Trabajo de Fin de Máster
Máster Universitario en Sistemas Inteligentes
Julio, 2016

Algoritmos y Herramientas para Composición Automática de Melodías

Autor: Diego Milla de Castro
Tutora: Belén Pérez Lancho

Departamento de Informática y Automática
Universidad de Salamanca

Revisado por:

Belén Pérez Lancho - lancho@usal.es
María Navarro Cáceres -maria90@usal.es

Información del Autor:

Diego Milla de Castro
dmilla@usal.es
diego.milla26@gmail.com

Resumen

Esta investigación explora las posibilidades de la composición inteligente de melodías controlada mediante dispositivos de tipo *joystick*. Se estudia el uso de los dos ejes disponibles en estos dispositivos para dirigir el tono y la duración de las notas de la melodía generada. La investigación realizada parte de un conjunto de ficheros MIDI de los que se extraen las notas con sus respectivas duraciones para entrenar un modelo de Markov. Se propone un algoritmo de control para modificar las transiciones presentes en este modelo en función de la posición del dispositivo, y se genera una melodía en tiempo real en función de las probabilidades modificadas de estas transiciones. También se plantean diversos experimentos para estudiar el efecto sobre la composición de la melodía de las diferentes posibilidades ofrecidas por el entrenamiento del modelo y el algoritmo de control.

Índice

Índice de figuras

1. Introducción

Los diferentes avances computacionales y en el campo de la Inteligencia Artificial que han ocurrido a lo largo de los últimos años han llamado la atención de investigadores con todo tipo de orígenes y motivaciones, creando de este modo campos innovadores que unen conceptos aparentemente tan dispares como, por ejemplo, la Inteligencia Artificial y el Arte. De estas dos disciplinas nace el área de la Creatividad Computacional (también conocida como Creatividad Artificial), que puede ser vagamente definida como el análisis computacional y/o la síntesis de obras de arte, de un modo parcial o completamente automatizado [1]. Dada la particular naturaleza de este campo de investigación, que junta dos sectores con métodos y objetivos muy distintos (a veces incluso opuestos), el estado del arte es muy diverso y difícil de comparar.

Sin embargo, en los últimos años se está desarrollando de forma significativa el área de la Creatividad Computacional con la entrada de nuevos actores tan importantes como *Google*, con proyectos como *Deep Dream* ([2], una red neuronal inversa que transforma imágenes) o más recientemente este mes de Junio con el anuncio de *Magenta* ([3], el equivalente para la generación de música). Sin llegar al extremo de una generación completamente automatizada, también emerge la posibilidad de utilizar algoritmos para controlar la generación de arte mediante dispositivos de control que permitan indicar características de un nivel lo suficientemente abstracto para dirigir el proceso de creación. Es el caso, por ejemplo, de *MotionComposer* ([4]): una investigación de 2015 en la que utilizan un dispositivo para componer música a partir del movimiento de personas con discapacidad.

Aprovechando un dispositivo de tipo *joystick* utilizado previamente para otro tipo de investigaciones en el laboratorio de $I + D$ de la *USAL*, este proyecto explora las posibilidades de la generación inteligente de melodías controlada por este tipo de dispositivos. Se propone la utilización del dispositivo para controlar las variaciones de tono y duración de las notas de la melodía generada. Esta investigación requiere un conjunto de tareas muy variadas, por lo que sus objetivos son bastante diversos: seleccionar y procesar los datos de entrada, obtener un modelo que los represente, desarrollar un algoritmo de generación de melodías a partir de este modelo que pueda ser "dirigido" por el usuario, implementar el proceso de comunicación con el dispositivo, ofrecer una realimentación al usuario tanto visual como a través del propio dispositivo, y realizar diversos experimentos para estudiar los resultados obtenidos.

La metodología propuesta parte de unos ficheros MIDI ([5]) de referencia de los que se extraen las melodías para entrenar una cadena de Markov. Este modelo es el encargado de la parte "inteligente" de la generación melodías ya que es el que determina cuáles son las diferentes notas posibles para cada punto de la melodía. Dentro de estas posibilidades, la interacción mediante el dispositivo permite dirigir la generación de la melodía modificando las probabilidades de las notas presentes en la cadena de Markov. Se ha desarrollado una aplicación que permite experimentar con esta metodología, generando la melodía y dibujando dos gráficos en tiempo real para

poder visualizar los resultados. Esta memoria detalla todo el proceso de investigación realizado. Tras una breve introducción del estado del arte, se detallan las diferentes etapas de la metodología expuesta previamente. A continuación se incluye una serie de experimentos que permiten estudiar el efecto sobre la generación de la melodía de las diferentes posibilidades ofrecidas por el entrenamiento del modelo y el algoritmo de control. Por último, se comentan los resultados obtenidos y se tratan posibles líneas de investigación futuras.

2. Estado del Arte

Tradicionalmente la composición de música se caracteriza por la realización de una serie de tareas tales como la definición de la melodía y ritmo o la armonización entre otras muchas otras. Hoy en día, todas estas tareas pueden ser automatizadas por ordenadores en mayor o menor grado ([1]). En el caso de grados de automatización relativamente pequeños, han surgido lenguajes de programación específicos y herramientas gráficas que sirven de ayuda al compositor automatizando tareas específicas, u ofreciéndole una base sobre la que inspirarse. Existe una gran comunidad investigando y desarrollando software comercial en este área que se conoce como *"Composición Algorítmica Asistida por Ordenador"* (*CAAC*, del inglés *"Computer Aided Algorithmic Composition"*). Dentro de este campo cabe destacar el lenguaje *Csound* desarrollado por investigadores del *MIT* ([6]) y el lenguaje visual *MAX/MSP* ([7]). A nivel nacional, merece la pena destacar el proyecto *ReacTable* ([8]) de la *Universidad Pompeu Fabra* en el que exploran la posibilidad de generar música en directo componiendo en tiempo real utilizando una mesa y unas determinadas piezas como interfaz.

En lo que se refiere a la composición algorítmica con grados más altos de automatización, particularmente aquellos en los que el usuario no es considerado como la principal fuente de creatividad, se trata todavía de un campo de investigación más reducido ya que cuenta con pocos equipos de investigación trabajando en él. Además, dada la diversidad de los investigadores ya comentada en la introducción, se realizan trabajos muy diversos y cuyas metodologías varian de igual modo. Cabe destacar el congreso NIME (*New Interfaces for Musical Expression*) que, pese a tratarse de un evento que cubre un espectro más amplio, reúne a la gran mayoría de las investigaciones de este sector.

Acércandonos más a lo que concierne a este trabajo, la generación automática de melodías controlable mediante dispositivos externos, no se ha encontrado nada directamente relacionado. Lo más cercano podría ser la investigación [9] en la que se generan 3 melodías simultaneamente, controladas por 2 hamsters cada una. A diferencia de la metodología propuesta, el trabajo citado se basa en unas reglas predefinidas y no se entrena con un conjunto de datos. También merece la pena mencionar a *Pachet*, que ha trabajado intensivamente en la generación automática de melodías y en 2011 publicó un trabajo sobre la posibilidad de adaptar una cadena de Markov a ciertas restricciones predefinidas ([10]). Sin embargo, la metodología propuesta no es adecuada para un control interactivo en tiempo real ya que se

focaliza más bien en ciertas modificaciones puntuales para generar una secuencia más adaptada a un determinado estilo, y el coste computacional es relativamente alto. Dadas estas circunstancias, esta investigación ha sido realizada sin ninguna referencia directa por lo que se ha buscado un enfoque innovador y sencillo pero basándose en un modelo ya utilizado previamente para la generación automática de melodías: las cadenas de Markov.

3. Metodología

Para poder realizar esta investigación han sido necesarias diversas etapas que van desde la obtención y extracción de los datos hasta la visualización y el análisis de los resultados y que se explican en detalle en los diferentes apartados de esta sección. El esquema de la figura 1 permite hacerse una idea general de las diferentes etapas que se llevan a cabo para la generación de la melodía dirigida mediante el dispositivo. El primer paso es la obtención y el procesamiento de los datos de entrada que se detalla en el apartado 3.1 y, a continuación, se entrena una cadena de Markov de la forma especificada en la parte 3.2. En la subsección 3.3 se incluye toda la información relacionada con el dispositivo. El proceso de generación de la melodía a partir de la cadena de Markov y de la posición del dispositivo se trata en el apartado 3.4. Por último, en la subsección 3.5 se explica el proceso de realimentación y visualización de resultados.

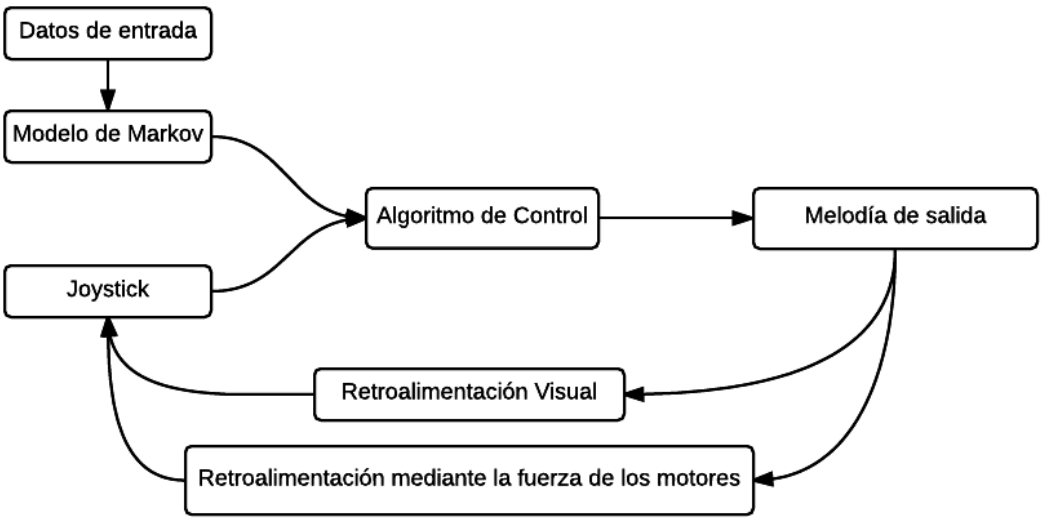

Figura 1: Generación inteligente controlable de melodías

Al tratarse de una investigación bastante particular en un campo muy heterogéneo, no existe ninguna herramienta disponible que ofrezca la libertad necesaria para realizar este proyecto por lo que ha sido necesario desarrollarla. De todas las

opciones disponibles, se ha elegido el lenguaje de programación *Scala* ([11]): un lenguaje multi-paradigma diseñado para expresar patrones comunes de programación de forma concisa, elegante y con tipos seguros que se ejecuta en la máquina virtual de *Java*. Podría considerarse una abstracción de *Java*, y es completamente compatible con este lenguaje, hasta el punto que se pueden incluír ficheros .java y .scala en un mismo proyecto o llamar a clases u objetos de *Java* desde *Scala*. El principal argumento detrás de esta decisión es la posibilidad de utilizar los completos paquetes MIDI de *Java* (*"javax.sound.midi"*) desde un punto de vista más abstracto que ofrece más posibilidades y permite un desarrollo más rápido que programando directamente en *Java*. Más concretamente, se ha utilizado el framework *Akka* ([12]) para desarrollar una aplicación distribuida, concurrente, tolerante a fallos y escalable mediante el uso de los actores disponibles en este framework. Aprovechando la flexibilidad disponible al implementar la herramienta desde 0, se ha podido integrar todo el proceso de investigación en una misma aplicación.

Se puede consultar el código fuente de la aplicación en *GitHub*, en la dirección `https://github.com/dmilla/kMMG-Final`. Si se desea probar (no es necesario el dispositivo, también se puede utilizar mediante el gráfico de la posición del control), se recomienda el uso del IDE *IntelliJ IDEA* con *Java 8* y los plugins *Scala* y *Maven* ya que es la herramienta que se ha utilizado para el desarrollo. Para añadir el proyecto al IDE, basta con elegir la opción *"Project from Version Control"* en el menú de nuevo proyecto y copiar el enlace al repositorio *Git* mencionado previamente. Para ejecutarlo, se debe crear una configuración de ejecución de tipo aplicación con *TFM.kMarkovMelodyGenerator.kMMGUI* como clase principal. Téngase en cuenta que se trata de una aplicación experimental con una interfaz de usuario muy básica y que puede generar errores, aunque en su mayoría el framework *Akka* se encarga de gestionarlos para que no afecten al funcionamiento de la aplicación.

