Diseño de interface para el desarrollo de una pantalla sensible al tacto con aplicación musical

Autor: Emiliano Causa

Proyecto de investigación ""Diseño y desarrollo de aplicaciones e interfases de Realidad Aumentada destinadas a síntesis y procesamiento de audio digital." perteneciente al programa PICTO-ARTE – Director Carmelo Saitta y Pablo Cetta – Área Transdepartamental de Artes Multimediales del Instituto Universitario Nacional del Arte (IUNA) – Yatay 843 Ciudad Autónoma de Buenos Aires Tel. 54-11-4862-8209

emiliano.causa@gmail.com

Palabras claves

Pantalla sensible al tacto, Pitch Class Set, interpretación gestual, interfaces tangibles

1. Introducción

El presente texto expone el proceso de diseño de una interface de pantalla sensible al tacto (del tipo multitacto) aplicada a un editor gestual de música. La aplicación consiste en un editor de partitura analógica que permita escribir gestos musicales de una forma sensilla, a la vez que se determina su dinámica, tempo y grado de consonancia. El presente trabajo forma parte de la investigación "Diseño y desarrollo de aplicaciones e interfaces de Realidad Aumentada destinadas a síntesis y procesamiento de audio digital" (dirigida por Carmelo Saitta y Pablo Cetta). El Dr. Pablo Cetta viene desarrollando (junto al Dr. Pablo Di Liscia) investigaciones sobre los Pitch Class Set, para el diseño armónico de la música. El presente desarrollo intenta crear un interface intuitiva para la creación musical aprovechando el conjunto de librerías de programación creadas en la investigación sobre Pitch Class Set. Una aplicación de este tipo permitiría crear gestos musicales y definir su nivel de consonancia en forma dinámica, gracias a la aplicación de los Pitch Class Set.

2. Elementos musicales

En los primeros pasos de este desarrollo, durante reuniones con el Dr.Pablo Cetta y el Lic. Matías Romero Costas, se definió un conjunto de gestos musicales (notación musical) que podrían trazarse sobre una pantalla sensible al tacto. Este conjunto está definido por la siguiente notación:

Alturas puntuales

Diseño melódico

Acordes

Arpegios

Duración

Glissando

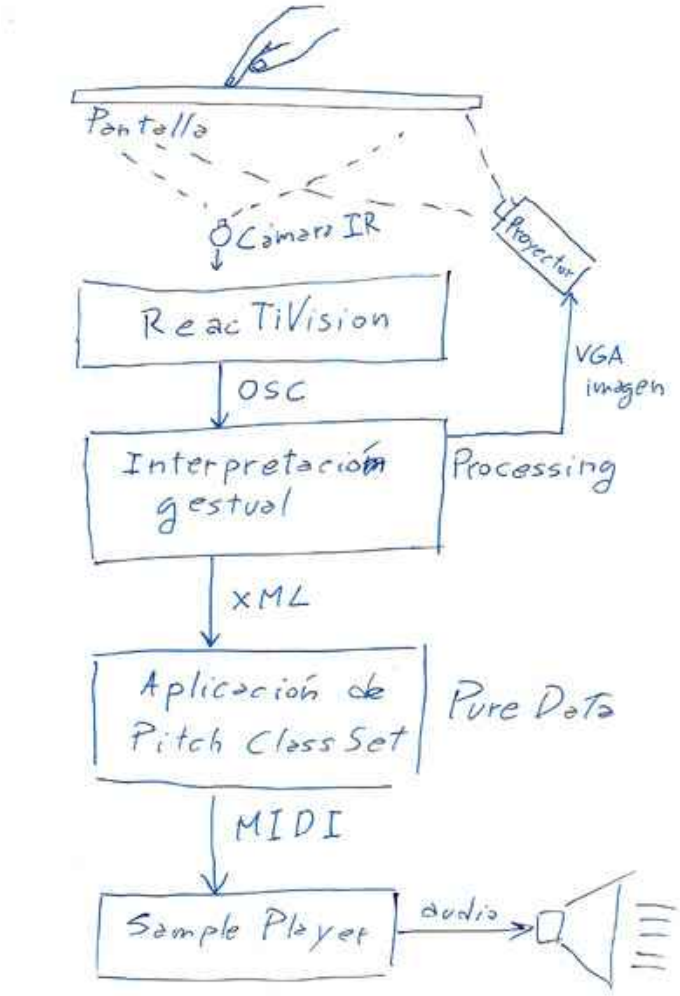
Trémolo

Trino

3. Elección de herramientas

Dado que las librerías ya citadas están desarrolladas en Pure Data (una herramienta de software para desarrollos de composición algorítmica) se diseño una plataforma que vincula diferentes software para implementar una pantalla sensible al tacto con una interface que interprete la notación recién descripta. Para el desarrollo de la pantalla se eligió ReacTiVision. Una aplicación que realiza un sistema de sensado óptico de interfaces tangibles y también funciona con una pantalla sensible al tacto. Este software es Open Source, lo que nos permite utilizarlo para este desarrollo.

Con el fin de interpretar los gestos desarrollado sobre la pantalla, para traducirlos en los diferentes elementos musicales, se utiliza Processing. ReacTiVision se comunica con Processing vía OSC, un protocolo de comunicación pensado para la transmisión de sonido, pero que en este caso transmite el tipo de elemento de intertace que se encuentra sobre la pantalla, así como los gestos desarrollados con los dedos.



Una vez que Processing interpreta los gestos y los traduce en elementos (por ejemplo un dedo apoyado sobre la pantalla en forma solitaria es interpretado como una altura puntual), se comunica con Pure Data mediante un protocolo de red, transmitiendo en formato XML cada elemento. Por último, Pure Data se encarga de gestionar los mensajes MIDI necesarios para reproducir los elementos musicales. Estos mensajes llegan a un Sample Player que se encarga de producir finalmente el sonido.

3.1. ReacTiVision y las Interfaces Tangibles (*)

ReacTiVision (mtg.upf.es/reactable/) es una herramienta de software desarrollada por Sergi Jordà, Martin Kaltenbrunner, Günter Geiger y Marcos Alonso, quienes conforman el Grupo de Tecnología Musical dentro del Instituto Audiovisual en la Universidad Pompeu Fabra (Barcelona España). Esta tecnología permite reconocer patrones bitonales (llamados “fiducials”) impresos a piezas de interfaces tangibles que funcionan sobre una pantalla sensible al tacto. Esta interface tangible consiste en piezas de acrílico que se apoyan sobre una pantalla sensible, esta es capaz de reconocer las piezas, gracias a los patrones bitonales y generar respuestas audiovisuales en consecuencia. Los creadores, construyeron este software para desarrollar una pantalla sensible para interpretación musical en tiempo-real (un instrumento de improvisación de música electrónica) llamada ReacTable.

