UNIVERSIDAD DE COSTA RICA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA
ESTRUCTURAS ABSTRACTAS DE DATOS Y ALGORITMOS PARA INGENIERÍA
IE-0217
I CICLO 2014
PROPUESTA DEL PROYECTO DE ESTRUCTURAS DE DATOS

Data Structures Implementations for K-Da Library

Estudiantes:
David Pérez Bolaños - B04769
Andrey Pérez Salazar - B25084
Andrés Sánchez López - B26214

1 Introducción

La creación de librerías en lenguajes de programación ayuda a generar una interfaz bien definida para una cierta funcionalidad en específico, es decir, dan al usuario ciertas ventajas a la hora de programar de una manera sencilla. Estas también sirven para separar por módulos un programa y de esta manera generar un código más claro y ordenado.

Una librería es un conjunto de funciones utilizadas para desarrollar software, no son programas, pero por lo general si son utilizadas por los programas para poder funcionar de forma correcta; el desarrollo de librerías sirve como apoyo a los programadores para tener más facilidades en la implementación de código en sus programas y a contar con más recursos para realizar sus proyectos. Existen muchas librerías para los diferentes lenguajes informáticos que hay.

En este caso, se implementó una librería cuya función es comparar los movimientos entre dos personas. Nuestra librería esta programada en el lenguaje C++. La idea de esta librería fue crear métodos o funciones en código, e implementar diferentes herramientas como el Kinect y el lenguaje informático Processing para lograr la función principal: comparar los movimientos de dos personas, esto, de manera virtual; esta técnica de comparación de movimientos humanos se utiliza mucho en videojuegos, efectos especiales, medicina, simuladores, entre otros.

Para continuar con el desarrollo del proyecto, se desarrollaron los siguientes objetivos: la implementación de algunas estructuras de datos, esto, en la manipulación de información obtenida mediante el Kinect de manera que se quiere realizar un análisis implementando algunas de ellas, para así determinar y comparar, qué estructuras son las más eficientes a la hora de manipular esta información.

En general, las estructuras de datos implementarán las funciones básicas de acceso o búsqueda de datos, lo cual nos ayudará a tener un manejo más rápido y ordenado de los datos obtenidos por el Kinect.

2 Desarrollo

2.1 K-Da Library

Como se mencionó con anterioridad, la creación de librerías de programación corresponden a una funcionalidad en específico. En nuestro caso, la librería implementada corresponde a un conjunto de métodos computacionales para llevar a cabo la comparación de movimientos. Para lograr esto, se implementó la librería en lenguaje C++, sin embargo, los datos de los movimientos son tomados de un controlador de juego libre y entretenimiento llamado Kinect; creado por Alex Kipman, desarrollado por Microsoft para la videoconsola Xbox 360 y OpenNI, que es el software utilizado para reproducir una interfaz en las computadoras, de las imágenes que recibe el Kinect de donde toma los datos. De este software se hablará detallamente más adelante. Además se usa también el lenguaje de programación Processing para poder obtener del Kinect los datos que se produzcan. Relacionadas estas cuatro importantes herramientas es que nuestra librería da el funcionamiento esperado.

2.2 Funcionamiento de K-Da Library

Retomando las herramientas que forman parte de la librería, iniciamos este apartado para describir, de una manera general, como se da la comparación de los movimientos.

2.2.1 Código en C++

El código de la librería fue desarrollado en el lenguaje C++. Este código presenta tres diferentes clases, una encargada de recibir los datos de movimiento que provienen del Kinect, sin embargo, es importante mencionar que, en nuestro caso, los movimientos captados son únicamente realizados por humanos, es decir, sí se presenta un movimiento de algún objeto no humano el Kinect no tomará los datos de tal movimiento, esto ya que está dentro del propio desarrollo del Kinect esta utilizar técnicas de reconocimiento de voz y reconocimiento facial para la identificación automática de los usuarios. Otra de las clases presentes en el código se encarga de convertir los datos (esto último se explicará con mucho más detalle más adelante) que fueron recibidos del Kinect y es aquí donde toma participación la última de las clases que es la encargada de realizar la comparación de los movimientos; cabe decir que para realizar una comparación es necesario contar con mínimo dos movimientos.

2.2.2 Kinect

Este dispositivo cuenta con una cámara RGB, un sensor de profundidad, un micrófono de múltiples matrices y un procesador personalizado que ejecuta el software patentado, que proporciona captura de movimiento de todo el cuerpo en 3D, reconocimiento facial y capacidades de reconocimiento de voz. Son las características anteriores lo que permite que esta última, pero no menos importante, herramienta es la que recolecta los datos del movimiento que se realicen. Utilizando algunos de sus componentes mencionados anteriormente, el Kinect es capaz de grabar los movimientos que realice un humano frente a él. Los datos que este produce corresponden a puntos de los *joints* en tres dimensiones, largo, ancho y profundidad, o más en general, posiciones en x, y y z de un plano que el mismo Kinect establece. Así bien, durante el movimiento ya sea en el *joint* de la muñeca, hombro o algún otro (según esté especificado en el código Processing), el Kinect generará un dato de tres dimensiones para tal *joint* en un tiempo específico, de manera que conforme el movimiento continúa se generan más datos 3D, con diferente posición, y obviamente, en un tiempo diferente.