3.1. Obtención y Procesamiento de Datos

Siendo el objetivo generar y controlar una melodía variando tanto el tono como el ritmo, se necesitan datos de entrada que representen estas dos características. Actualmente, no existe un formato que trate únicamente estas dos variables por lo que es necesario extraer los datos de otro tipo de archivos. Se ha decidido utilizar ficheros MIDI (*Musical Instrument Digital Interface*, [5]) ya que además de estar ampliamente disponibles contienen la información necesaria de forma bien estructurada y fácilmente accesible. Además, son archivos bastante ligeros ya que permiten codificar una canción completa en unos cientos de líneas, por ejemplo en algunos *kilobytes*. Esto se debe a que no registran el sonido en sí, ya que contienen las instrucciones (como si fuera una partitura) que permiten reconstruir la canción utilizando un secuenciador y un sintetizador que trabajen con las especificaciones MIDI. La figura 2 a continuación resume el proceso de obtención y procesamiento de datos que se detalla a lo largo de esta sección

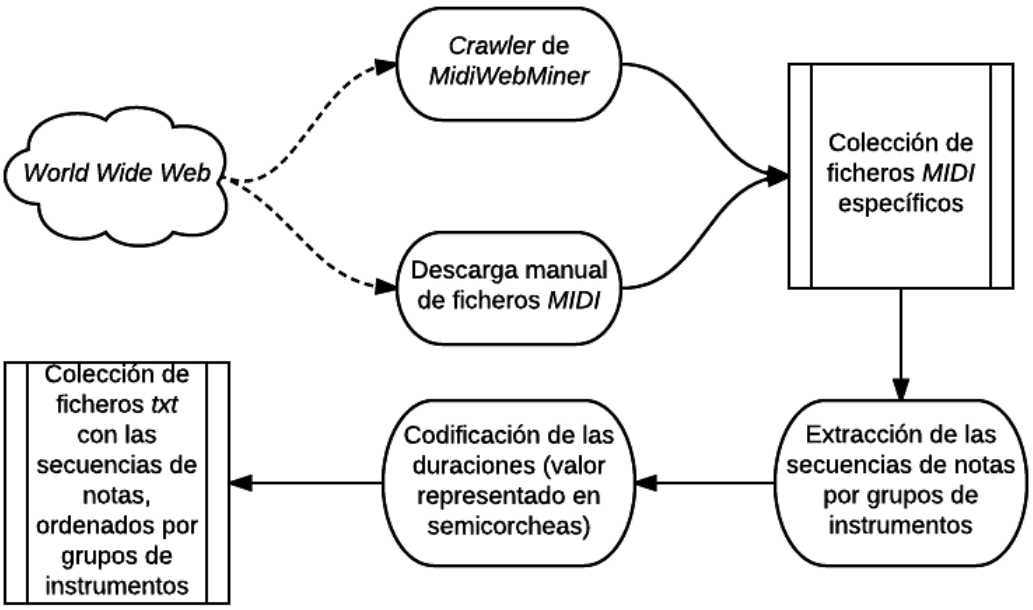

Figura 2: Obtención y procesamiento de datos

El primer paso es entonces obtener los ficheros MIDI de los que se va a extraer los datos para la investigación. La selección de los archivos con los que se va a trabajar depende del experimento que se quiera realizar, y puede variar desde una gran colección de ficheros variados y mezclados hasta un fichero con una canción o secuencia en particular. Cualquier MIDI puede formar parte de un experimento, pero para los casos en los que se desee experimentar con grandes cantidades de ficheros se ha desarrollado un *crawler* sencillo que permite descargar todos los ficheros MIDI que cumplan ciertas condiciones de una página web en concreto (nótese que se trata de un *crawler* experimental que no funciona en todas las páginas). Los detalles sobre el funcionamiento y la implementación del *crawler* no se incluyen en este informe ya que no está relacionado directamente con la investigación tratada en este documento. Sin embargo, si se desea, se puede consultar más información sobre la estrategia de *crawling* utilizada en el apéndice A

Una vez disponibles los ficheros MIDI con los que se va a experimentar, el siguiente paso es extraer la información necesaria para el proyecto: las notas y su duración. Según las especificaciones MIDI ([5]), cada fichero MIDI representa una secuencia (que se corresponde generalmente con una canción o composición) que a su vez se compone de una o varias pistas. Estas pistas se caracterizan por una sucesión de eventos MIDI (mensajes MIDI asociados a un tiempo en particular). Las diferentes pistas de la secuencias se reproducen simultáneamente, por lo que generalmente (aunque no tiene por que ser así, ya que también puede haber dos eventos simultáneos en la misma pista) se utilizan para instrumentos diferentes.

Dadas estas especificaciones del formato MIDI, se pueden extraer fácilmente los datos necesarios mediante unos pocos bucles. Primero, se itera entre todos los

ficheros MIDI objetivo para obtener la secuencia correspondiente. A continuación, se itera entre las diferentes pistas de la secuencia, y una vez dentro de cada pista se itera entre los diferentes eventos de cada una. Para este proyecto se necesitan las notas y su duración, por lo que nos interesan únicamente los eventos de tipo *NOTE_ON* y *NOTE_OFF* ([13]). Como su nombre lo indica, estos eventos sirven para determinar el inicio y el fin de una determinada nota, especificada en el primer byte del mensaje MIDI. El segundo byte del mensaje indica la intensidad de la nota (denominada "velocidad" en los mensajes MIDI), y para este proyecto se ignora excepto en el caso de que sea un mensaje de tipo *NOTE_ON* con velocidad 0, que es el equivalente a un mensaje de tipo *NOTE_OFF* ([14]). También se tienen en cuenta los eventos de tipo *PROGRAM_CHANGE*, que indican un cambio de instrumento, para extraer las notas por grupos de instrumentos ya que las melodías están destinadas a un instrumento en particular y de este modo se pueden seleccionar únicamente aquellas que correspondan con el grupo de instrumentos con el que se desee experimentar.

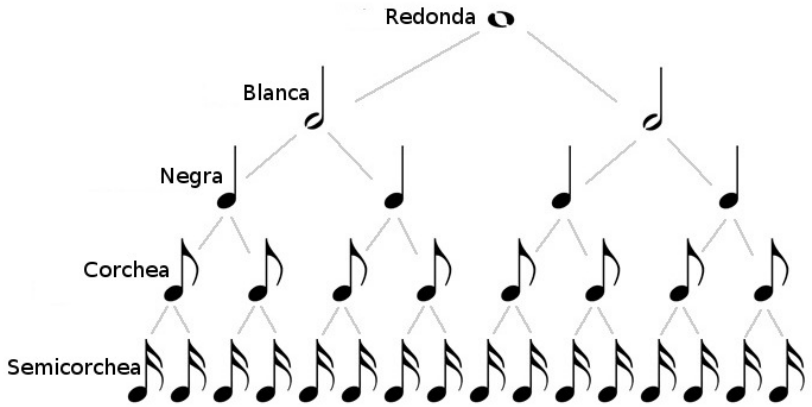

Figura 3: Duración de las notas

Para determinar la duración de una determinada nota, cada vez que se encuentra un evento *NOTE_ON* se itera entre los eventos siguientes hasta encontrar el evento *NOTE_OFF* (o *NOTE_ON* con velocidad 0) que determina su fin. Cada secuencia MIDI se caracteriza por una resolución que determina el número de ticks por nota negra (ver figura 3). Mediante la diferencia de la posición de ambos eventos en la pista, podemos determinar su duración en "ticks" MIDI. Para este proyecto la duración mínima de las notas es de una semicorchea, por lo que se normaliza la duración de cada nota a su duración más cercana en semicorcheas y se guarda como el número de estas. Las notas se guardan por su valor MIDI (un byte, es decir un número de 0 a 127 que se corresponde con las notas tal y como se puede ver en la figura 4). Los silencios (tiempos entre el final de una nota y el inicio de la siguiente) se consideran como una nota más, en este caso se les ha asociado el valor -1, y también se les asocia una duración en semicorcheas.

Octave	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-2	0	1	2	3	4	5	6	7	8	9	10	11
-1	12	13	14	15	16	17	18	19	20	21	22	23
0	24	25	26	27	28	29	30	31	32	33	34	35
1	36	37	38	39	40	41	42	43	44	45	46	47
2	48	49	50	51	52	53	54	55	56	57	58	59
3	60	61	62	63	64	65	66	67	68	69	70	71
4	72	73	74	75	76	77	78	79	80	81	82	83
5	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107
7	108	109	110	111	112	113	114	115	116	117	118	119
8	120	121	122	123	124	125	126	127				

Figura 4: Valores MIDI de las notas por octavas

Tras realizar todo este proceso, se exportan los resultados a una colección de ficheros de texto en los que se encuentran las secuencias de notas con sus respectivas duraciones en semicorcheas. Por ejemplo, una nota negra Do en la segunda octava se representaría "(48,4)": es decir, la nota 48 durante 4 semicorcheas. Si le añadimos una corchea Fa en la misma octava a continuación, la secuencia se representaría "(48,4) - (53,2)". De este modo se guardan los datos en ficheros fácilmente accesibles si se desean utilizar de nuevo sin necesidad de volver a realizar el proceso de extracción. Si se desea probar la aplicación, la parte *MidiWebMiner* accesible a través del botón de mismo nombre es la que permite descargar y extraer los datos. Se pueden consultar los detalles de la implementación de este proceso en el apéndice B.

3.2. Elección y Entrenamiento del Modelo

El siguiente paso en esta investigación consiste en obtener un modelo que represente los datos obtenidos y sirva para la generación de melodías. Las cadenas de Markov son una herramienta ampliamente utilizada para modelizar las propiedades temporales de diversos fenómenos, desde la estructura de un texto hasta fluctuaciones económicas. Al tratarse de modelos relativamente fáciles de generar, también se utilizan para aplicaciones de generación de contenido, como la generación de textos o de música (por ejemplo en [9]). El enfoque tradicional de los algoritmos de generación de secuencias de Markov suele ser muy rígido y de tipo "de izquierda a derecha"([15]), propiedades que los hacen fundamentalmente inadaptados a un control interactivo como el que se plantea en este proyecto. Sin embargo, las propiedades de las cadenas de Markov siguen siendo muy pertinentes para la generación inteligente de melodías, por lo que se ha decidido usar este tipo de modelo pero

utilizando un algoritmo de generación de secuencias diferente y adaptado al control interactivo que se explica en la sección 3.4.

Brevemente, una cadena de Markov representa un tipo especial de proceso estocástico en el que la probabilidad de que ocurra un evento (en este caso una nota o silencio con una determinada duración) depende únicamente del evento anterior (o N eventos anteriores, en función del orden del modelo). Esta característica de "falta de memoria" (o "memoria limitada") es lo que se conoce como *propiedad de Markov*. En la figura 5 a continuación se puede ver un ejemplo de una sencilla cadena de Markov con 2 estados y 2 transiciones posibles para cada uno, con sus respectivas probabilidades de transición.

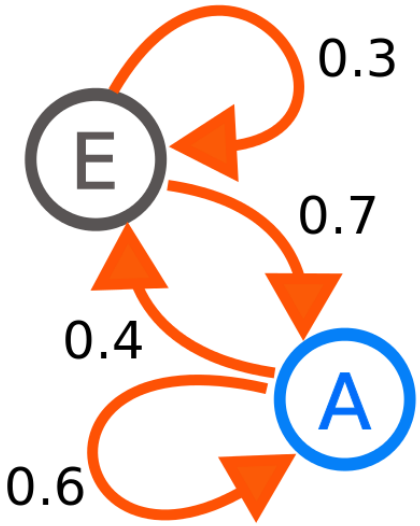

Figura 5: Ejemplo de una Cadena de Markov estable con dos estados

Por motivos relacionados con el control explicados en la sección 3.4, el modelo se entrena con datos normalizados en vez de con los datos tal y como se han extraído. Por un lado, las notas se normalizan a su equivalente en las tres primeras octavas (es decir, a un valor entre 0 y 35, respetando las equivalencias de notas mostradas en la figura 4). Para este proceso de normalización de las notas, se ha desarrollado un algoritmo que determina cuáles son las tres octavas consecutivas que reagrupan el mayor número de notas de los datos de entrada y que serán consideradas las octavas de referencia para los datos correspondientes. Las notas por debajo de esta referencia se trasladan a la octava de referencia inferior, y aquellas por encima a la octava de referencia superior. De este modo todas las notas se sitúan dentro de las octavas de referencia, que se normalizan a su vez a su valor correspondiente en las tres primeras octavas para mayor simplicidad y generalización en el código. Sin embargo, la melodía se generará a continuación respetando las octavas de referencia, ya que también se ha implementado un parámetro de normalización de la melodía de salida que se ajusta automáticamente en función de la normalización de los datos de entrada.

Por otro lado, las duraciones se normalizan a 8 duraciones posibles: semicorchea, corchea, corchea con puntillo, negra, negra con puntillo, blanca, blanca con puntillo y redonda (un puntillo equivale a multiplicar por 1,5 la duración de la nota). De este modo las 8 duraciones posibles expresadas en semicorcheas son 1, 2, 3, 4, 6, 8, 12 y 16.

La cadena de Markov se construye a continuación mediante un sencillo proceso de entrenamiento. Lo primero es seleccionar el grupo de instrumentos del mismo tipo con el que se va a experimentar (pianos, guitarras, bajos, etc...), ya que después del paso anterior los datos extraídos están organizados de este modo. Después se itera entre todas las secuencias de notas con duración extraídas para el grupo de instrumentos seleccionado, actualizando constantemente el estado de la cadena de Markov según va iterando el algoritmo y añadiendo progresivamente cada transición a su estado correspondiente. Una vez finalizado el bucle, para asegurar la estabilidad de la cadena de Markov (es decir, que exista al menos una transición para cada estado), se sigue iterando desde el principio de los datos (pero esta vez con las notas del final de la secuencia como estado) un número de veces igual al orden del modelo que se esté entrenando.