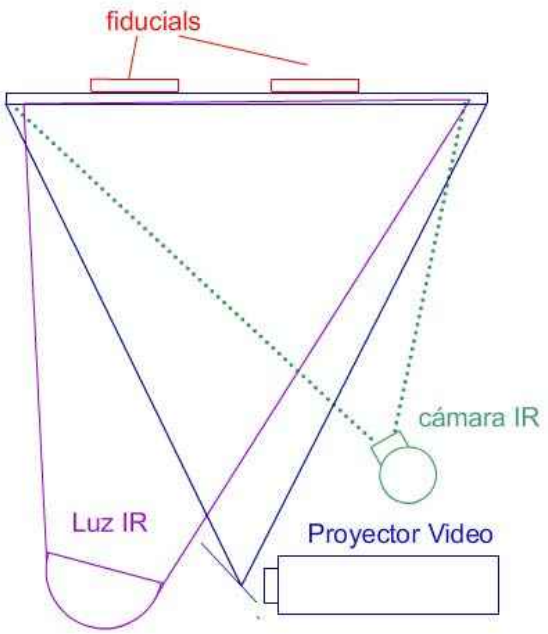


Figura 3: Esquema ReacTable

Como muestra la figura 3, ReacTiVision permite hacer el reconocimiento de patrones bitonales, a través de un sistema óptico, que en el caso de la ReacTable se implementa con luces y cámara infrarrojas. La pantalla es un acrílico con superficie esmerilada, las imágenes se retro-proyectan desde abajo usando un cañón de video, a su vez una luz infrarroja permite iluminar los patrones que serán captados por una cámara, también infrarroja. Dicha luz y cámara son infrarrojas para no interferir la luz del proyector de video (que pertenece al rango visible de la luz), y para que la cámara no vea a su vez las proyecciones.

Uno de los aspectos más interesantes de ReacTiVision es que está construido como un software independiente, que envía datos respecto de los parámetros de los "fiducials": la ubicación, identificación, rotación y otros; vía el protocolo OSC (Open Sound Control). Esto permite que cualquier otro software que reciba mensajes en OSC, pueda comunicarse con ReacTiVision e interpretar información respecto del estado de cada uno de los patrones bitonales ubicados sobre la pantalla. Debido a esto, existe en el sitio de ReacTiVision, ejemplos de conexión de este software con lenguajes como: C++, Java, C#, Processing, Pure Data, Max/MSP, Flash y otros.

3.2. Processing y la librería ReacTiVision (*)

Processing (www.processing.org) es un lenguaje de programación diseño para artistas por Ben Fry y Casey Reas. Es un lenguaje "open source" desarrollado en Java de gran potencia y facilidad de aprendizaje. Una de las ventajas de Processing es el extenso desarrollo de librerías que extienden las posibilidades de conexión de este lenguaje con otros formatos, protocolos o lenguajes. Como hemos dicho en el apartado anterior, existe una librería que permite conectar, vía OSC, a Processing con ReacTiVision.

ReacTiVision tiene una aplicación ejecutable que se conecta a la cámara y reconoce los patrones (fiducials) que estén en la imagen, enviando por OSC los parámetros de cada patrón (en un protocolo que los autores llamaron TUIO). Esta librería implementa un conjunto de instrucciones que permiten leer dichos parámetros. Por ejemplo, la siguiente línea de código crea un objeto "cliente de protocolo TUIO" :

```
TuioClient client = new TuioClient(this);
```

Y las instrucciones que siguen, son funciones que se ejecutan frente a eventos provocados por los patrones:

```
void addTuioObject(int s_id, int f_id, float xpos, float ypos, float
angle) {}


void removeTuioObject(int s_id,int f_id ) {}


void updateTuioObject (int s_id, int f_id, float xpos, float ypos,
float angle, float xspeed, float yspeed, float rspeed, float maccel,
float raccel) {}


void addTuioCursor(int s_id, float xpos, float ypos) {}


void removeTuioCursor(int s_id) {}


void updateTuioCursor (int s_id, float xpos, float ypos, float
xspeed, float yspeed, float maccel) {}
```

Por ejemplo, TUIO distingue dos tipos de elementos: objetos y cursores. Los objetos son los patrones bitonales, mientras que los cursores son los dedos que se apoyan sobre la pantalla (dado que el sistema también es capaz de reconocer el tacto). Cada una de estas funciones, informan un evento de un patrón o de tacto:

addTuioObject: informa la aparición de un nuevo patrón sobre la pantalla.

removeTuioObject: informa que un patrón salió de la pantalla.

updateTuioObject: informa los cambio que sufre un patrón, ya sea de posición como de rotación.

addTuioCursor: informa la aparición de un dedo sobre la pantalla.

removeTuioCursor: informa que un dedo salió de la pantalla.

updateTuioCursor: informa los cambio que sufre un dedo, un cambio de posición.

4. Diseño gestual

Los últimos 3 eventos, que son disparados por acciones con los dedos, son los que se utilizan para interpretar los gestos que son traducidos en los elementos musicales. Basados en esto se desarrollo un diseño gestual que intenta lograr una interface muy intuitiva para el usuario. A continuación explicaremos, para cada elemento, cuál el tipo de gesto que los dedos deben realizar sobre la pantalla, así como la forma en que este elemento es descripto en un mensaje de XML. La aplicación generada con Processing debe encargarse de traducir estos gestos en cada un de sus correspondientes mensajes XML.

4.1. Alturas puntuales

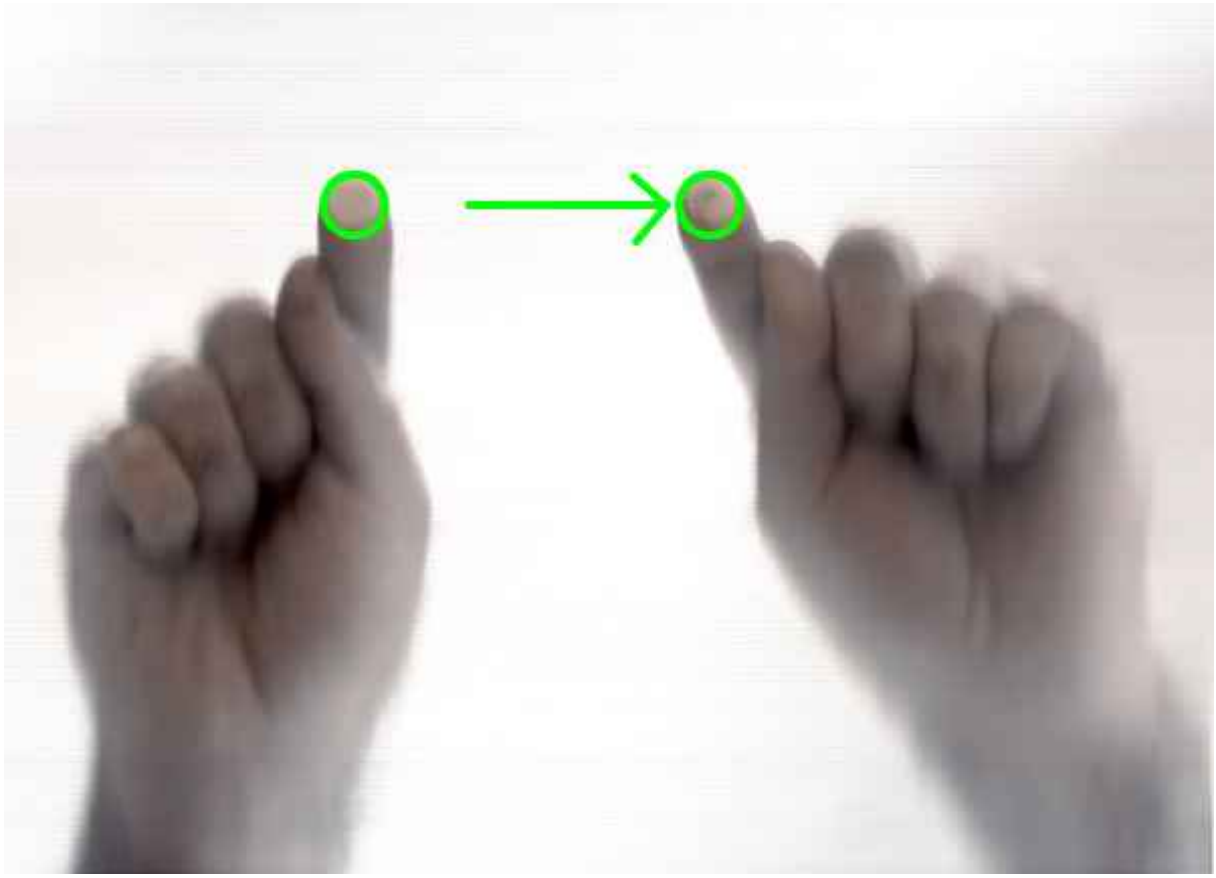
Para la realización de este gesto el usuario deberá hacer presiones sobre la pantalla, usando la punta del dedo y en forma breve:

Este gesto es traducido en el siguiente mensaje XML:

```
<NOTA ID='#'>
  <A> altura </A>
  <T> tiempo_inicio </T>
</NOTA>
```

4.2. Duración

Para establecer la duración de una nota tenida, luego de apoyar el dedo de la nota, se apoya otro por el lado derecho y se realiza un trazo en forma paralela:

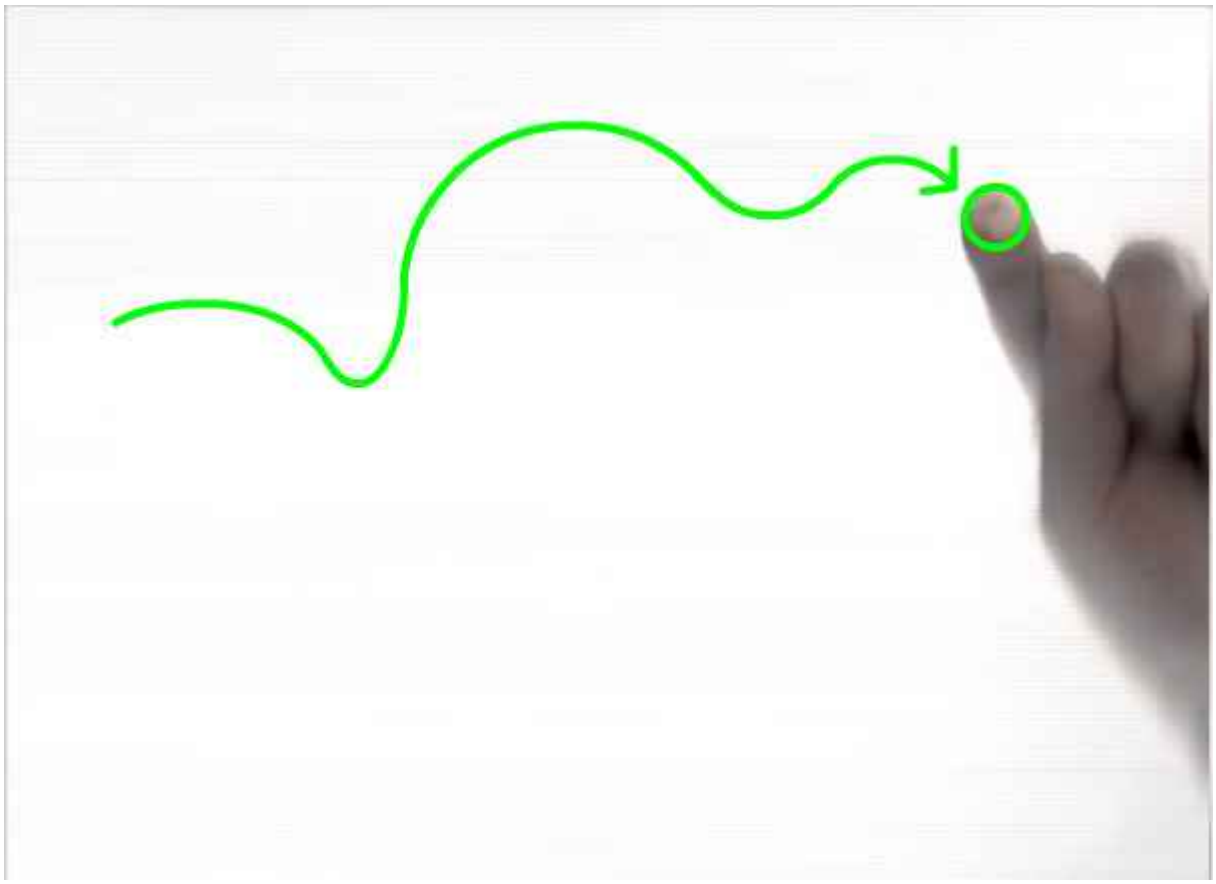


Luego, el gesto es traducido en el siguiente mensaje XML:

```
<NOTA ID='#'>
  <A> altura </A>
  <T> tiempo_inicio </T>
  <D> duración </D>
</NOTA>
```

4.3. Diseño melódico

Este elemento se construye mediante la realización de un trazo continuo en la pantalla:



El mensaje XML que le correspone a este gesto (y su respectivo elemento), es de longitud variable, ya que la melodía podría estar formando por diferentes cantidad de notas. El trazo continuo es traducido mediante un algoritmo de cuantización en una secuencia de notas:

```
<MELODIA ID='#'>
  <CANTIDAD> cantidad_de_notas </CANTIDAD>
  <NOTA>
    <A> altura </A>
    <T> tiempo_inicio </T>
    <D> duración </D>
  </NOTA>
  <NOTA>
    <A> altura </A>
```

