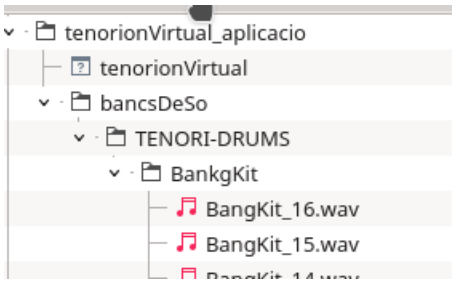
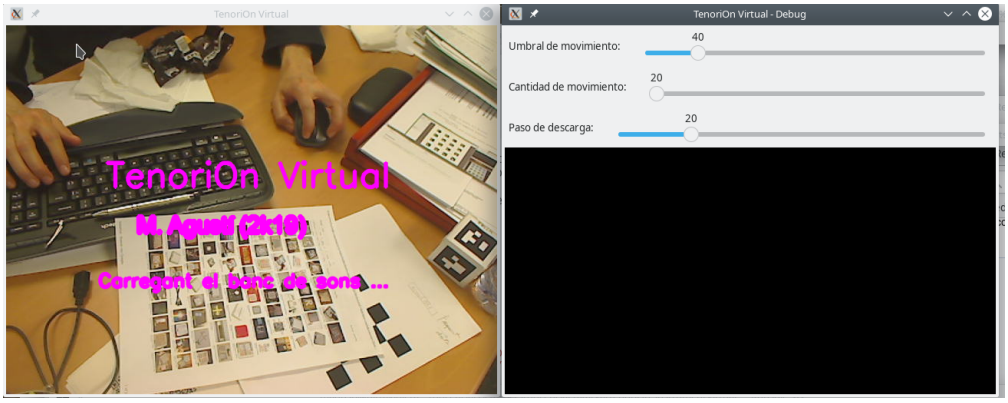


Figura 7: Estructura de la aplicación desarrollada: ejecutable y un bancos de sonido.

Y para instalar los paquetes necesarios y compilar la aplicación en Linux podemos hacerlo ejecutando las órdenes siguientes:

```
$ sudo apt-get install opencv-dev openal-dev alut-dev
$ g++ TENORI-ONVirtual.c -o TENORI-ONVirtual `pkg-config openal
opencv --cflags` -lalut `pkg-config openal opencv --libs` -lm
```

Al ejecutar la aplicación se muestra una ventana con la imagen de la cámara y un texto de presentación, Figura 8a, junto con otra ventana (que tiene interés para nosotros como desarrolladores) que es la que muestra cómo los parámetros utilizados influyen en el desarrollo de la aplicación, Figura 8b, permitiendo ajustar los tres parámetros de funcionamiento de forma visual. En primer lugar, el “Umbral de movimiento”, permite configurar cuánto debe variar el nivel de intensidad de un punto de la imagen para considerar que ha habido cambio. Este valor permite obtener la “imagen de diferencias”, cuyos puntos toman valores de grises resultado de la diferencia entre la imagen actual y la última tomada, en la que se han eliminado los valores inferiores a este umbral. Eliminando, así, las pequeñas variaciones debidas a cambios de iluminación.



a) b)

Figura 8: Inicio de la aplicación.

En segundo lugar, la “Cantidad de movimiento” controla el tamaño de los objetos que se encuentran en la imagen de diferencias, obteniendo una imagen binaria en la que se aislará el objeto de mayor tamaño. Hemos mantenido una lista de otros objetos en la implementación, pensando en que pueda haber más de un participante o que se utilicen las dos manos ;-). En tercer lugar, el “Paso

de descarga”, que es el valor del decremento de la “carga” de los elementos del interfaz. Cuanto mayor sea, menos tiempo estarán activos los elementos de la botonera. Si se quisiera un modo de funcionamiento binario de botón activado que se mantiene hasta que el usuario vuelve a actuar sobre él, bastará con definir un valor de descarga igual a cero y que sea el algoritmo el que haga el cambio de estado en esta situación. Además, en cualquier momento, la ventana de depuración alterna el contenido de la imagen de diferencias o la resultante de la binarización al pulsar la tecla ‘b’. Y se puede utilizar la tecla ‘F’ para mostrar en pantalla completa el interfaz visual y dejar de ver la ventana de depuración o ‘f’ para volver a los tamaños originales. También se puede utilizar la tecla ‘p’ o ‘P’ para bajar o subir los BPM, que es la velocidad a la que se mueve al referencia temporal y, por tanto, a la que se ejecuta la música.