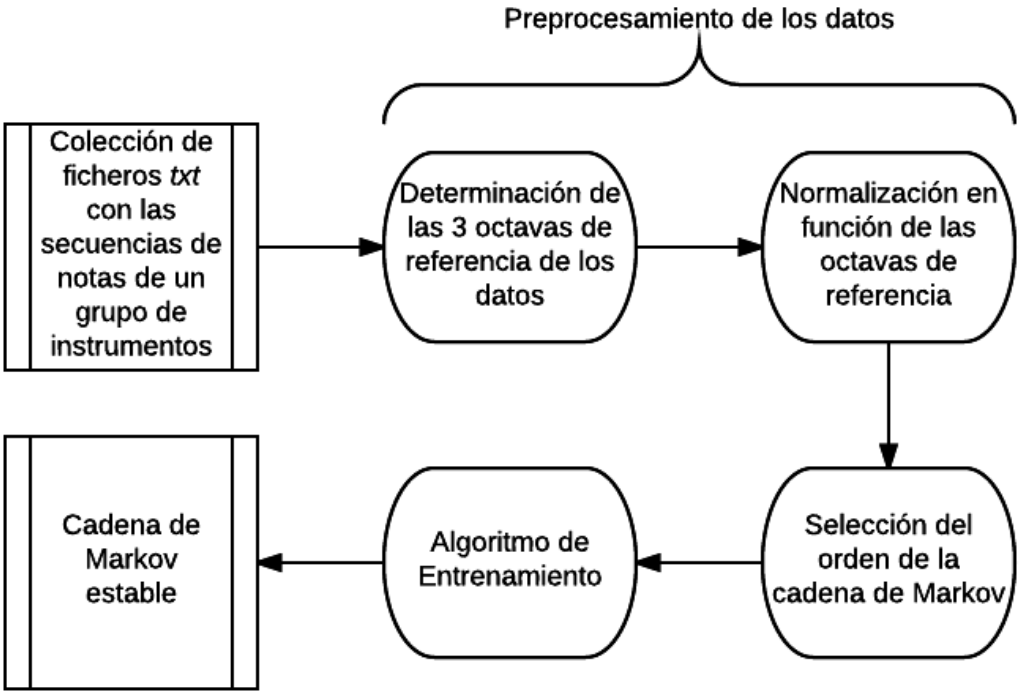

Figura 6: Entrenamiento de la cadena de Markov

El esquema de la figura 6 muestra las diferentes etapas expuestas previamente. Todo este proceso ha sido implementado manualmente, inspirándose de la lógica y el código abierto explicados en [16] y adaptando el ejemplo propuesto para añadir

la posibilidad de generar modelos de orden superior a 1. Los detalles de la implementación se pueden consultar en el apéndice C.

3.3. Dispositivo

Es conveniente explicar el dispositivo que se ha utilizado antes de entrar en los detalles del algoritmo de generación de melodías y de control. Se ha utilizado el dispositivo de tipo *joystick* disponible en el laboratorio del centro de I+D de la *USAL* que se puede ver en la figura 7. El principal investigador detrás de este dispositivo es el profesor Wataru Hashimoto del OIT (*Osaka Institute of Technology*), aunque también se han realizado investigaciones con colaboradores de la *USAL*. Técnicamente, el dispositivo está compuesto por 2 motores anclados a un marco de aluminio e interconectados formando un pantógrafo con 2 grados de libertad. Una de las ventajas de esta configuración es que los motores pueden ejercer una fuerza de retroalimentación que puede ser de utilidad para el usuario en diversas aplicaciones, como en el caso de este proyecto por ejemplo.

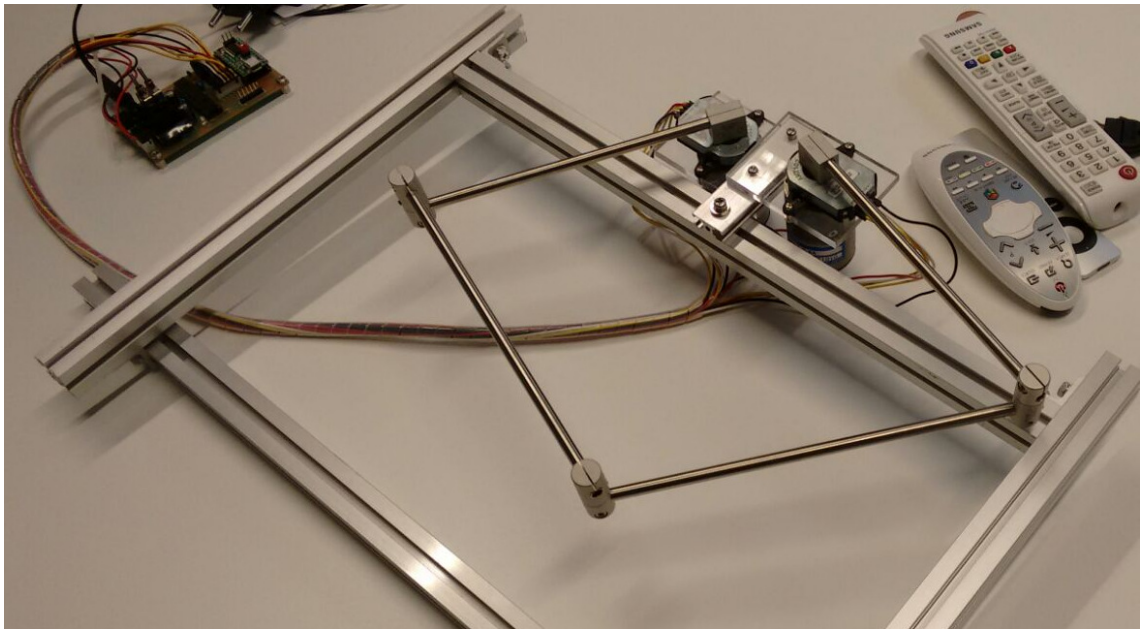

Figura 7: Dispositivo tipo *joystick*

Este dispositivo consta de dos motores y está construido de tal forma que cada par de ángulos de entrada de los motores $(\theta 1, \theta 2)$ se corresponde con un punto en particular en el plano esquematizado en la figura 8 a continuación:

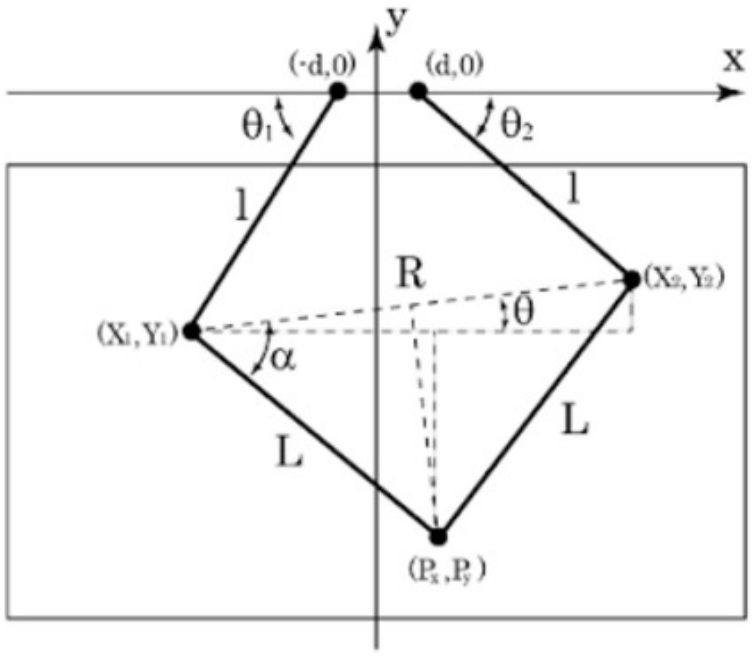

Figura 8: Plano de coordenadas del dispositivo

Gracias a este esquema podemos observar que, dados la distancia desde el centro hasta los motores (d) y la longitud de los "brazos" superior e inferior (l y L respectivamente), el sistema forma un conjunto de triángulos determinados por los ángulos $(\theta 1, \theta 2)$. Esto permite la obtención de las coordenadas (Px, Py) mediante el sistema de ecuaciones trigonométricas presentado en la figura 9. Para aplicar una fuerza de retroalimentación se ha traducido a *Scala* el código *C++* de un proyecto con el mismo dispositivo realizado anteriormente que permite calcular los valores que se envían a los motores para simular un determinado vector de fuerza en función de la torsión ejercida.

$$X_1 = -d - l\cos\theta_1 \quad (1)$$

$$Y_1 = -l\sin\theta_1 \quad (2)$$

$$X_2 = d + l\cos\theta_2 \quad (3)$$

$$Y_2 = -l\sin\theta_2 \quad (4)$$

$$R = \sqrt{\left((X_1 - X_2)^2 + (Y_1 - Y_2)^2\right)} \quad (5)$$

$$\theta = \tan^{-1}\left(\frac{(Y_1 - Y_2)}{(X_1 - X_2)}\right) \quad (6)$$

$$\cos\alpha = \frac{R/2}{L} = \frac{R}{2L} \quad (7)$$

$$\alpha = \cos^{-1}\left(\frac{R}{2L}\right) \quad (8)$$

$$P_x = X_1 + L\cos(\alpha - \theta) \quad (9)$$

$$P_y = Y_1 - L\sin(\alpha - \theta) \quad (10)$$

Figura 9: Sistema de ecuaciones para la obtención de coordenadas

Para comprender mejor el funcionamiento del dispositivo y poder desarrollar la aplicación sin necesidad de ir siempre al laboratorio, se ha construido una simulación utilizando la herramienta EJS ([17]). Debido a la construcción del dispositivo, el espacio de coordenadas que se pueden alcanzar no es cuadrado o rectangular como se necesita en este proyecto. En las figuras 10, 11, y 12 se puede observar esta particularidad: en las dos primeras se alcanzan puntos inferiores a -45 y -55 en los ejes X e Y respectivamente, mientras que en la última (en la que se intenta alcanzar la esquina inferior izquierda) el brazo derecho se extiende del todo y no permite alcanzar los valores de las dos primeras. La simulación mediante EJS ha permitido analizar fácilmente el espacio de trabajo del dispositivo y determinar la estrategia adecuada para solucionar este problema: transformarlo en un espacio rectangular tal y como se explica más adelante en esta misma sección.