<T> tiempo_inicio </T>

<D> duración </D>

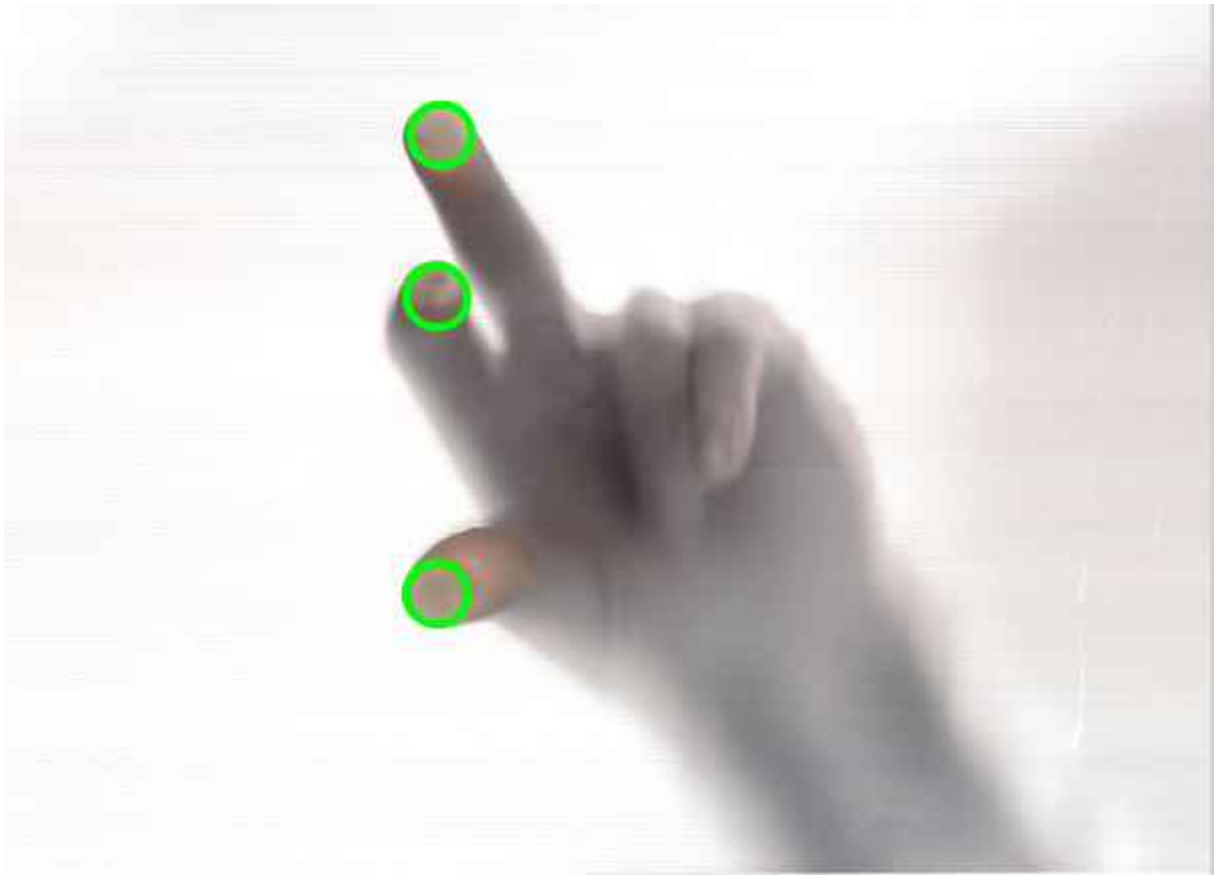
</NOTA>

...

</MELODIA>

4.4. Acordes

Los acordes se construyen mediante el apoyo simultáneo de varios dedos, respetando una forma vertical. En casos de necesitar hacer acordes de varias notas, 4,5 o más, es posible apoyar los 2 primeros dedos y manteniendo el primero, ir agregando nuevas notas. ReacTiVision no distingue respecto de la mano a la que pertenece cada dedo, por lo que para la realización de este gesto es posible (y seguramente necesario) utilizar ambas manos:

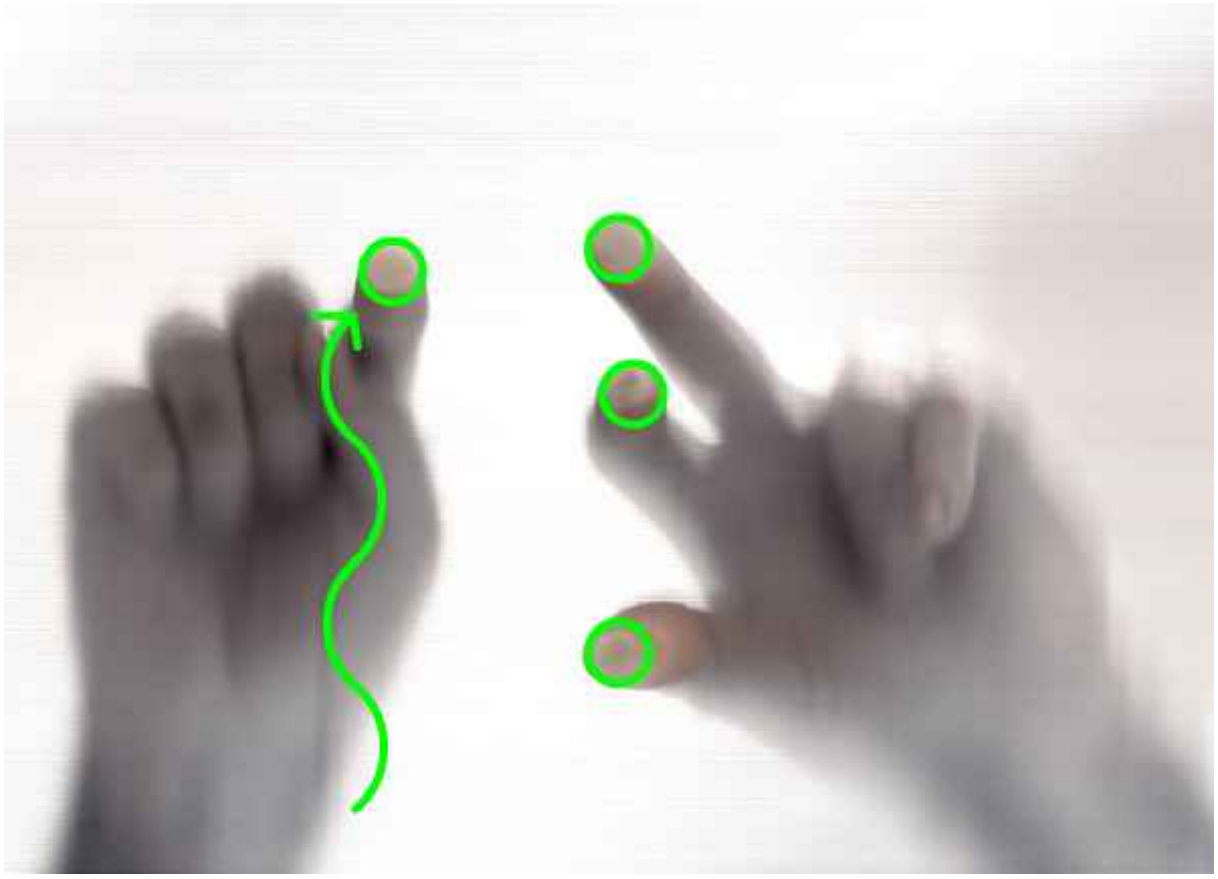


Al igual que lo que sucede con los mensajes XML para las melodías, los mensajes para acordes pueden tener diferentes longitudes en función de la cantidad de notas que contengan:

```
<ACORDE ID='#'>
  <CANTIDAD> cantidad_de_notas </CANTIDAD>
  <NOTA>
    <A> altura </A>
    <T> tiempo_inicio </T>
    <D> duración </D>
  </NOTA>
  <NOTA>
    <A> altura </A>
    <T> tiempo_inicio </T>
    <D> duración </D>
  </NOTA>
  ...
</ACORDE>
```