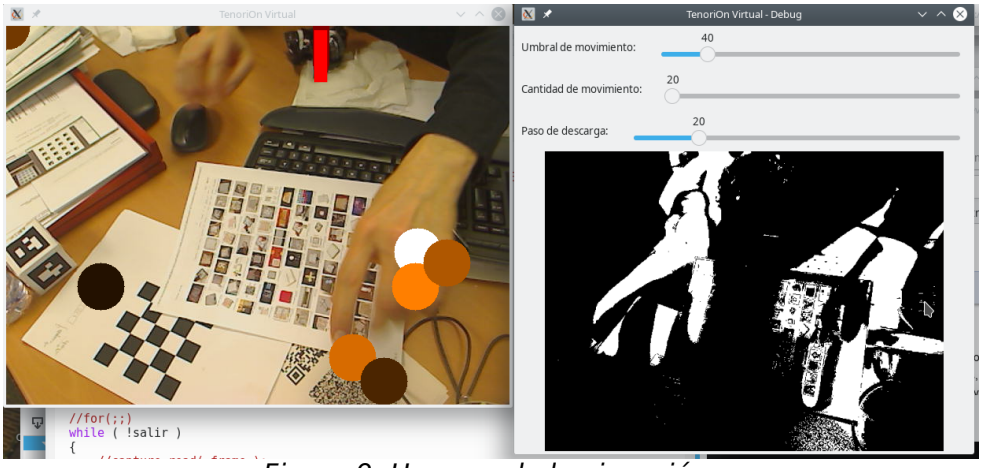


Figura 9: Un paso de la ejecución.

Durante la interpretación del instrumento, la referencia temporal se muestra (véase la Figura 9) en la parte superior con un rectángulo de color rojo y los botones con círculos de color: blanco si está activado, negro si está a punto de desactivarse y en una gradación de colores anaranjados para indicar su nivel de carga. Todos los estados son transitorios, esto es, reflejan la dinámica de la aplicación que va haciendo que evolucionen sus valores de “carga”. De fondo vemos la imagen de la cámara para posibilitar que el contexto (si se quiere) influya en la interpretación (si la cámara está enfocando al público o al espacio alrededor del intérprete) o a una escenografía fija de fondo (como la mesa y las hojas que tenemos por ahí). La ventana de la derecha de la Figura 9 muestra muchos elementos en blanco, esto es que ha habido mucho movimiento.

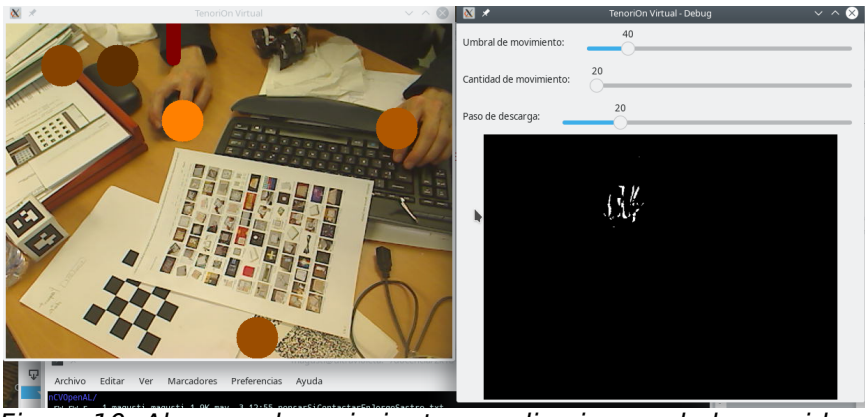


Figura 10: Al parar el movimiento van disminuyendo los sonidos.