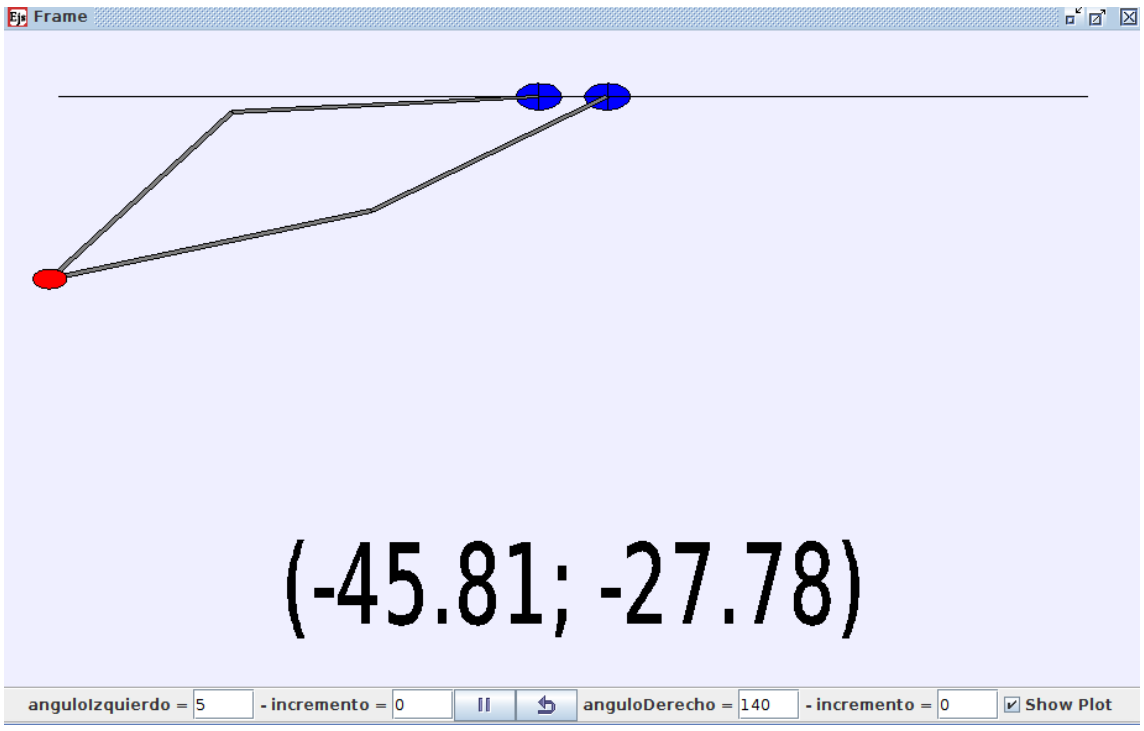

Figura 10: Simulación mediante EJS 1

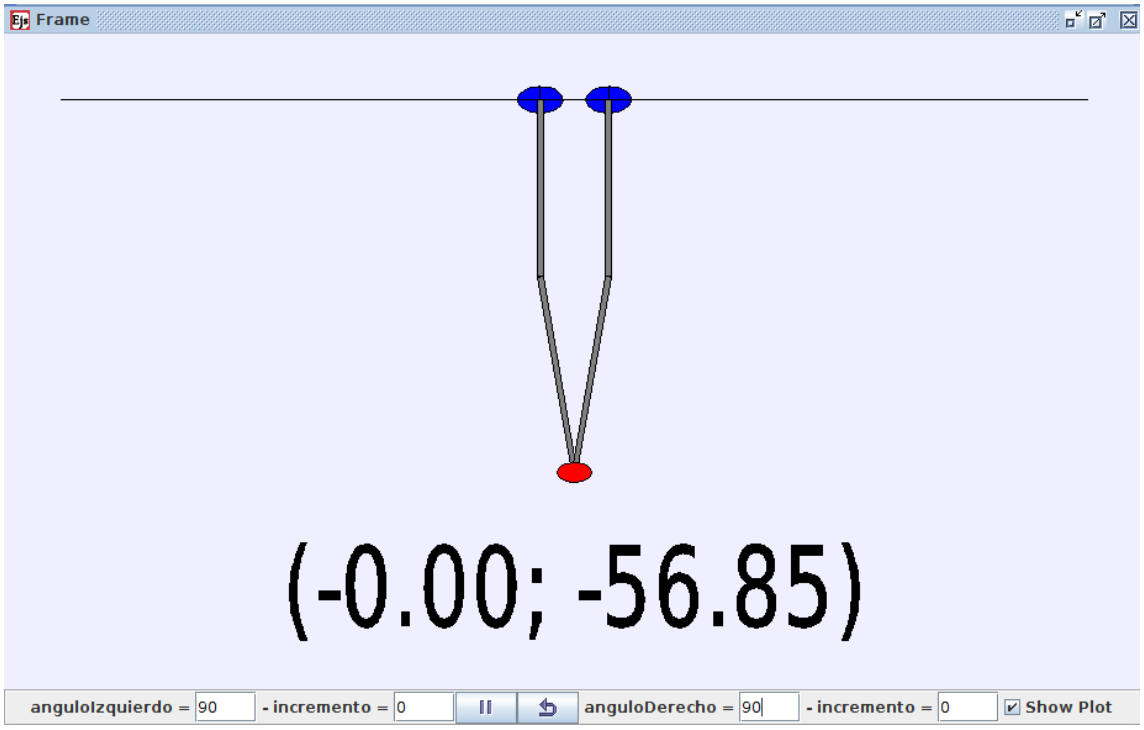

Figura 11: Simulación mediante EJS 2

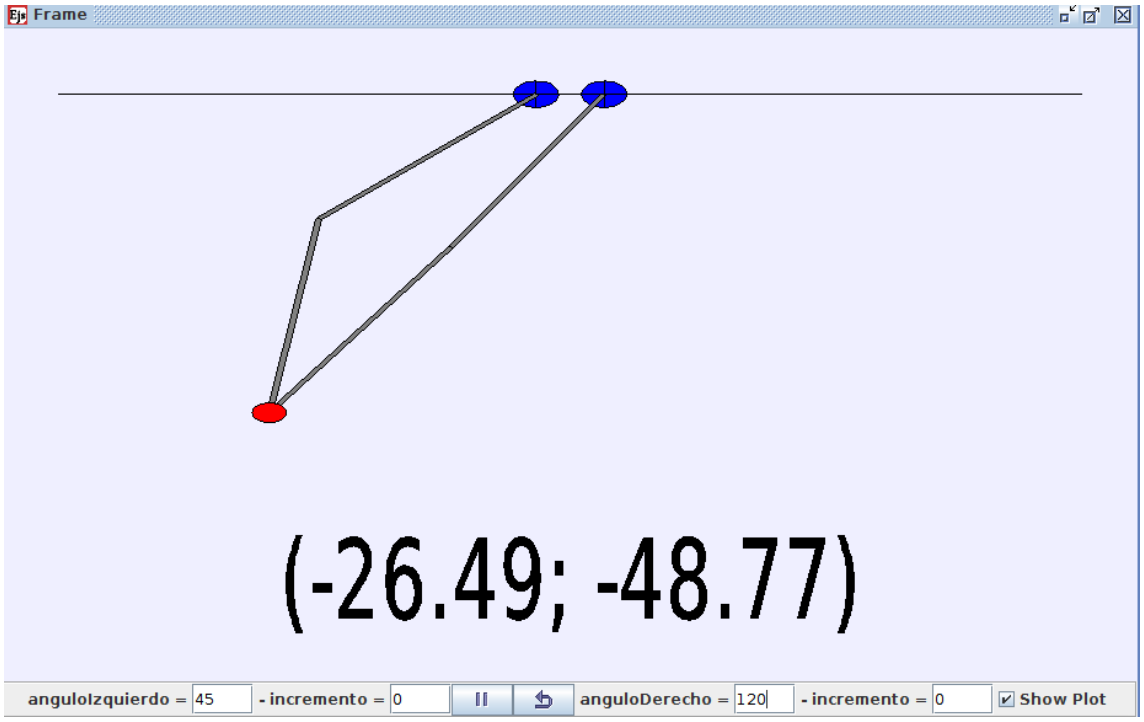

Figura 12: Simulación mediante EJS 3

Este espacio de coordenadas tan particular que alcanza el dispositivo no se puede emplear para este proyecto ya que es necesario un espacio rectangular en el se puedan

alcanzar todas los puntos (por los motivos explicados en la sección 3.4). Además, en la práctica, algunos puntos del espacio original también son dificilmente accesibles debido a la construcción del dispositivo (choques con el aluminio del marco, con los soportes de los motores, etc...). Para solucionar esto, se ha reducido el espacio de coordenadas del dispositivo a (-26, 26) para el eje X y (-35, -2) para el eje Y, considerando todos los valores fuera de estos rangos iguales al máximo o mínimo del rango que corresponda.

Desde un punto de vista más técnico, el dispositivo utiliza una conexión serie rs232 mediante un adapatador de USB a RS232. El buffer del dispositivo consta de 4 bytes, de los cuales los 2 primeros corresponden al motor izquierdo y los 2 últimos al motor derecho. Cada motor envía el ángulo de salida del brazo correspondiente, con un valor entre 0 y 4096, y recibe un valor entre -20000 y 20000 que indica la torsión ejercida. Se comunica con el dispositivo mediante un proceso constante de lectura/escritura cada milisegundo, por los motivos relacionados con la fuerza de realimentación de los motores detallados en la subsección 3.5. Toda la lógica relacionada con la conexión se ha implementado directamente en *Scala*, utilizando la librería *Flow* ([18]).

3.4. Generación de Melodías y Algoritmo de Control

3.4.1. Visión General

Para la generación inteligente y controlada de melodías, se propone un enfoque híbrido en el que el algoritmo que genera la melodía tiene en cuenta tanto la cadena de Markov como la posición del dispositivo. La figura 13 mostrada a continuación resume las diferentes etapas relacionadas con el algoritmo de control y la generación de melodías que se detallan a lo largo de esta subsección.

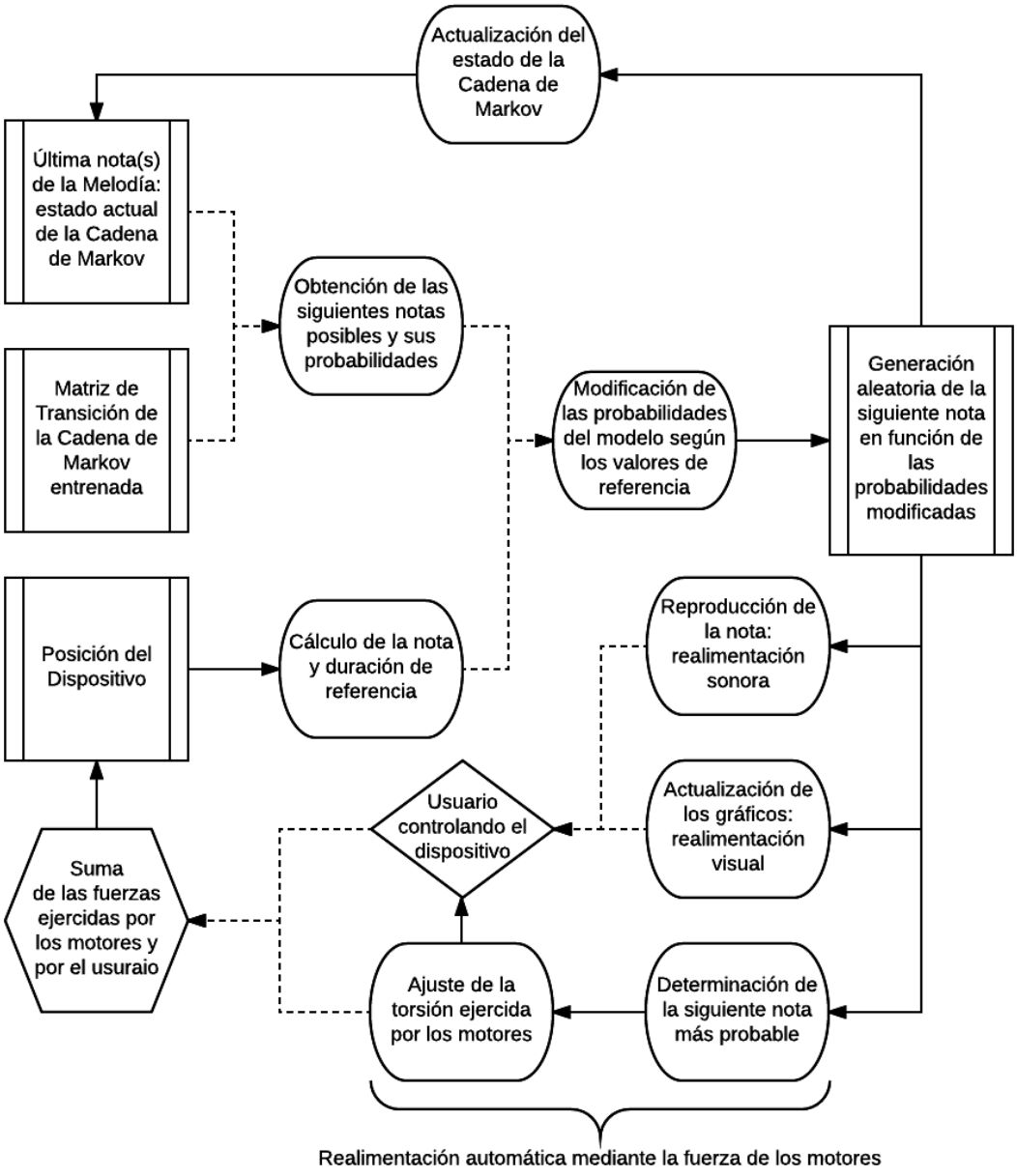

Figura 13: Esquema del algoritmo de control

El componente principal de la composición de las melodías es la cadena de Markov, ya que es la que determina cuáles son las transiciones posibles para cada estado (nunca sonará una transición que no esté presente en el modelo) y las probabilidades iniciales de cada una de estas transiciones. A continuación, en función de la posición del dispositivo (tal y como se explica en 3.4.2), se ajustan estas probabilidades de la manera que se explica en detalle más adelante en la subsección 3.4.3.

Después de este paso, se tratan las probabilidades de las transiciones restantes como una distribución estadística y se calcula la nota de salida generando una ob-

servación "aleatoria" en función esta distribución. Tras la generación de cada nota, se ofrece una realimentación al usuario tanto visual como mediante la fuerza de los motores del dispositivo, que se explica en detalle en la subsección 3.5.

3.4.2. Posición del Dispositivo

Antes de poder realizar cualquier tipo de control, es indispensable traducir la posición del dispositivo en una nota y una duración que se podrían denominarse nota y duración "de referencia". El dispositivo trabaja con un espacio de 2 dimensiones (ejes X e Y), por lo que se propone utilizarlas para dirigir las notas y sus respectivas duraciones que van generando la melodía. Se ha decidido qu el eje Y permita indicar la nota de referencia (más aguda o más grave), ya que intuitivamente se asocia "subir" con notas más agudas y "bajar" con notas más graves. A su vez, el eje X permite indicar una duración de referencia. Los silencios también se controlan mediante el eje Y, ya que al estar también asociados con una duración no se pueden controlar con el eje X (que se encarga en este caso de controlar la duración del silencio).

Para que todos los puntos de referencia sean accesibles mediante el uso del dispositivo, se ha tenido que reducir el espacio de coordenadas del dispositivo transformándolo en rectangular, tal y cómo se explica en 3.3. Una vez delimitadas las coordenadas que se van a emplear, y para mayor generalización del algoritmo de control, se normalizan ambos ejes a valores comprendidos entre 0 y 1. En el eje X, 0 representa la nota más corta (semicorchea) y 1 representa la nota más larga (redonda). En el eje Y, 0 representa silencio y luego se representan 3 octavas de referencia (que se explican en 3.2), desde la más grave hasta la más aguda. Estas necesarias delimitaciones del espacio de referencia justifican el proceso de normalización de las notas a la hora de entrenar el modelo.

También se ha añadido otra particularidad al control: la posibilidad de acortar notas demasiado largas desde el punto de vista del usuario. La lógica detrás de esto es muy sencilla: si se ha generado un nota larga y se desplaza el *joystick* hacia la izquierda en el eje X (hacia el valor 0, es decir hacia notas más cortas), en el momento en el que la duración indicada por la referencia sea inferior a la duración transcurrida desde el inicio de la nota, la nota actual se "corta" y se adelanta el cálculo de la siguiente.