2.2.3 OpenNI

Como bien se dijo antes, el Kinect es el encargado de recolectar los datos de las posiciones durante el movimiento que se presente. Sin embargo, el encargado de generar una interfaz para poder presenciar las imágenes que el Kinect recibe por cada uno de los movimientos es el OpenNI. OpenNI es un framework open source que provee APIs para el desarrollo de aplicaciones que utilicen interacción natural con el humano; de tal forma que facilita la comunicación de sensores de audio, video y profundiad del Kinect con una persona. Una de las empresas detrás de OpenNI es la empresa PrimeSense, que fue la creadora del Kinect del Xbox 360 de Microsoft, en un principio este driver y software no era abierto, pero para el 2010 PrimeSense lo liberó, generando así un gran aprevechamiento en áreas como la robótica entre muchas otras aplicaciones. En Abril del 2014 la empresa Apple adquirió PrimeSense por lo que el driver tanto como la información y documentación fueron cerradas en la web, esto obligó a muchas organizaciones a preservar la documentación y los binarios de la última versión a la que se tuvo acceso para así poderla utilizar en algún futuro.

2.2.4 Processing

Processing es un lenguaje de programación y entorno de desarrollo, creado por el MIT Media Lab, fue diseñado para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Simple-OpenNI es una librería de Processing que funciona como wrapper, de manera que podemos utilizar OpenNI de una manera más fácil. Por los motivos anteriores y por facilidad es que utilizamos processing como herramienta para tomar los datos que el Kinect genere según el movimiento y convertir estos datos en archivos *.txt* para luego estos, ser utilizados dentro del código en C++. En el código de Processing se

pueden elegir los movimientos de los que se quieren obtener los datos. Es decir, si necesitamos solo los datos de movimiento que genere el Kinect solo en el *joint* de la muñeca y no de todos los *joints* del diagrama del cuerpo humano en general se puede realizar el cambio dentro del código. En nuestro caso, Processing reproduce los datos del movimiento de los *joints* del torso, cuello, hombro y muñeca.

2.3 Clases

En este apartado se pretende dar una explicación más detallada de las funciones dentro del código, y de esta manera apreciar mejor el desarrollo del mismo, así, notar como se da la comparación a partir de los datos obtenidos del Kinect.

2.3.1 Class archivos

Una de las primeras clases que la librería posee es la clase *archivos*. Esta clase es la encargada de recibir los datos de los archivos *.txt*, y preparar estos datos para las siguientes funciones. Ahora bien, se sabe que los archivos *.txt* generados por Processing corresponden a una cantidad *n* de datos de posición (*x*, *y*, *z*), es por esto que, para el desarrollo de nuestra librería es necesario conocer la cantidad de datos que se obtuvieron por cada movimiento a comparar porque posteriormente será utilizado en otras funciones; de modo que en la clase *archivos* se tiene un método *getCantLineas();* cuya función es leer y guardar en una variable la cantidad de datos o líneas (ya que cada línea del archivo *.txt* corresponde a un dato) que se encuentren por cada movimiento. Una vez leídos la cantidad de datos del archivo *.txt*, se corre la función *guardarEnArreglo();*. Esta función es la encargada de tomar los datos del archivo generado por Processing y guardar estos datos en arreglo que forman parte de la librería para más adelante poder operar sobre ellos. Es por lo anterior, que se denota esta clase como la encargada de preparar los datos.

2.3.2 Class conversion

La siguiente clase que opera en nuestra librería corresponde a la clase *conversion*. Esta, como bien dice su nombre, se encarga de convertir los datos que provienen de la clase *archivos*, es decir, esta clase se desarrolla a lo largo de los arreglos que poseen los datos, que han sido implementados en la primer clase. Para conseguir una librería funcional, es necesario, en cierta manera, generalizar el proceso de la misma. Para entender mejor esta idea, detengámonos a pensar en las siguientes situaciones: se pretende comparar el movimiento que realiza una persona de gran tamaño contra el movimiento de otra persona de mucho menor tamaño, así bien, si nos regimos por la comparación de las posiciones de cada uno de las personas en tiempos relativamente parecidos, sucedería que por más acertado que sean los movimientos, la librería respondería que se han dado movimientos diferentes, esto, porque, a pesar de que los movimientos sean igualmente bien ejecutados hay una diferencia de posiciones en (*x*, *y*, *z*) entre los *joints* de las personas debido a su diferencia de tamaños. Otras situaciones como la anterior pueden suceder con otras cualidades similares, el tamaño de sus brazos, pies, torso, contextura del cuerpo en general. Esto por lo anterior, que se menciona que la librería se implementó de una manera más generalizada y es bajo la utilización de ángulos como datos. Debido a esta lógica, el desarrollo de la librería puede funcionar para la comparación de movimientos de personas sin importar sus diferencias físicas ya que los ángulos que se generan trás cada *frame* del movimiento no dependen cualidades físicas.