La Figura 10 muestra que, al parar el movimiento, los puntos van haciéndose más oscuros, esto es, se están “descargando” al no haber movimiento apreciable, según los los parámetros actuales. Si se quiere que el proceso de “descarga” sea más rápido, se puede subir el paso de descarga. Así, la Figura

11 muestra que no hay movimiento y los puntos de la interfaz toman pocos valores en la escala de los naranjas, que es la que refleja actualmente el proceso de descarga.

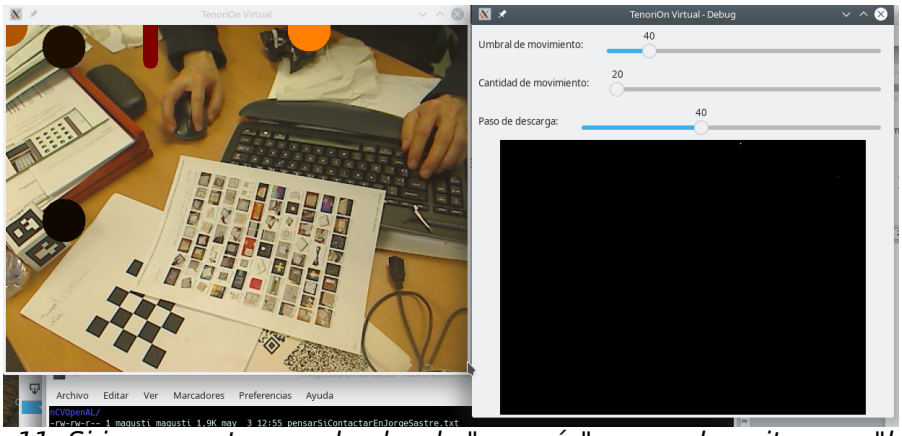


Figura 11: Si incrementamos el valor de "energía" que se le quita a un "botón" encendido, este permanece menos tiempo activado.

La Figura 12 muestra como el subir el valor de umbral del movimiento sirve para eliminar los pequeños “movimientos” en la imagen que son debidos a los pequeños cambios de iluminación propios de una escena real. Si no nos movemos, nada blanco aparece en la ventana de depuración.

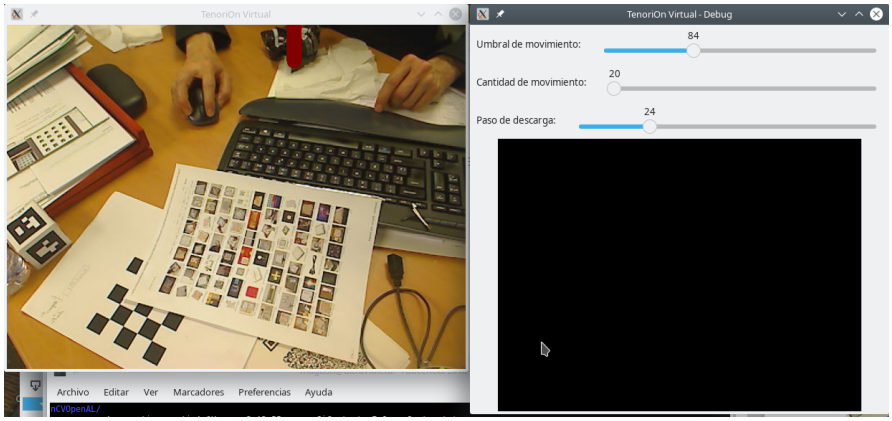


Figura 12: Subiendo el margen para que la diferencia de intensidade tenga que tomar un valor mayor para que se considere que ha habido movimiento.

El Listado 1 muestra el trocito de código que emplea estos parámetros para obtener la posición del “movimiento” en la escena que recoge la cámara, los comentarios en el código indican qué hace cada instrucción. Observemos la cadena de operaciones de *OpenCV* que toman estos valores como parámetros y el cálculo de momentos para determinar el centro de gravedad del contorno de mayor tamaño. Veremos resaltado el nombre de las imágenes que pueden aparecer en la ventana de depuración y el uso del parámetro de la aplicación “Umbral de movimiento”. Se utiliza el cálculo de momentos de *OpenCV* para determinar las propiedades geométricas de los objetos, esto es, las zonas donde hay movimiento. Ahora que ya se conoce dónde se realiza el movimiento se puede pintar el interfaz y hacer sonar lo que corresponda al estado del “instrumento” en el instante actual.