3.4.3. Controlando la Cadena de Markov

Una vez traducida la posición del dispositivo en una nota y una duración de referencia, el enfoque propuesto en este proyecto consiste en modificar las probabilidades de las diferentes transiciones posibles según de la cadena de Markov en función de estos valores. Estas modificaciones se realizan de dos formas complementarias: por un lado se aumenta la probabilidad de las transiciones más cercanas al valor de referencia, y por otro se evitan todas aquellas transiciones demasiado alejadas de este valor.

La probabilidad de las transiciones cercanas al control se aumenta en función

de un parámetro "k" que determina el peso de la referencia a la hora de calcular la probabilidad de cada transición t según la formula siguiente:

*P(t) = k * P(t según referencia) + (1-k) * P(t según cadena de Markov)*

La probabilidad de una transición según el control se calcula en función de la distancia entra esa transición y la transición de referencia. Como cada transición se caracteriza por una nota y una duración, existen dos distancias para cada transición: la distancia de la nota y la distancia de la duración. La distancia se ha definido como el valor absoluto entre la nota de referencia (o la duración de referencia) y la nota (o duración) de una determinad transición.

Para una transición exactamente igual que la de referencia, se ha decidido que la probabilidad según la referencia es 0.4. Para todas las transiciones posibles igual a un único valor de referencia, nota o duración, y cuya distancia respecto al valor que no comparten es 1 (nota o duración anexa), la probabilidad de referencia se ha determinado como 0.1. Por último, si la distancia tanto de la nota como de la duración es 1 (nota y duración anexas a los valores de referencia), la probabilidad de referencia se ha establecido a 0.05. De este modo, si existen todas las transiciones anexas a la transición de referencia, la suma de las probabilidades de referencia sería igual a 1 (0.4 de la transición de referencia, más 0.1 * 4 de las 4 transiciones anexas que comparten un único valor, más 0.05 * 4 de las 4 transiciones anexas a la referencia pero que no comparten ningún valor). Como también se puede dar el caso de que alguna de estas transiciones no exista, se ha tenido en cuenta en la generación de la melodía que la suma de las probabilidades puede ser inferior a 1.

Por otro lado, también se eliminan las transiciones que se alejan demasiado de los valores de control. Esta limitación del rango de salida se realiza mediante dos parámetros ajustables que determinan la distancia máxima de las notas y de la duración de las transiciones de salida respecto a los valores de referencia. Las distancias utilizadas son las mismas que las explicadas en el párrafo anterior, y simplemente se eliminan todas aquellas transiciones cuya distancia al control es superior a la especificada en el parámetro correspondiente. Existe una única excepción a esta regla: los silencios siempre se consideran dentro del rango de control de las notas (no de la duración), para evitar modificar innecesariamente el ritmo de la melodía generada. Al igual que en el ajuste anterior, estas modificaciones pueden dar lugar a casos en los que la suma de las probabilidades de las transiciones restantes sea inferior a 1, pero esto se ha tenido en cuenta para el cálculo de la transición siguiente en función de estas probabilidades. Nótese que en el caso de que ninguna transición se encuentre dentro del rango de control (es decir, que todas las transiciones posibles estén demasiado alejadas del punto de control) se utilizan las probabilidades originales de la cadena de Markov para evitar que la generación de melodía se pare.

Los algoritmos incluídos en el apéndice D son los que se encargan de calcular las probabilidades finales para cada transición, y resumen todo lo expuesto anteriormente en esta subsección. Después de este proceso, la siguiente transición se calcula aleatoriamente respetando las probabilidades obtenidas después de modificar las transiciones de la cadena de Markov en función de la posición del dispositivo.

3.5. Visualización de la Melodía y Realimentación

Para facilitar la visualización de la generación de la melodía se han construído dos gráficos dinámicos. El primero, que se puede ver en la figura 14, muestra la posición actual del *joystick* en el espacio normalizado entre 0 y 1. El fondo de este gráfico se rellena de forma dinámica, mostrando todas las transiciones posibles en función del estado actual de la cadena de Markov y coloreándolas en función de su probabilidad teórica (sin tener en cuenta la posición del dispositivo). Si la probabilidad de la transición es inferior al 2 %, el espacio correspondiente se colorea en rojo. Cuando la probabilidad es superior al 2 % pero inferior al 5 %, se colorea en naranja, y si es superior al 5 % pero inferior al 10 % se rellena de amarillo. Por último, el espacio correspondiente a aquellas transiciones con una probabilidad superior al 10 % se muestra en el color verde. De este modo el usuario puede hacerse una idea de un vistazo sobre las transiciones posibles, y sus respectivas probabilidades, lo que le permite realizar un control más pertinente.

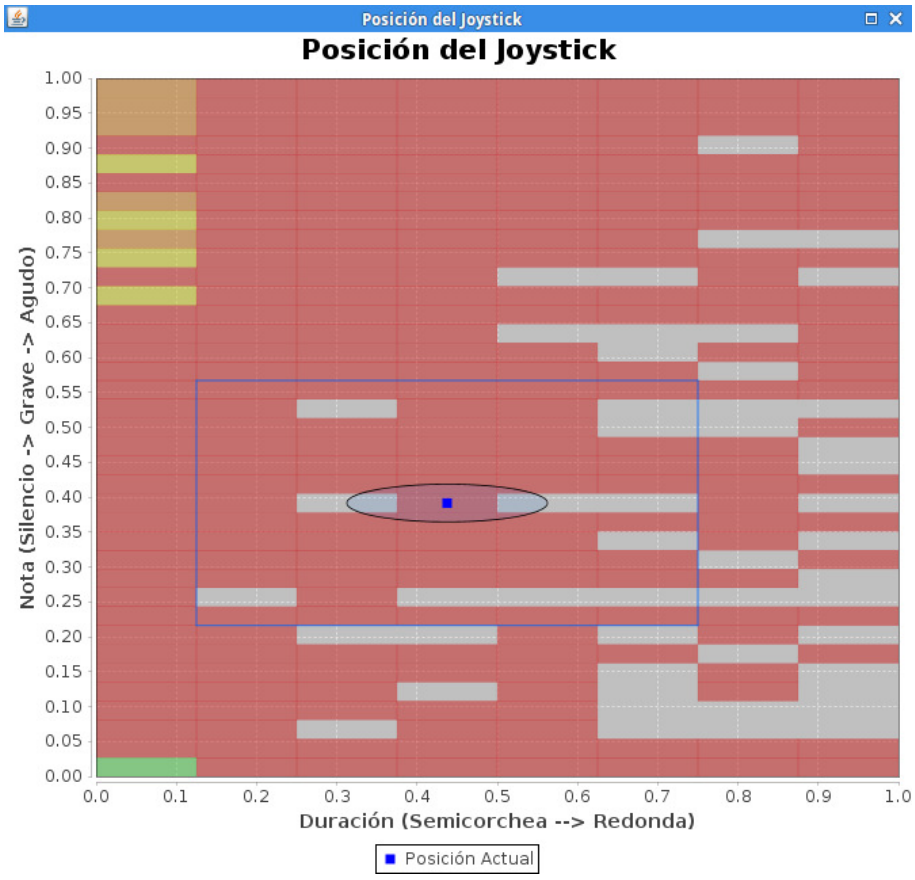

Figura 14: Posición del dispositivo

También se incluye una elipse que se dibuja sobre el espacio que corresponde a aquellas transiciones cuya probabilidad se aumenta por el control, y se delimita mediante un rectángulo azul el espacio en el que entran las transiciones permitidas por el control (en función de las distancias máximas explicadas en la subsección 3.4.3). El objetivo es, junto al fondo con las transiciones coloreadas, que el usuario pueda

saber fácilmente si el control va a afectar o no a la melodía y si está dirigiéndola hacia espacios más o menos probables. Además de servir de ayuda para la visualización del proceso, este gráfico también se puede utilizar para dirigir la generación de la melodía cuando no está conectado el dispositivo. Para ello, basta con hacer clic sobre el punto del gráfico hacia el que se desee mover el punto de referencia.

Por su lado, el segundo gráfico (ver figura 15) muestra un histograma con las notas y la duración de referencia en función del tiempo, utilizando como unidad el tick de la secuencia MIDI (eje horizontal del gráfico). El histograma de las notas de referencia se representa mediante la línea roja en la escala normalizada MIDI (valor entre -1 y 35, siendo -1 el valor que representa los silencios). El histograma de las duraciones de referencia se representa a su vez mediante la línea azul, utilizando el número de semicorcheas de duración como escala. A medida que se va generando la melodía, se va representando dinámicamente en tiempo real en el fondo de este gráfico mediante una serie de rectángulos amarillos que representan las notas generadas. La posición del rectángulo en el eje vertical indica la nota (según el mismo eje que para el histograma de la nota de referencia), y su anchura se corresponde con su duración en semicorcheas (en este proyecto, la resolución MIDI está configurada de forma que un tick corresponde con una semicorchea). Si una nota se corta manualmente de la forma explicada en 3.4.2, este evento se representa mediante una barra vertical roja sobre el rectángulo de la nota que se ha cortado.

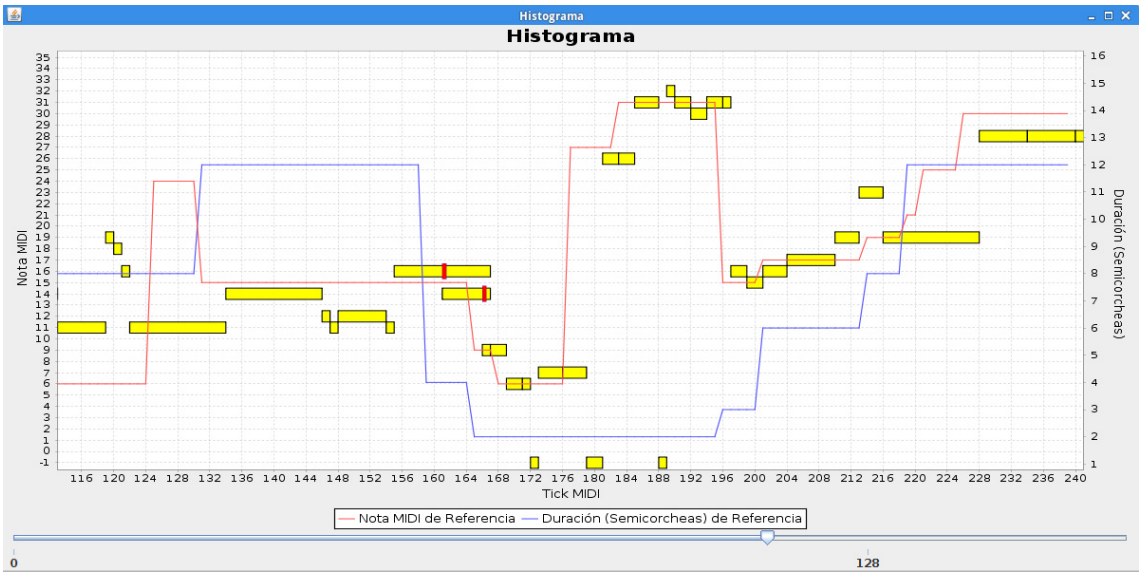

Figura 15: Histograma de la referencia y melodía generada

El objetivo de este segundo gráfico es facilitar la visualización de la melodía generada y el impacto de la referencia sobre su evolución. Por ejemplo, en la figura 15 podemos observar que las notas generadas siguen a la nota de referencia, y que según se modifica la duración de referencia sus duraciones también se adaptan a esta referencia. Al tratarse de un gráfico que se genera en tiempo real durante un tiempo indefinido, se ha implementado la posibilidad de desplazar el gráfico horizontalmente mediante un *slider* en la parte inferior por si se desea consultar alguna parte

anterior. En caso de que se requiera una visualización más compleja de la melodía, o simplemente se desee exportar el resultado, también se ha incluído la posibilidad de exportar la melodía generada en formato MIDI.

También se propone una realimentación mediante la fuerza de los motores del dispositivo. El objetivo es que el dispositivo tienda a desplazarse automáticamente hacia el punto correspondiente con la la transición (nota y duración) más probable en función el estado actual. Para ello, se calcula un vector de fuerza en el espacio (x, y) de forma proporcional a la distancia entre la posición actual del dispositivo y la posición correspondiente a esta transición objetivo. Se trata de un algoritmo de control clásico, por lo que también es necesario aplicar una constante proporcional directa multiplicando el vector resultante por un factor de "ajuste" que permite que la fuerza de realimentación sea más o menos notable. Valores muy pequeños de esta constante se traducen en una realimentación insignificante, y valores demasiado grandes generan movimientos demasiado bruscos que provocan que el dispositivo oscile alrededor de la posición objetivo o que incluso uno de los brazos llegue a impactar con el soporte del motor opuesto.

Para evitar estas ocurrencias, ya que pueden dañar el dispositivo, la fuerza de realimentación se ha ajustado a un valor relativamente sutil, y se recomienda que siempre que se manipule el dispositivo no se deje de sujetar hasta que no se haya finalizado la experimentación. La frecuencia del proceso de lectura/escritura de los datos del dispositivo ha sido crítica para el correcto funcionamiento de esta funcionalidad, ya que en un principio era de 10ms y el dispositivo mostraba saltos muy bruscos incluso con valores de fuerza pequeños. Aumentarla a 1ms permitió resolver este problema, ya que en cada iteración del bucle de lectura/escritura se recalcula la fuerza que se debe aplicar a cada motor en función de la nueva posición del dispositivo.