4.5. Arpegios

Este elemento se construye de forma similar al acorde, se apoyan varios dedos en forma vertical, pero luego, dejando algunos, se realiza un trazo zigzagueante en forma paralela:

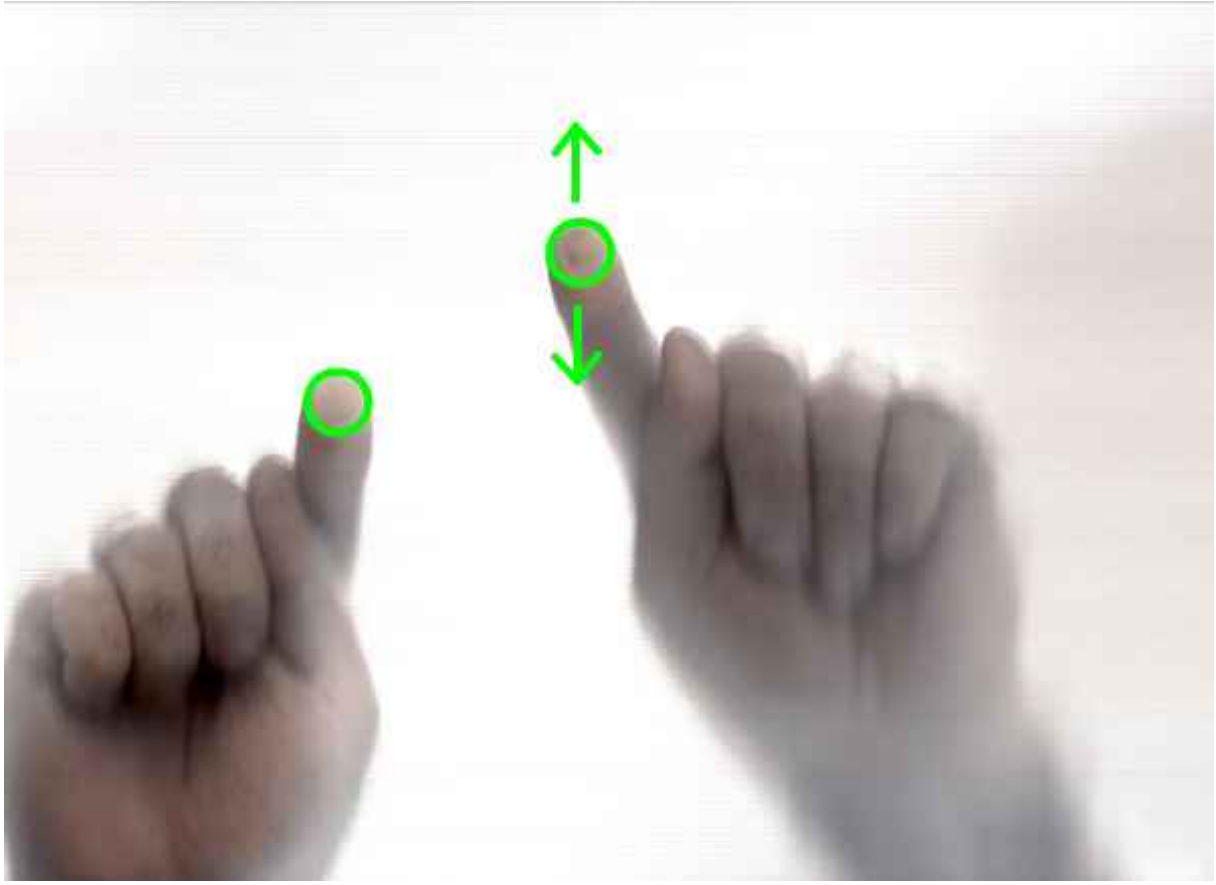


De igual manera, la construcción del mensaje XML correspondiente al arpegio es similar al del acorde:

```
<ARPEGIO ID='#'>
  <CANTIDAD> cantidad_de_notas </CANTIDAD>
  <NOTA>
    <A> altura </A>
    <T> tiempo_inicio </T>
    <D> duración </D>
  </NOTA>
  <NOTA>
    <A> altura </A>
    <T> tiempo_inicio </T>
    <D> duración </D>
  </NOTA>
  ...
</ARPEGIO>
```

4.6. Glissando

El glissando se construye mediante dos dedos que se colocan en forma horizontal, desplazando luego el del extremo derecho (hacia arriba o abajo) para establecer el grado del glissando:



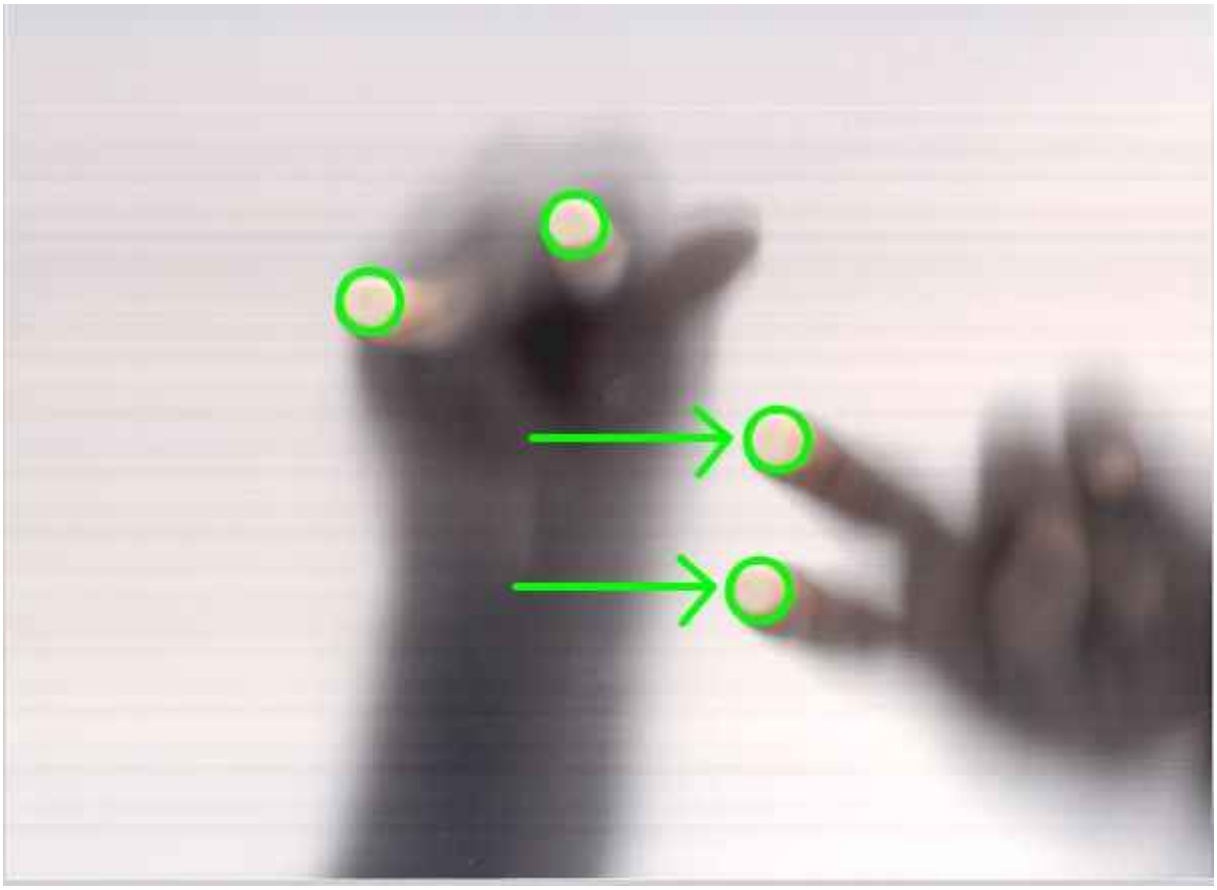
El mensaje XML correspondiente, expresa los puntos incial y final del glissando:

```
<GLISSANDO ID='#'>
  <INICIO>
    <A> altura_inicial </A>
    <T> tiempo_inicial </T>
  </INICIO>
  <FINAL>
    <A> altura_final </A>
    <T> tiempo_final </T>
  </FINAL>
```