El Listado 2 muestra el contenido de la función `pintarBotoneraTENORI-ON` que es la encargada de pintar los botones encendidos acorde con la dinámica del sistema, utilizando colores en el rango establecido en el diseño de la aplicación. Esta dinámica se realiza en función del parámetro de “Paso de descarga” que aparece destacado en el código y que lleva cuenta de la energía que les asigna el movimiento o la descarga que supone el paso del tiempo sin observar movimiento encima de los botones.

```
      …
    capture.retrieve(frame, 0 ); // Toma la imagen → frame
   // Detectar la posición de la acción → ¿objeto de mayor tamaño?
    flip(frame, frame, 1); // 1 around y-axis
    cvtColor(frame, image, CV_BGR2GRAY); //RGB a grises
  // Diferencias entre cuadros consecutivos en el tiempo → movimiento
  absdiff( image, lastImage, diffImage );
  lastImage = image.clone(); // actualizo el útimo cuadro <- actual
  // Binarizar la imagen por el valor umbral de movimiento
  threshold(diffImage, bitImage, umbralMoiment, 255, THRESH_BINARY);
  // Detectar objetos por contorno
  findContours( bitImage, contornos, RETR_TREE, CHAIN_APPROX_SIMPLE );

  nObjecte = -1; // Si no hay movimiento → no contornos → no objetos
  if( contornos.size() != 0 )  {
    mu.resize( contornos.size() );
    nObjecte = 0;
    mu[nObjecte] = moments( contornos[0], false);
    for( idx=1 ; idx < contornos.size(); idx++ )   {
              mu[idx] = moments( contornos[idx], false);
              if ( mu[idx].m00 > mu[nObjecte].m00 )  nObjecte = idx;
              idx++;
    }
  } // Fi de if( contornos.size() != 0 )
  ...
```

Listado 1: Operaciones para detectar la posición del movimiento.

```
void pintarBotoneraTENORI-ON( int width, int height,
            vector<Moments> mu, int nObjecte, int pasDeDescarregaDelBoto )  {
  int i, j,xOBJ_INT, yOBJ_INT;   Mat roiFrame, roiBitImage;
  Point2f centreObjecte;   float xOBJ_FLOAT, yOBJFLOAT;

  if ( nObjecte != -1 )   {
   xOBJ_FLOAT = static_cast<float>(mu[nObjecte].m10/mu[nObjecte].m00);
   yOBJ_FLOAT = static_cast<float>(mu[nObjecte].m01/mu[nObjecte].m00);
   xOBJ_INT = static_cast<int>(mu[nObjecte].m10/mu[nObjecte].m00) % NUM_COLS;
   yOBJ_INT = static_cast<int>(mu[nObjecte].m01/mu[nObjecte].m00) % NUM_FILES;
   centreObjecte = Point2f( xOBJ_FLOAT, yOBJ_FLOAT );
   // Hace visible la posición del objeto más grande
    circle( frame, centreObjecte, radioPunter,
              CV_RGB(255,128,0), CV_FILLED, LINE_8, 0 );
   // Comprobar si hay movimiento en una misma zona o no → Cargar/Descargar
   for(i=0;i<NUM_FILES;i++){
    for(j=0;j<NUM_COLS;j++){
      if  (i == yOBJ_INT) &&  (j == xOBJ_INT)){
        tempsAparat[i][j]++;
        if ( tempsAparat[i][j] > NCUADROS_TEMPS_APARAT ) {
         tempsAparat[i][j] = 0;   botoneraTENORI-ON[i][j] = 255;
         circle( frame, centreObjecte, radioPunter,
                   CV_RGB(256,256,256), CV_FILLED, LINE_8, 0 );
       }// Fi de if ( tempsAparat[[] > NCUADROS_TEMPS_APARAT
      } // Fi de then de if ((i == yOBJ_INT) && (j == xOBJ_INT))
      else       {
       botoneraTENORI-ON[i][j] -= pasDeDescarregaDelBoto;
       botoneraTENORI-ON[i][j]  = ( botoneraTENORI-ON[i][j] < 0? 0 :
                                    botoneraTENORI-ON[i][j] );
      }
    }// For(j
   }// For(i
  } // Fi de  if ( nObjecte != -1 )
} // Fi de pintarBotoneraTENORI-ON
```

Listado 2: Código de pintar la botonera en el TNR-On virtu-AL.