Mediante esta realimentación directamente a través del dispositivo, se ofrece una realimentación "intuitiva" al usuario sobre el control que está realizando: si se deja llevar y desplaza el dispositivo hacia los puntos que menos resistencia presentan, estará generando una melodía más en la línea de lo establecido por el modelo. Sin embargo, si intenta desplazar el dispositivo hacia puntos cuyas transiciones correspondientes sean menos probables, se dará cuenta rápidamente ya que la fuerza que debe ejercer para realizar ese desplazamiento será más importante.

3.6. Parámetros Disponibles

Para finalizar el apartado de *Metodología*, se incluye una breve recapitulación de los diferentes parámetros disponibles para la experimentación. El primer y más importante factor que determinará la melodía generada es, naturalmente, el conjunto de ficheros MIDI que se utilizará como datos de entrada. De estos datos depende directamente el entrenamiento la cadena de Markov, y por consecuencia las transiciones que serán posibles o no a la hora de genera la melodía. Otro factor de gran importancia, casi tanto como el primero, es el orden (número de estados anteriores de los que depende la siguiente transición) de la cadena de Markov que se utilizará

para entrenar el modelo. Por regla general según aumenta el orden el modelo se vuelve más complejo y la melodía generada debería ser más elaborada.

También se puede experimentar modificando los parámetros relacionados con el control. El peso de la referencia, "k", puede ser modificado a cualquier valor entre 0 y 1: un valor de 0 indica que las probabilidades de la cadena de Markov no se verán modificadas por la referencia, mientras que un valor de 1 indica lo contrario y en ese caso las probabilidades de la cadena de Markov serán ignoradas. También se pueden ajustar otros dos parámetros relacionados con el control: la distancia máxima entre la nota de referencia y la nota de salida, y la distancia máxima entra la duración de referencia y la duración de salida. Estos parámetros sirven para delimitar el rango de transiciones posibles a únicamente aquellas dentro del rango de control indicado por estos parámetros que se ha explicado en la subsección 3.4.3.

Por último, también se ha implementado la posibilidad de realizar algunos ajustes que afectan indirectamente a la generación de la melodía: el *tempo*, la normalización de salida, y el instrumento de reproducción. El *tempo* indica la velocidad de reproducción en BMPs (*Beats Per Minute*). La normalización de salida indica cuál es la octava mínima en la que se va a reproducir la melodía generada, y se ajusta por defecto para corresponder con las octavas de referencia para los datos de entrada según el proceso explicado en 3.2. El instrumento de salida se puede ajustar a cualquiera de los especificados en el estándar MIDI ([19]), indicando el valor del mensaje *PROGRAM_CHANGE* correspondiente al instrumento deseado. Este último ajuste se ha incluído principalmente para poder reproducir la melodía con un instrumento similar al de los datos de entrada, que tal y como se explica en la sección 3.1 se clasifican en función al grupo de instrumentos que corresponden.

4. Experimentos

A lo largo de esta sección se exponen los diferentes experimentos que se han realizado para estudiar el efecto sobre la generación de la melodía de las diferentes posibilidades ofrecidas por el entrenamiento del modelo y el algoritmo de control. Para empezar se estudia el impacto del orden de la cadena de Markov sobre la generación de la melodía. Una vez determinado el impacto del orden del modelo, se experimenta entrenándolo con datos de diferentes estilos y con diferentes tonalidades dentro del mismo estilo. Por último se muestra cómo afectan los diferentes parámetros relacionados con el algoritmo de control.

4.1. Orden del Modelo de Markov

El primer experimento propuesto consiste en estudiar el impacto del orden de la cadena de Markov sobre la melodía generada. Por este motivo se han ajustado los parámetros relacionados con el control de forma que la melodía no se vea afectada por la posición del dispositivo (peso de la referencia, k, igual a 0 y la distancia máxima a la referencia al mayor rango posible para que no se elimine ninguna transición).

También se han elegido unos datos de entrada completamente aleatorios, partiendo de un total de aproximadamente 2000 ficheros MIDI seleccionados al azar sobre los que se han extraído las melodías según el procedimiento explicado en 3.1.

A continuación, se han ido entrenando diferentes cadenas de Markov (utilizando las secuencias extraídas del grupo "Piano") empezando por el de primer orden e incrementándolo uno a uno hasta llegar a 8. Para poder analizar mejor estos modelos, después de cada entrenamiento se ha calculado el número de estados diferentes presentes en cada uno y el porcentaje de esos estados para los que existe más de una transición posible (es decir, que se puedan ver afectados por el usuario). Los resultados, que se pueden ver en la tabla incluída a continuación, muestran que según aumenta el orden del modelo, el número de estados con más de una transición posible disminuye notablemente. En el contexto de este proyecto, esto implica que un modelo de orden superior al tercero o al cuarto vería muy reducidas las posibilidades de que la melodía se vea afectada por la referencia marcada por el usuario. Por ejemplo, si se quisiera utilizar un modelo de cuarto orden con estos datos aleatorios, el usuario únicamente podría afectar a la melodía generada aproximadamente un 15 % del tiempo, siendo el resto del tiempo generada de forma completamente automática.

Orden del Modelo	Número de Estados	Estados con más de una transición posible
1	296	100 %
2	48 518	68 %
3	306 668	28 %
4	595 106	15 %
5	800 717	9 %
6	931 516	6 %
7	1 022 351	5 %
8	1 091 844	4 %

Sin embargo, los modelos de orden superior también parecen generar melodías más elaboradas. Esto se puede explicar por la mayor "memoria" de la que dispone el modelo, al tener en cuenta un mayor número de notas precedentes. En las melodías generadas que se adjuntan con este informe se han incluído tres relacionadas con este experimento (*ModeloOrden1.mp3*, *ModeloOrden3.mp3* y *ModeloOrden6.mp3*) en las que se puede apreciar el impacto del orden del modelo sobre la melodía: la generada con el modelo de sexto orden es más compleja que aquellas de ordenes inferiores, siendo aquella de primer orden la que suena más "aleatoria". El modelo de tercer orden parece más equilibrado, y además ofrece un número razonable de transiciones controlables (28 %).

4.2. Entrenando por Estilos

En este segundo experimento se busca identificar el impacto de los datos de entrada sobre la melodía generada. Para ello se ha ajustado la generación de la melodía de la misma forma que para el experimento previo: evitando cualquier tipo de

modificación que dependa de la posición del dispositivo. Dado que este impacto es evidente (la generación de la melodía depende directamente de los datos de entrada), el objetivo de este experimento es ver hasta qué punto se pueden diferenciar diferentes estilos mediante esta metodología. Para ello se ha entrenado un modelo de tercer orden (un valor equilibrado según el experimento anterior) con melodías de tres compositores clásicos: *Johann Sebastian Bach*, *Frédéric Chopin* y el español *Isaac Albéniz*.

Los resultados generados se encuentran anexos en los ficheros *Bach.mp3*, *Chopin.mp3* y *Albeniz.mp3*. Pese a que se puede apreciar una cierta diferencia de estilo, el resultado generado sigue siendo bastante aleatorio. El ritmo parece que se captura mejor pero las variaciones de tono (notas) no están al nivel que se podría esperar.

4.3. Entrenando Por Tonalidad

Teniendo en cuenta los resultados del experimento previo, se ha decidido entrenar el modelo únicamente con las canciones de una determinada tonalidad de un determinado compositor. El objetivo de este experimento es comprobar si con los datos adecuados, se puede generar una melodía con mejores variaciones de tono. El compositor elegido ha sido de nuevo *Johann Sebastian Bach* para poder comparar el resultado con los anteriores en los que no se tenía en cuenta la tonalidad.

Tras clasificar las canciones por tonalidades en función de las informaciones obtenidas en [20], se ha entrenado un modelo de Markov de tercer orden con todas aquellas de las tonalidades Sol Mayor y Re Menor. Los resultados se pueden escuchar abriendo los ficheros anexos *Bach - Sol Mayor.mp3* y *Bach - Re Menor.mp3*. Esta vez, las melodías generadas son bastante más curiosas por lo que parece que el modelo ha conseguido aprender la tonalidad. Por desgracia, clasificar los datos de entrada en función de la tonalidad es bastante complicado ya que es una información generalmente ignorada y difícil de obtener (en ciertos casos incluso puede cambiar a lo largo de una misma melodía). Podría ser interesante investigar alguna forma de extraerla directamente de los ficheros MIDI y automatizar este proceso.

4.4. Parámetros de Control

Por último, se han realizado varios experimentos para estudiar la influencia de los diferentes parámetros relacionados con el control en la generación de la melodía. En este caso se ha entrenado un modelo de segundo orden (por tener un número considerable de transiciones en las que pueda afectar la referencia según lo expuesto en 4.1) con los datos de la tonalidad Sol Mayor utilizados previamente. Se ha procurado realizar un control similar en todos los experimentos para que sean fácilmente comparables.

El primer parámetro estudiado ha sido "k", el peso de la referencia, marcando una referencia similar sobre los mismos datos pero variando su valor de 0,3 a 0,8. La distancia máxima entre la referencia y la salida ha sido dejada en unos valores

intermedios de 6 para las notas y 2 para las duraciones. Los resultados pueden verse en los histogramas de las figuras 16 y 17 a continuación. Se puede observar que con un parámetro "k" de 0,3 la melodía generada es más variada, ya que oscila más alrededor de la nota de referencia que en el caso de 0,8, cuando sigue esta misma referencia de forma más rigurosa. También es el caso para las duraciones, que se muestran relativamente más estables con el valor superior.

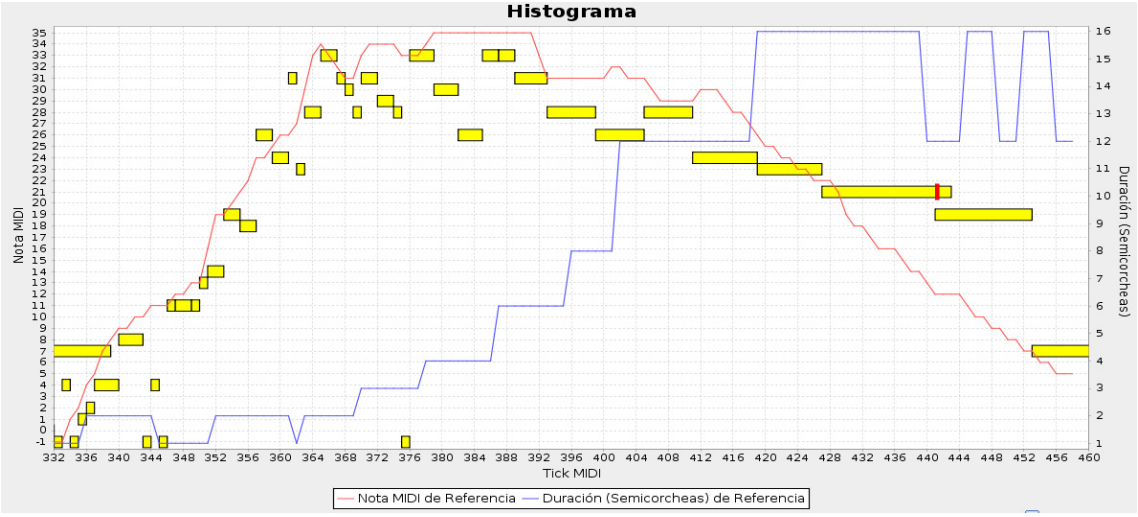

Figura 16: Melodía generada con k = 0.3

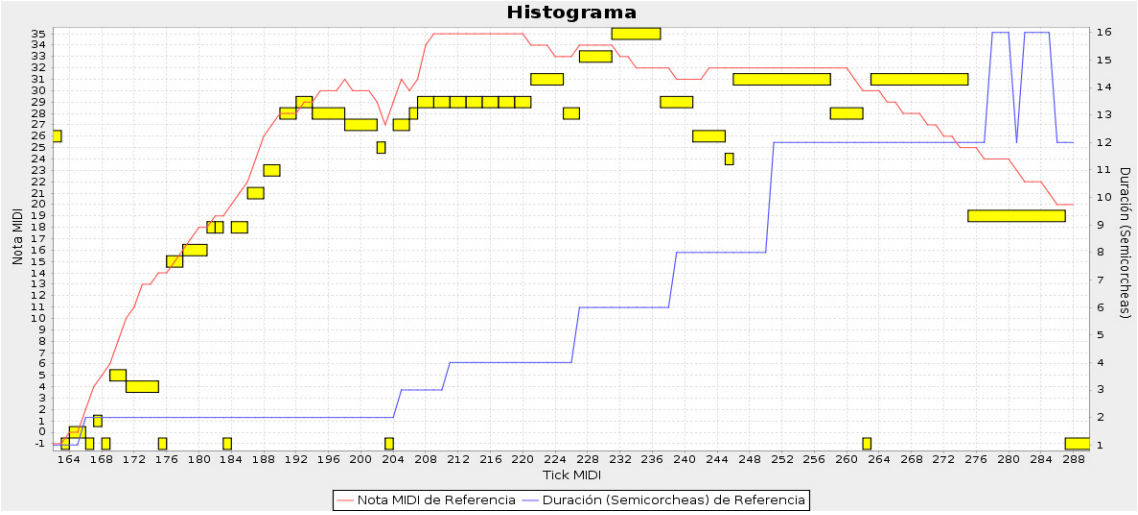

Figura 17: Melodía generada con k = 0.8

También se ha estudiado el impacto del rango de control (determinado por la distancia máxima entre la referencia y la salida). Para ello se ha aplicado la misma referencia que previamente sobre los mismos datos, pero esta vez con un parámetro "k" neutro ajustado a 0,5 y variando la distancia máxima respecto a la referencia permitida. Por un lado se ha experimentado con un rango estrecho, permitiendo únicamente aquellas transiciones alejadas como mucho de 3 notas y una duración

(dentro de las 8 duraciones normalizadas posibles) respecto a la referencia. Nótese que, tal y como se explica en 3.4.3, si no existe ninguna transición posible dentro del rango permitido se ignora la posición del dispositivo para evitar parar la generación de la melodía. Los resultados pueden verse en la figura 18.