</GLISSANDO>

4.7. Trémolo

El trémolo se construye apoyando dos dedos para establecer las alturas y moviendo otros dos dedos debajo, en forma horizontal para indicar que es un trémolo y nooo un glissando:

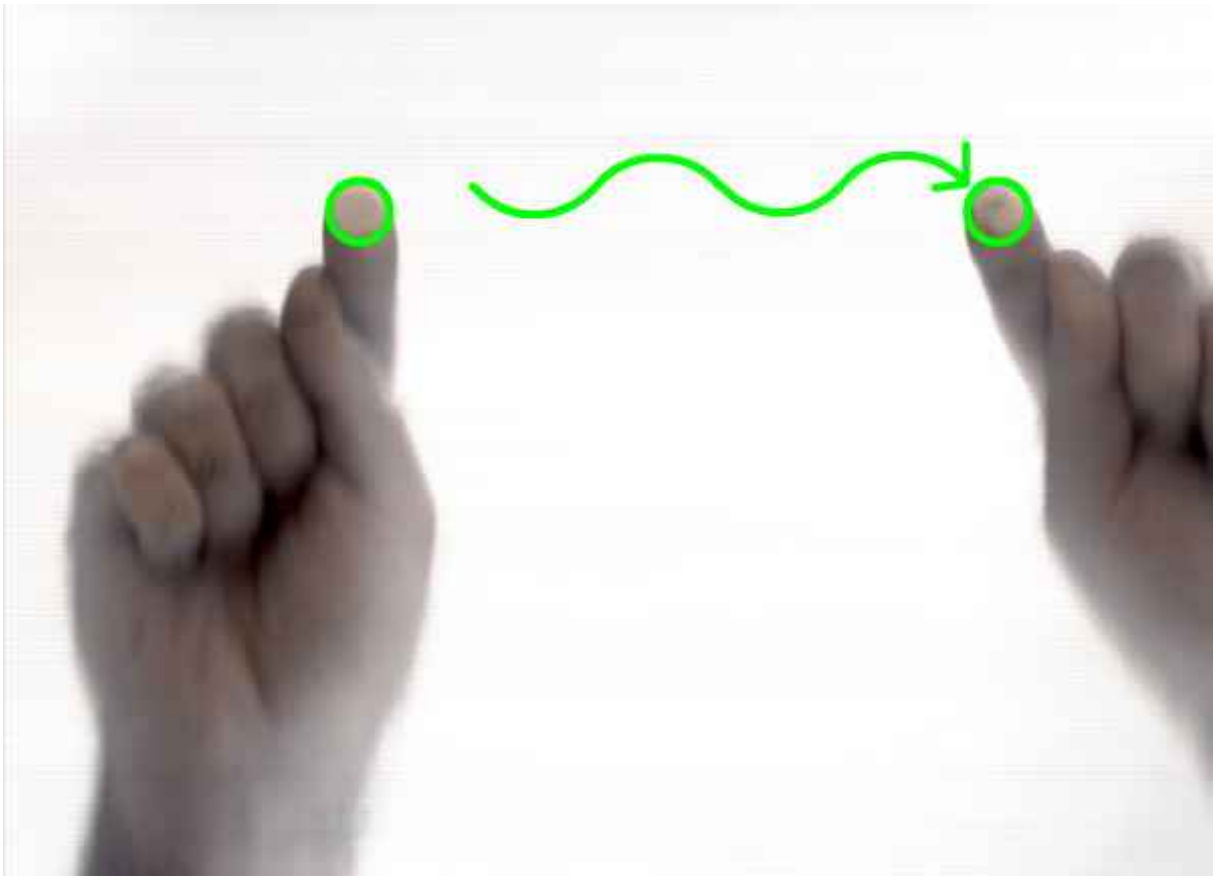


El mensaje XML del trémolo es:

<TREMOLO ID='#'>
  <A> altura_inicial </A>
  <T> tiempo_inicial </T>
  <AF> altura_final </AF>
  <D> duración </D>
</TREMOLO>

4.8. Trino

El trino se construye de forma similar a la nota tenida, sólo que cuando el dedo a la derecha se desplaza, lo hace siguiendo un movimiento zigzagueante:



El mensaje XML del trino es similar al de la duración:

```
<TRINO ID='#'>
  <A> altura </A>
  <T> tiempo_inicio </T>
  <D> duración </D>
</TRINO>
```