El Listado 3 muestra el código de la función `tocarTENORI-ON` que dibuja la referencia temporal para indicar la columna de “botones” que se comprueban para generar nuevos sonidos o no. En caso de que uno de ellos esté activado, se resaltará el botón y se hará sonar lo que corresponda, según el banco de

sonidos. Este se carga al principio de la aplicación, así que en tiempo de ejecución está definido unívocamente el sonido a reproducir.

```
void tocarTenorion( int width, int height,
                    int j, Mat bitImage, int quantitatMoiment ) {
  int i;
  line( frame, Point( ((width/NUM_COLS)*j), 0),
        Point( ((width/NUM_COLS)*j), 50),   // Referencia de tiempos
       CV_RGB(128,0,0), (width/NUM_COLS/2), LINE_8, 0 );
  for(i=0;i<NUM_FILES;i++) {
    for(j=0;j<NUM_COLS;j++) {
    if( botoneraTenorion[i][j] > 0 ) { //cantidad de movimiento
      circle( frame, Point( i*(width/NUM_COLS), j*(height/NUM_FILES)),
            radioPunter,
          CV_RGB(botoneraTenorion[i][j],botoneraTenorion[i][j]/2,0),
            CV_FILLED, LINE_8, 0 );
      alSourcePlay(fontsAudio[(i*NUM_COLS)+j]);
    } // if ( nPuntsCanviats > quantitatMoiment )
    } // for(j=0;j<NUM_COLS;j++)
  } //for(i=0;i<NUM_FILES;i++)
} // Fi de tocarTenorion
```

Listado 3: Código encargado de reproducir el sonido del TNR-On virtu-AL.

5 Conclusión

El TENORION apareció allá por el año 2007 y costaba alrededor de 1200€. De primera mano todavía está por esos precios y baja a los 400€ en equipos usados. Las versiones para iOS, el TNR-i y el TNR-e, están disponibles por 21,99€. Aquí hemos propuesto un buen rato de diversión mientras se pone en marcha el ejemplo y muchos días de experimentación, si se quiere, al ir añadiendo otras funcionalidades.

Hemos establecido un posible mecanismo para posicionar las notas en el espacio utilizando el movimiento, a partir de las diferencias en el contenido de dos imágenes tomadas en secuencia y regulado con unos parámetros para que el usuario pueda establecer las tolerancias del sistema .

¡Ánimo, quedan cosas por hacer y mucha diversión por el camino experimentando con la aplicación! Si te interesa, envíame un correo y te hago llegar el código completo.

6 Bibliografía

[1] *Yamaha Design "Synapses" TENORI-ON* Disponible en <http://www.yamaha.com/>.

[2] Nishibori, Yu; Iwai, Toshio (2006). "Tenori-on" (PDF). Proceedings. Int'l. Conf. on New Interfaces for Musical Expression (NIME-06). pp. 172–175.

[3] *Wikipedia contributors. Yamaha Tenori-on. Wikipedia, The Free Encyclopedia. March 27, 2019, 01:07 UTC*. Disponible en <https://en.wikipedia.org/ >.

[4] *TENORI-ON* , el juguetito de Yamaha. Disponible en <https://www.nightclubber.com.ar/>.

[5] Manuals: TNR-i Quick Guide. Disponible en <https://uk.yamaha.com/>.

[6] *DISC Yamaha Tenori-On With FREE Case*. En la URL <https://www.gear4music.com>.

[7] *Tone Matrix Music Box. ECE 477 Senior Design - Group 17*. Disponible en <https://engineering.purdue.edu/>.

[8]. *ToneMatrix*, Disponible en <https://geeksroom.com/>.

[10] *Tyonar*. Disponible en <https://www.xatakandroid.com/>.