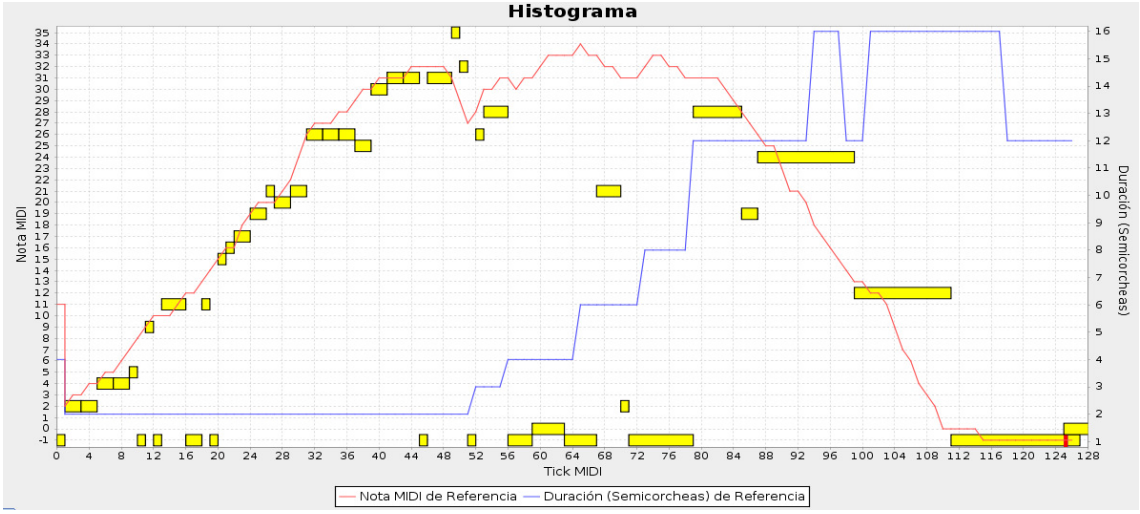

Figura 18: Melodía generada con un rango posible estrecho

Por otro lado, también se ha realizado el mismo experimento pero con un rango más amplio que permite notas hasta a una octava de distancia (12 notas) y a 3 duraciones de la referencia. La figura 19 muestra la melodía generado con estos ajustes. Al igual que el caso anterior, el rango estrecho obliga a la melodía a seguir más de cerca a la melodía mientras que el rango más amplio la permite una mayor libertad.

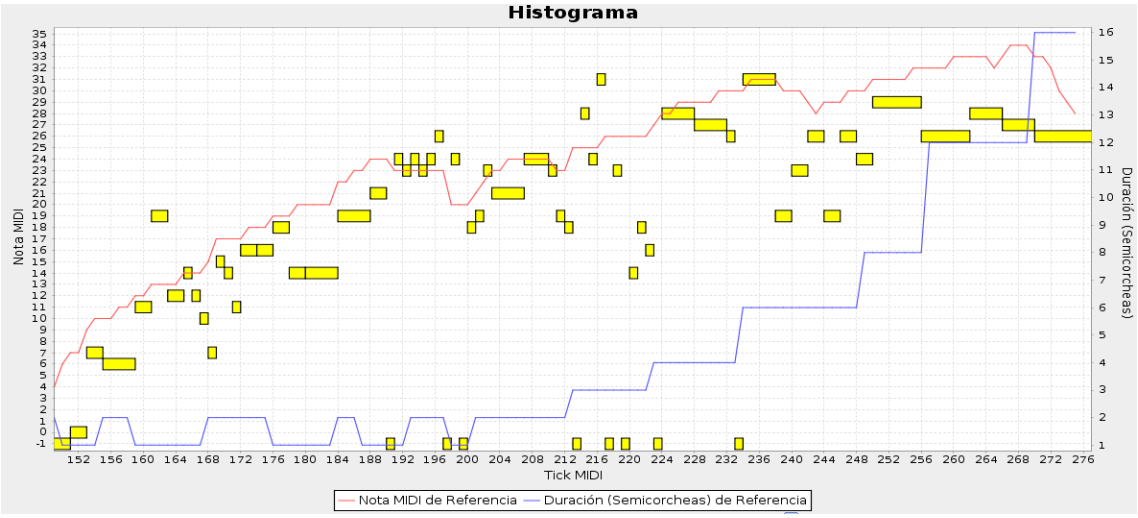

Figura 19: Melodía generada con un rango posible amplio

Realizando experimentos variados con los parámetros relacionados con el control y variando los datos, se ha podido apreciar que el parámetro “k” es de mayor utili-

dad cuando existen muchas transiciones posibles (permite mayor precisión), mientras que delimitar el rango funciona mejor en casos con un menor número de transiciones (permite alcanzar transiciones alejadas más facilmente). Pese a parecer dos parámetros que realizan la misma función a primera vista, utilizados correctamente son complementarios y permiten al usuario ajustar de manera relativamente intuitiva la manera de la que desea "dirigir" la generación de la melodía.

5. Conclusión

A lo largo de este informe se ha presentado todo el proceso de investigación realizado para estudiar las posibilidades de la utilización de un dispositivo de tipo *joystick* para la composición inteligente controlable de melodías, utilizando los ejes X e Y del dispositivo para el control de la duración y el tono de las notas generadas. El enfoque propuesto utiliza un conjunto de archivos MIDI, de los cuales se extraen las notas con sus respectivas duraciones reagrupadas por grupos de instrumentos. A continuación, se entrena una cadena de Markov con los datos del grupo deseado, y se van modificando las probabilidades de transición de este modelo en función del control para generar una melodía aleatoriamente respetando estas probabilidades "controladas".

Para realizar esta investigación se ha desarrollado una aplicación que podría ser descrita como un secuenciador inteligente controlable, ya que por una parte aprende a generar secuencias a partir de conjuntos de ejemplos y por otra admite la intervención directa del usuario para guiar el proceso de generación de la melodía. La acción del usuario sobre el sistema se ha implementado de dos formas: mediante el movimiento del ratón sobre una ventana interactiva o bien manejando un dispositivo mecánico tipo *joystick* capaz de recibir realimentación del propio modelo a través de sus motores.

Esta aplicación también permite configurar ciertos parámetros de control para modificar el peso relativo de la acción del usuario sobre la secuencia generada por el modelo o la resistencia que el dispositivo ofrece. Aunque el algoritmo de control propuesto se ha particularizado al dispositivo concreto que se presenta en este trabajo, sería fácilmente adaptable a otros tipos de dispositivos o actuadores.

Los resultados de los diferentes experimentos realizados remarcan la importancia de los datos de entrada sobre la melodía generada. Al entrenarse únicamente con dos dimensiones, las notas y sus duraciones, el modelo de Markov desconoce otras características importantes de las melodías extraídas como puedan ser la tonalidad o la armonización. Teniendo este factor en cuenta, se pueden seleccionar únicamente los datos de una tonalidad en particular, por ejemplo, para que la melodía generada respete esta tonalidad y los resultados sean mejores.

Sin embargo, obtener esta información es bastante complicado por lo que se podría buscar algún modo de intentar extraerla directamente del fichero MIDI. Si esto no fuese posible, se podría intentar extraer algún otro tipo de información adicional como por ejemplo la posición de cada nota dentro del compás. También

sería interesante explorar las posibilidades ofrecidas por un control relativo (quizás basado en lógica difusa) que no esté limitado a 3 octavas y que permita evitar tener que normalizar los datos de entrada.

Referencias

[1] Jose D Fernández and Francisco Vico. Ai methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48:513–582, 2013. [*Citado en págs. 1 y 2.*]

[2] Google. Deep dream. `https://github.com/google/deepdream`. [*Citado en pág. 1.*]

[3] Google. Magenta. `https://github.com/tensorflow/magenta`. [*Citado en pág. 1.*]

[4] Andreas Bergsland and Robert Wechsler. Composing interactive dance pieces for the motioncomposer, a device for persons with disabilities. In Edgar Berdahl and Jesse Allison, editors, *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 20–23, Baton Rouge, Louisiana, USA, May 31 – June 3 2015. Louisiana State University. [*Citado en pág. 1.*]

[5] Stanley Jungleib. *General Midi*. AR Editions, Inc., 1996. [*Citado en págs. 1, 4 y 5.*]

[6] Richard Boulanger et al. The csound book, 2000. [*Citado en pág. 2.*]

[7] Miller Puckette. Max at seventeen. *Computer Music Journal*, 26(4):31–43, 2002. [*Citado en pág. 2.*]

[8] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The reactable: exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 139–146. ACM, 2007. [*Citado en pág. 2.*]

[9] Levy Marcel Ingles Lorenzo. Intelligent midi sequencing with hamster control. `https://courses.cit.cornell.edu/eceprojectsland/STUDENTPROJ/2002to2003/lil2/hamsterMIDI_done_small.pdf`. [*Citado en págs. 2 y 7.*]

[10] François Pachet and Pierre Roy. Markov constraints: steerable generation of markov sequences. *Constraints*, 16(2):148–172, 2011. [*Citado en pág. 2.*]

[11] Martin Odersky. The scala language specification. [*Citado en pág. 4.*]

[12] Martin Thurau. Akka framework. *University of Lübeck, available online at http://media. itm. uni-luebeck. de/teaching/ws2012/sem-sse/martin-thurauakka. io. pdf [consulted March 29, 2014]*, 2012. [*Citado en pág. 4.*]

[13] Midi messages specifications. `https://www.midi.org/specifications/item/table-1-summary-of-midi-message`. [*Citado en pág. 6.*]

[14] How does the midi system work? `http://www.indiana.edu/~emusic/etext/MIDI/chapter3_MIDI4.shtml`. [*Citado en pág. 6.*]

[15] François Pachet and Pierre Roy. Markov constraints: steerable generation of markov sequences. *Constraints*, 16(2):148–172, 2010. [*Citado en pág. 7.*]

[16] Markov chains in scala. `http://blog.circuitsofimagination.com/2015/02/15/Markov-Chains.html`. [*Citado en pág. 9.*]

[17] Francisco Esquembre and J Sanchez. Easy java simulations. *Last accessed*, 5, 2012. [*Citado en pág. 12.*]

[18] Jacob Odersky. Flow: Serial communication for akka. `http://www.jodersky.ch/flow/`. [*Citado en pág. 14.*]

[19] General Midi: Instrumentos Disponibles. `https://en.wikipedia.org/wiki/General_MIDI#Program_change_events`. [*Citado en pág. 21.*]

[20] JS Bach. Obras de bach por tonalidades. `http://www.jsbach.es/`. [*Citado en pág. 23.*]

A. Apéndice A - Estrategia de *crawling* para la descarga de ficheros MIDI

Los mayoría de los sitios web que ofrecen grandes cantidades de ficheros MIDI de forma gratuita presentan por lo general estructuras simples pero variadas. Por ejemplo, en *http://mididatabase.com/* la estructura nos podría permitir determinar el estilo de cada MIDI, pero en *http://www.download-midi.com/* (que también parece estar estructurada de forma similar) los enlaces de descarga no están estructurados por estilos. Por ello se ha elegido una estrategia de *crawling* general que permita descargar MIDIs de cualquier página independientemente de su estructura: se recorren en busca de archivos MIDI todos los enlaces que se encuentran en una página objetivo, y a continuación todos los enlaces encontrados en esas páginas (excepto aquellos que ya se hayan recorrido), y así hasta alcanzar una profundidad determinada. Cada vez que se encuentra un fichero MIDI (indicado en el campo *"Content-Type"* de la cabecera de la respuesta HTTP), se descarga en el directorio especificado. También se ha añadido la posibilidad de seguir únicamente los enlaces que contengan una determinada palabra.

Tras ejecutar unas primeras pruebas se observó que el *crawler* era rechazado en un gran número de páginas, por lo que es necesario ajustar el campo *"User-Agent"* de las peticiones HTTP para simular un navegador tradicional. Tras precisar *Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)* en este campo, el *crawler* conseguía acceder a casi todas las páginas. Sin embargo, tras ejecutar otras pruebas se descubrió que en la página *http://www.download-midi.com/* cada vez que el *crawler* intentaba acceder a un enlace de descarga era redirigido a la página en la que se encotraba este enlace de descarga. Haciendo pruebas desde un navegador, se descubrió que la página permitía la descarga únicamente desde la página que contenía el enlace. Para solucionar esto, es necesario modificar otro campo de las peticiones HTTP, en este caso el campo *"Referer"*, de forma que se ajustase dinámicamente indicando siempre (excepto para la página inicial) la página en la que el *crawler* había encontrado el enlace al que quiera acceder.