5. Algoritmo

Con el fin de interpretar los gestos descriptos anteriormente se diseño el siguiente algoritmo, que es aquí presentado en la forma de un pseudo-código que mezcla instrucciones estándares de cualquier lenguaje de programación junto a enunciados expresados en lenguaje coloquial. El algoritmo de presenta de esta manera para dejar en claro la noción de que el mismo podría ser implementado en cualquier lenguaje de programación (independientemente de nuestra preferencia por Processing). Durante el algoritmo se evaluan diferentes condiciones que permiten distinguir a cada uno de los gestos. Por ejemplo, una de las primeras condiciones a evaluar es la cantidad de dedos implicados en el gesto, ya que sólo 2 de los gestos utilizan un solo dedo. Otra de las condiciones se relaciona con la existencia y discriminación entre dedos quietos y en movimiento, así como la cantidad que hay de cada uno de estos tipos. Cuando hay movimiento, es importante reconocer aquellos gestos que involucran un movimiento zigzagueante, tales como el trino o el arpegio. Otro tema importante es analizar el tipo de distribución que se expone cuando hay varios dedos, es decir, ver si es horizontal o vertical. Se presenta, entonces el siguiente algoritmo:

```
if( cantidad de dedos es mayor a 1 ){

  if( existen dedos móviles ){

    if( cantidad de dedos móviles es igual a 1 AND
    cantidad de dedos quietos es igual a 1 AND
    dedo movil se ubica a la derecha respecto del quieto ){

      if( el dedo movil se desplaza en forma horizontal ){

        if( el dedo movil se desplaza en forma zigzagueante ){

          TRINO( tomar nota inicial de la posicion x,y del dedo quieto y
          y la distancia en x entre ambos dedos para la duración )
        }else{

          DURACIÓN( tomar nota inicial de la posicion x,y del dedo quieto y
          y la distancia en x entre ambos dedos para la duración )
        }
      }else{

        GLISSANDO( tomar nota inicial de la posicion x,y del dedo quieto y
        definir el glissando con la posición del dedo derecho )
      }

    }else if( cantidad de dedos móviles es mayor a 2 AND
    cantidad de dedos quietos es igual a 2 AND
    dedo movil se ubican debajo respecto de los quietos ){
```

```
      if( los dedos móviles se desplazan en forma horizontal ){

          TREMOLO( tomar nota inicial de la posicion x,y del dedo quieto
          de la izquierda y la posicion y del otro dedo quieto )
      }

    }else{

      if( la cantidad de dedos móviles es igual a 1 AND
      la distribución de los dedos quietos es vertical ){

        if( el dedo móvil se ubica a derecha AND
        el dedo movil tiene movimiento zigzagueante){

          ARPEGIO( tomar la posción horizontal del dedo ubicado arriba y
          las posiciones verticales de cada dedo )
        }else{

          ERROR
        }
      }else{

        ERROR
      }
    }
  }else{

   if( los dedos tiene una distribución vertical ){

     ACORDE( tomar la posción horizontal del dedo ubicado arriba y
     las posiciones verticales de cada dedo )
   }else{

     ERROR
   }
  }
}else{

  if( el dedo está quieto ){
```

```
    NOTA_PUNTUAL( tomar posición x,y para posición temporal y altura )
  }else{

    if( no hay superposición vertical ){

      DISEÑO_MELÓDICO( tomar recorrido )
    }else{

      ERROR
    }
  }
}
```

5.1. El análisis del movimiento

A la hora de ejecutar el algoritmo expuesto arriba, es necesario que el análisis se realice una vez que el gesto haya finalizado. Para especificar esto mejor, definiremos un gesto como lo que sucede en pantalla, desde que aparece algún dedo hasta que desaparecen todos, esto quiere decir que la aplicación debe iniciar el registro del gesto en el momento que la pantalla pase del vacio a la existencia de uno o varios dedos, luego debe registrar todo acontecimiento hasta el momento que terminó toda acción, una vez sucedido esto, el gesto se presentará al algoritmo de interpretación de gestos.

5.2. Las condiciones del algoritmos

Existe en el algoritmo ciertas condiciones que requieren su propio algoritmo de análisis. Estas condiciones son:

1. Dedos quietos vs. dedos en movimiento
2. Desplazamientos horizontales vs. desplazamientos verticales
3. Desplazamientos zigzagueantes vs. desplazamientos directos
4. Superposición vertical

5.3. Quietud vs. movimiento

De las condiciones presentadas, esta es quizás la más sencilla de evaluar. Pero una aclaración válida para todas es que el registro de los gestos se realiza guardando la posición inicial de estos (es decir, el punto en el que aparecen) y luego los desplazamientos relativos que se producen de un tiempo al siguiente (entendiendo que los tiempos de ejecución se produce en pasos discretos). Los registros de desplazamientos deben hacerse usando coordenadas polares, es decir en términos de ángulos y distancias, en cambio las posiciones iniciales deben describirse con coordenadas cartesianas.

En función del tipo registro descripto, la quietud puede analizarse como “distancias de desplazamientos” que son menores a un umbral. No se puede pretender que sean de valor cero, ya que el ruido de la captura de movimiento hace que haya pequeños desplazamientos. Un valor de umbral permite discriminar el movimiento real respecto del ruido.

5.4. Desplazamientos horizontales vs. desplazamientos verticales

Antes de intentar dilucidar la forma de establecer este análisis, cabe aclarar que existen trazos horizontales, otros verticales y por último, un tipo de trazo que no pertenece a ninguna de estas dos categorías. Una de las variables principales que permite interpretar la horizontalidad de un trazo sería el “ángulo promedio” de los desplazamientos. Si este promedio se acerca a 0º, 180º ó 360º, es proobable que el trazo sea horizontal, por supuesto que esto debe medirse nuevamente en función de un umbral. Cuando el ángulo promedio se diferencia por un valor menor al umbral, entonces se puede presuponer que es horizontal.

Sin embargo surge aquí el problema de que ciertos trazos podrían tener este tipo de ángulo en promedio, pero en sus variaciones terminar no siendo horizontal. Para distinguir estos casos, además del “ángulo promedio”, hay que analizar la “amplitud de los ángulos”, es decir cuál fue el ángulo mínimo y cuál el máximo. Los trazos cláramente horizontales deben tener una amplitud menor a 90º (u otro valor que se crea conveniente estipular como umbral), entre el mínimo y el máximo, pero de seguro un trazo con amplitudes mayores a los 180º, no son trazos horizontales.

Una vez que se determina que un trazo no es horizontal, entonces queda determinar si es vertical o de otro tipo. El procedimiento es el mismo, pero esta vez los promedios debe acercarse a 90º o 270º.

5.5. Desplazamientos zigzagueantes vs. desplazamientos directos

Para determinar si un trazo tiene un desplazamiento zigzagueante o directo, no sirve el ángulo promedio, ni la amplitud, sino que debe utilizarse la “varianza”, es decir, las variaciones respecto del promedio. Si la varianza es muy baja, el trazo es directo, en cambio si esta supera cierto umbral, entonces el trazo es zigzagueante.

5.6. Algoritmo de análisis de trazos de dedos

El siguiente algoritmo muestra el análisis sobre un “arreglo” que contiene las posiciones en X e Y que ha recorrido el dedo. En función de esto se realiza un promedio ponderado, es decir se suman los ángulos de cada punto multiplicados por el peso de su recorrido en el peso actual:

```
float factor = distancia[i] / totalDistancia;
promedio += angulo[i]*factor;
```

Para esto, es necesario obtener previamente la distancia total recorrida en el trazo.

El algoritmo completo:

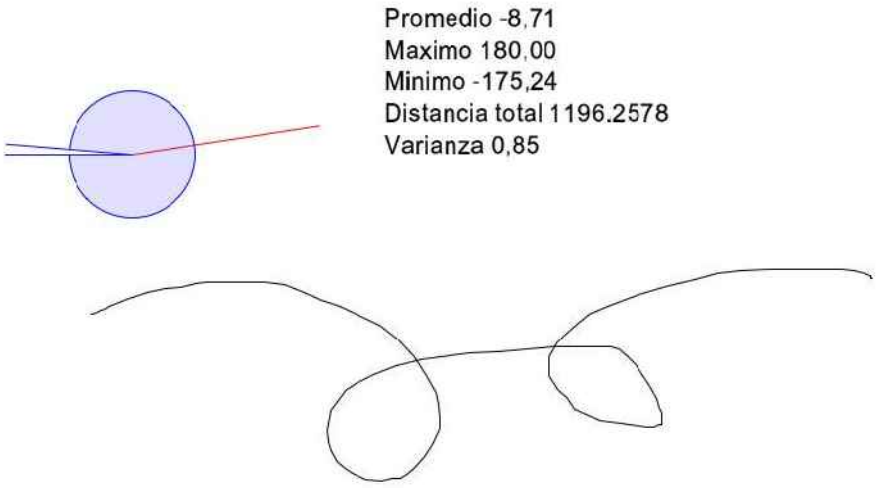
```
promedio = 0;
varianza = 0;
totalDistancia = 0;

boolean min_y_max_cargados = false;

for( int i=0 ; i<cantidad ; i++ ){
  totalDistancia += distancia[i];
}
```

```
if( totalDistancia > 0 ){

  for( int i=0 ; i<cantidad ; i++ ){

    float factor = distancia[i] / totalDistancia;
    promedio += angulo[i]*factor;

    if( distancia[i] > umbral_distancia ){
      if( min_y_max_cargados ){

        if( angulo[i]<minimo ){
          minimo = angulo[i];
        }
        if( angulo[i]>maximo ){
          maximo = angulo[i];
        }

      }
      else{
        minimo = angulo[i];
        maximo = angulo[i];
        min_y_max_cargados = true;
      }
    }

  }

  for( int i=0 ; i<cantidad ; i++ ){

    float factor = distancia[i] / totalDistancia;
    float diferencia = abs( angulo[i] - promedio );
    varianza += diferencia*factor;

  }
}
```

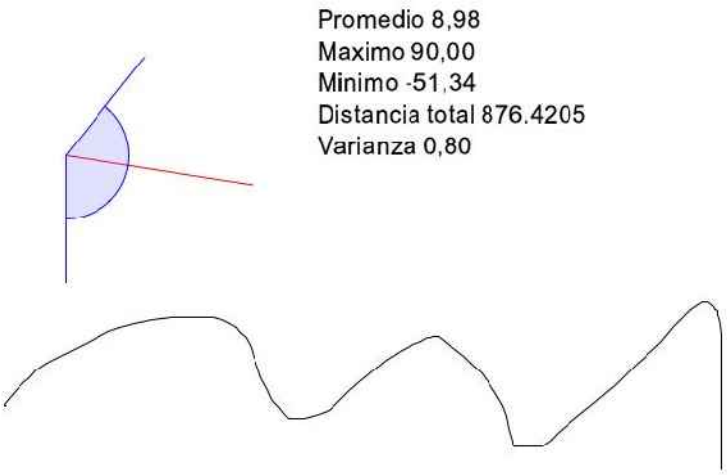
Puede ver los resultados de analizar diferntes curvas con este algoritmo.

A continuación se puede ver un trazo el línea casi recta. Su varianza es muy baja, así como sus extremos (mínimos y máximos) se encuentran cerca entre sí:

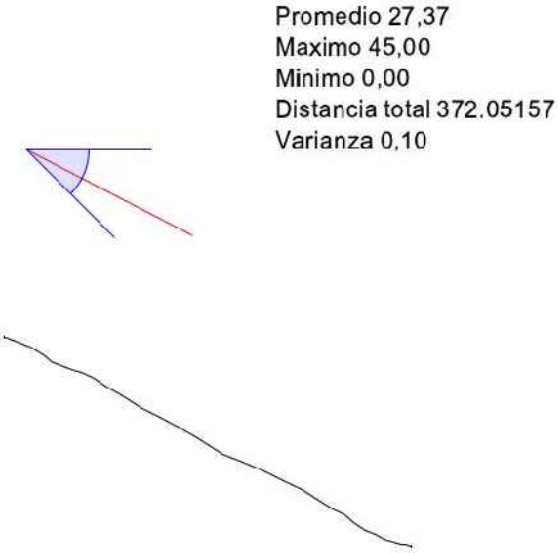
La siguiente es una curva que posee un promedio cercano a la curva anterior, pero sus extremos se encuentra muy lejanos, justamente producto de los bucles que posee:



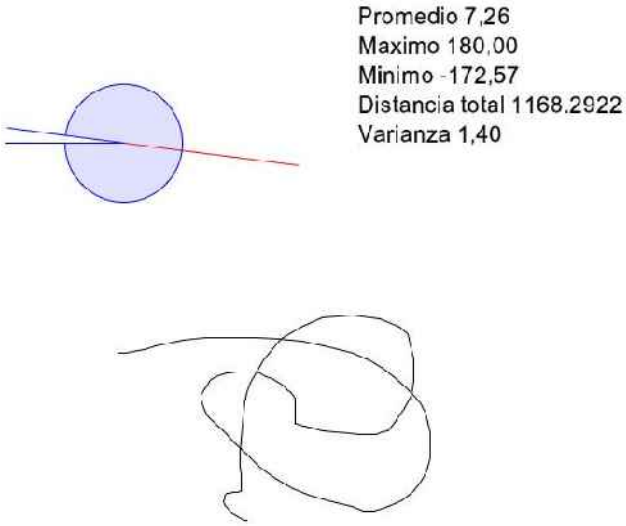
En la curva que sigue, los extremos están en un rango medio, pero su varianza sigue siendo alta en comparación con la primer curva:



La siguiente curva muestra un trazo casi recto pero en otra dirección:



Por último, se puede ver una curva sin dirección definida, que produce una varianza y extremos, muy altos:



Las gráficas y sus respectivos análisis, nos muestran que se pueden hacer las siguientes asociaciones:

1. Desplazamientos horizontales vs. desplazamientos verticales: pueden medirse en función del ángulo promedio, atendiendo que los extremos estén dentro de ciertos límites.
2. Desplazamientos zigzagueantes vs. desplazamientos directos: puede medirse con la varianza, nuevamente atendiendo a los extremos.

6. Conclusión

El texto presentado, expone los criterios de diseño adoptados para la realización de esta aplicación, demostrando los algoritmos involucrados en la resolución de la interpretación gestual. Queda pendiente para posteriores trabajos el diseño de una interface para la creación de estructuras musicales a partir de los resultados obtenidos con la notación presentada.

7. Referencias bibliográficas

[1] Gualterio Volpe (2003) *"Computational models of expressive gesture in multimedia systems"*, Ed. InfoMus Lab. University of Genova, Italia [Bibliografia]

[2] Lev Manovich, (1998) *"The Language of New Media"*, Ed. MIT-Press, USA [Bibliografia]

[3] www.processing.org

[4] mtg.upf.es/reactable/

(*) Estos capítulos están extraídos del artículo "Desarrollo de una Aplicación con Interfaces Tangibles" del mismo autor, fueron agregados (a pesar de la obvia duplicación) debido a la pertinencia de sus contenidos para el presente texto.