Para poder configurar fácilmente el *crawler* se han incluído 4 campos en la interfaz gráfica. Los campos *Web Objetivo* y *Profundidad Máxima* son obligatorios, ya que son necesarios para la estrategia de *crawling*. El campo *Seguir enlaces con la palabra* es opcional (se puede dejar vacío), y por defecto contiene la palabra "midi" para evitar seguir enlaces que no tengan nada que ver con el objetivo de la aplicación. También se puede usar para hacer una descarga selectiva en ciertas páginas cuya estructura lo permita, como por ejemplo en *http://mididatabase.com/* (tanto las páginas como los archivos MIDI incluyen el estilo en el enlace). De este modo se pueden descargar únicamente los MIDIs del estilo "Dance", por ejemplo, si se siguen únicamente los enlaces que incluyan la palabra "dance". El cuarto campo indica dónde se van a descargar los MIDIs que se encuentren, y se inicializa por defecto a una carpeta nueva llamada *MidiWebMiner* en la carpeta por defecto del usuario.

B. Apéndice B - Código de la Extracción de las Notas

El método incluído a continuación recibe como parámetro un fichero MIDI, y extrae las secuencias de notas. El formato de la extracciones son el valor MIDI de las notas con sus respectivas duraciones en semicorcheas. Guarda los resultados en ficheros .txt organizados en carpetas en función del grupo de instrumentos al que corresponde cada uno.

```
def extractNotesFromMidi(midiFile: File): Boolean = {
    val sequence: Sequence = MidiSystem.getSequence(midiFile)
    val resolution = sequence.getResolution
    val divisionType = sequence.getDivisionType
    if (divisionType != Sequence.PPQ) {
      return false
    }
    val ticksPerSemiQuaver = resolution/4.0f
    val tracks = sequence.getTracks
    if (tracks.nonEmpty) {
      val notes = HashMap(
        "Piano" -> ListBuffer.empty[(Int, Int)],
        "Organ" -> ListBuffer.empty[(Int, Int)],
        "Guitar" -> ListBuffer.empty[(Int, Int)],
        "Bass" -> ListBuffer.empty[(Int, Int)],
        "Strings" -> ListBuffer.empty[(Int, Int)],
        "Reed" -> ListBuffer.empty[(Int, Int)],
        "Pipe" -> ListBuffer.empty[(Int, Int)],
        "Synth Lead" -> ListBuffer.empty[(Int, Int)]
      )
      for (track <- tracks) {
        var lastNoteEndTick: Long = -1
        var j = 0
        while (j < track.size) {
          val event = track.get(j)
          val msg = event.getMessage
          if (msg.isInstanceOf[ShortMessage]) {
            val sm = msg.asInstanceOf[ShortMessage]
            val midiCommand = sm.getCommand
            if (midiCommand == ShortMessage.PROGRAM_CHANGE) {
              val newMidiInstrument = sm.getData1
              newMidiInstrument match {
                case x if x < 8 => selectedInstrument = "Piano"
                case x if x > 15 && x < 24 => selectedInstrument = "Organ"
                case x if x > 23 && x < 32 => selectedInstrument = "Guitar"
```

```
        case x if x > 31 && x < 40 => selectedInstrument = "Bass"
        case x if x > 39 && x < 48 => selectedInstrument = "Strings"
        case x if x > 63 && x < 72 => selectedInstrument = "Reed"
        case x if x > 71 && x < 80 => selectedInstrument = "Pipe"
        case x if x > 79 && x < 88 => selectedInstrument = "Synth Lead"
        case _ => {
          selectedInstrument == "Other"
          println("notes for MIDI PC " + newMidiInstrument + " are not
            supported yet")
        }
      }
    } else if (midiCommand == ShortMessage.NOTE_ON && sm.getData2 != 0
      && selectedInstrument != "Other") {
      val initialTick = event.getTick
      var eventIndex = j + 1
      val currentNote = sm.getData1
      // add silence as -1 note
      if (lastNoteEndTick != -1 && lastNoteEndTick < initialTick) {
        val semiQuaversDuration =
          ((initialTick - lastNoteEndTick)/ticksPerSemiQuaver).toInt
        val markovStatus: (Int, Int) = (-1, semiQuaversDuration)
        notes(selectedInstrument) += markovStatus
      }
      while (eventIndex < track.size) {
        val event = track.get(eventIndex)
        val eventTick = event.getTick
        val msg = event.getMessage
        if (msg.isInstanceOf[ShortMessage]) {
          val sm = msg.asInstanceOf[ShortMessage]
          val midiCommand = sm.getCommand
          if (midiCommand == ShortMessage.NOTE_OFF
            && currentNote == sm.getData1) {
            if (divisionType == Sequence.PPQ) {
              val semiQuaversDuration =
                ((eventTick - initialTick)/ticksPerSemiQuaver).toInt
              val markovStatus: (Int, Int) =
                (currentNote, semiQuaversDuration)
              notes(selectedInstrument) += markovStatus
              lastNoteEndTick = eventTick
            }
            eventIndex = track.size()
          } else if (midiCommand == ShortMessage.NOTE_ON
            && currentNote == sm.getData1 && sm.getData2 == 0) {
            if (divisionType == Sequence.PPQ) {
              val semiQuaversDuration =
                ((eventTick - initialTick)/ticksPerSemiQuaver).toInt
```

```
                    val markovStatus: (Int, Int) =
                      (currentNote, semiQuaversDuration)
                    notes(selectedInstrument) += markovStatus
                    lastNoteEndTick = eventTick
                  }
                  eventIndex = track.size()
                }
              }
              eventIndex += 1
            }
            //notes(selectedInstrument) += sm.getData1
          }
        }
        j += 1
      }
    }
    for ((instrument, notes) <- notes) {
      if (notes.size > 8) {
        val outFile =
          new File(midiFile.getAbsoluteFile.getParentFile.getAbsolutePath +
          "/notes with duration/" + instrument + "/" + midiFile.getName + ".txt")
        textToFile(notes.mkString(" - "), outFile)
        notify("Se han extraido exitosamente " + notes.size + " notas de " +
          instrument + " del fichero " + midiFile.getName)
        extractedFiles += 1
      }
    }
    return true
  }
  return false
}
```

C. Apéndice C - Código de la Cadena de Markov

Las clases presentadas a continuación definen el modelo empleado: una cadena de Markov.

```
class MarkovChain[S](transitionMap: Map[ListBuffer[S], MarkovTransitionSet[S]]) {

  /**
    * Default constructors creates an empty Markov Chain
    */

  def this() = this(Map[ListBuffer[S], MarkovTransitionSet[S]]())

  // Create a new MarkovChain.
  def addTransition(prevState: ListBuffer[S], nextState: S) = {
    val transitions =
      if (transitionMap.contains(prevState)) transitionMap(prevState)
      else new MarkovTransitionSet[S]()

    val newTransitions = transitions.addTransition(nextState)
    new MarkovChain(transitionMap.updated(prevState, newTransitions))
  }

  def transitionProbability(prevStates: ListBuffer[S], nextState: S) = {
    transitionMap.get(prevStates) match {
      case Some(transitionSet) => transitionSet.probabilityFor(nextState)
      case None => 0.toDouble
    }
  }

  def transitionsFor(state: ListBuffer[S]) = {
    transitionMap.get(state) match {
      case Some(transitionSet) => transitionSet.toList
      case None => List[(S, Double)]()
    }
  }

  def states() = {
    transitionMap.keys
  }

  def controllableTransitionsCount(): Int = {
    transitionMap.filter((t) => t._2.controllable).size
  }

}
```

```
class MarkovTransitionSet[S](transitionCounter: Map[S, Int]) {

  def this() = this(Map[S, Int]())

  def apply(state: S) = probabilityFor(state)

  // Create a new MarkovTransitionSet.
  def addTransition(state: S) = {
    val count = countFor(state)
    new MarkovTransitionSet[S](
      transitionCounter.updated(state, count+1)
    );
  }

  def countFor(state: S) = {
    if(transitionCounter.contains(state))
      transitionCounter(state);
    else
      0
  }

  def totalCount(): Double = {
    val counts = transitionCounter.values
    val i = counts.foldLeft(0)((a, b) => a + b)
    i
  }

  def probabilityFor(state: S) = {
    countFor(state).toDouble / this.totalCount
  }

  def toList() = {
    transitionCounter.toList.map(tup => (tup._1, tup._2.toDouble / totalCount))
  }

  def controllable: Boolean = transitionCounter.size > 1

}
```

El método presentado a continuación se encarga de crear la cadena de Markov con secuencias de notas con su respectiva duración.

```
def extractMarkovChain(path: String) = {
    notify("Calculando Cadena de Markov de orden " + order +
      " con las notas y duracion de los archivos txt en la carpeta: " + path)
```

```
markovChain = new MarkovChain[(Int, Int)]()
val pathFile = new File(path)
val notesWithDuration = ArrayBuffer.empty[(Int, Int)]
var count = 0

for(file <- pathFile.listFiles if file.getName endsWith ".txt"){
  try {
    val fileNotes = extractNotesWithDurationFromTxt(file)
    notify("Extraidas " + fileNotes.size + " notas de " + file.getName)
    notesWithDuration ++= fileNotes
    count += 1
  } catch {
    case e: Exception => notify("Excepcion extrayendo notas del archivo " +
      file.getName + " en " + path + " : " + e);
  }
}

val notesWithDurationList = notesWithDuration.toList
//finding best normalization
val bestNormalization = findBestNormalization(notesWithDurationList)
kMMGUI.conductor ! UpdateOutputNormalization(bestNormalization)
kMMGUI.settings ! UpdateOutputNormalization(bestNormalization)

val normalizedNotesWithDurations =
    normalizeNotesAndDurations(notesWithDurationList, bestNormalization)

val prevStatus = new FixedList[(Int, Int)](order)
var transitionsCount = 0
normalizedNotesWithDurations.foreach { case (note: Int, duration: Int) =>
  if (!prevStatus.full) {
    prevStatus.append((note, duration))
  } else {
    val statusList = ListBuffer.empty[(Int, Int)]
    prevStatus.foreach(statusList += _)
    markovChain = markovChain.addTransition(statusList, (note, duration))
    transitionsCount += 1
    prevStatus.append((note, duration))
    //notify("prevStatus: " + prevStatus)
  }
}

// asegurarse de que siempre existen transiciones para cada estado
(0 to order - 1).foreach{ case i: Int =>
  val statusList = ListBuffer.empty[(Int, Int)]
  prevStatus.foreach(statusList += _)
  markovChain = markovChain.addTransition(statusList,
```

```
        normalizedNotesWithDurations(i))
      prevStatus.append(normalizedNotesWithDurations(i))
    }
    kMMGUI.conductor ! InitializeState(prevStatus)
    kMMGUI.conductor ! TransitionsList(markovChain.transitionsFor(prevStatus.list))
    val statesCount = markovChain.states.size
    val controllableStatesCount = markovChain.controllableTransitionsCount()
    notify("Modelo generado correctamente, numero total de estados: " +
        statesCount)
    notify("Estados con mas de una transicion posible: " +
        ((controllableStatesCount.toDouble/statesCount) * 100).round +
        " (" + controllableStatesCount + ")")
    notify("Â¡Notas extraÃdas exitosamente de los " + count +
        " ficheros encontrados en " + path + "! Se ha generado un modelo de Markov
        NumberFormat.getIntegerInstance().format(transitionsCount) +
        " transiciones")
}
```

D. Apéndice D - Algoritmo de Control

Los algoritmos presentados a continuación se encargan de ajustar las transiciones de la cadena de Markov en función de los valores de referencia según la lógica explicada en la subsección 3.4.3.

```
def calcControlProbabilities(markovProbabilites: List[((Int, Int), Double)],
    controlNote: Int,
    controlDuration: Int) = {
        var probs = scala.collection.mutable.Map[(Int, Int), Double]()
        markovProbabilites.foreach{
          case((note, duration), prob) =>
            val noteDistance: Int = math.abs(controlNote - note)
            val durationDistance: Int = math.abs(controlDuration - duration)
            if ((noteDistance <= maxNoteDistanceToControl || note == -1)
                && durationDistance <= maxDurationDistanceToControl)
                    probs += ((note, duration) -> calcNoteAndDurationProbability(
                        prob, note, noteDistance, duration, durationDistance))
        }
        if (probs.isEmpty) {
          markovProbabilites.foreach{
            case((note, duration), prob) =>
              probs += ((note, duration) -> prob)
          }
        }
        probs
    }

def calcNoteAndDurationProbability(markovProbability: Double,
    note: Int, noteDistance: Int,
    duration: Int, durationDistance: Int): Double = {
        var outProb: Double = (1.0 - k) * markovProbability
        if (noteDistance == 0 && durationDistance == 0) {
          val increase: Double = k * 0.4
          outProb += increase
        } else if ((noteDistance == 0 && durationDistance == 1) ||
            (durationDistance == 0 && noteDistance == 1)) {
              val increase: Double = k * 0.1
              outProb += increase
        } else if (noteDistance <= 1 && durationDistance <= 1) {
          val increase: Double = k * 0.05
          outProb += increase
        }
        outProb
    }
```