Es la función *convertir();* dentro de la clase *conversion* la que se encarga de convertir los datos de posición (*x*, *y*, *z*) en ángulos. Usando la librería *Eigen* y álgebra lineal, realizamos operaciones dentro de este método para conocer el ángulo que se va generando. Conociendo las posiciones de los *joints* de la muñeca/hombro y torso/cuello, se puede encontrar su vector, como su magnitud entros esos *joints* de modo que, conociendo la fórmula del producto punto, es posible despejar el

ángulo entre ambos vectores. De lo anterior, es que se encarga la función *convertir();*. Otra de las funciones de esta clase corresponde al método *llenarArregloAngulos();* que es la encargada de, una vez generados los ángulos, meterlos en un arreglo que luego será operado en otras funciones.

2.3.3 Class compara

La clase compara es la encargada de conseguir la respuesta a la comparación de los movimientos. En esta clase, ocurren una serie de funciones para reconocer que tan acertado estuvieron los movimientos de la segunda persona imitando a la primera. Además de conseguir la respuesta, también califica el movimiento en varias categorías según se verá más adelante.

Uno de los métodos en esta clase corresponde a la función *sacapromedios();*. Esta función es la encargada de recibir el arreglo de ángulos que se genera en la clase *conversion*. Una vez recibo, la función toma cada 10% del 100% de los datos y los coloca en un nuevo arreglo que correspondería a un arreglo de promedios. En otras palabras, se opera sobre la lista de datos de modo que se consigue un nuevo arreglo promedio de 10 posiciones en la que en la primer posición se tenga el primer 10% del total de los datos, en la segunda posición, el segundo 10% del total y así consecutivamente, hasta tener en la última posición el décimo 10% del total de los datos. Para llevar a cabo esta función se utilizan dos *for* para recorrer los datos de ángulos y para llenar el arreglo de promedios. Ahora bien, recordemos que es necesario dos movimientos para poder realizar una comparación, por tal razón, son dos los datos de ángulos que entran en esta función, lo que implica que son dos los arreglos con ángulos promedios que se obtendrían. Así, toma participación la siguiente función, *arreglo_promedio();*. Esta método recibe ambos arreglos de promedio; se utilizan diferentes *for* para recorrar y comparar cada una de las posiciones de ambos arreglos, la primer posición del arreglo generado con los ángulos que se presentaron a lo largo del movimiento de la primer persona contra la segunda persona, de modo que se comparan ambas diez posiciones. Dentro de la comparación se da un grado de variación de 5 grados entre los ángulos promedios ahí representados, y según se encuentre dentro o fuera del rango el ángulo de la persona uno con respecto a la persona dos, se generará un nuevo arreglo de otras 10 posiciones lleno de unos y ceros. De modo que, si hay un uno corresponde a que la variación del ángulo de uno con el otro es aceptable, caso contrario, un cero, indicando que hay un gran diferencias de ángulos en ambas posiciones por lo que se podría decir que en este momento, el movimiento no fue muy acertado. Al final, habrá entonces un nuevo arreglo que indica posición de movimiento acertado en ese 10% del total.

Seguidamente los métodos *comparar_angulos();* y *comparar_velocidad();* son los que finalmente evalúan que tan bueno estuvo el movimiento y se lo indica al usuario, esto lo hace utilizando como base los datos suministrados por las clases y métodos anteriores. Primeramente el método *comparar_angulos();* lo que hace es tomar el arreglo de unos y ceros proveniente de *arreglo_promedio();* y sumar todos los datos del arreglo, al ser este un arreglo de puros unos y ceros y de longitud 10, el valor de esta suma debería ser máximo 10 y mínimo cero, por lo tanto se puede hacer una evaluación de que tan bien estuvo el movimiento en base al valor de esta suma, si por ejemplo la suma dió uno, esto quiere decir que solo una decima parte del movimiento lo hizo casi igual o muy parecido al movimiento de la primera persona o movimiento “ideal” por lo tanto el restante 90% del movimiento lo hizo mal en comparación al movimiento “ideal”, lo que haría que en general el movimiento haya sido muy malo (para este ejemplo). Si por otro lado la sumatoria dió un valor de 5 esto quiere decir que la mitad del movimiento estuvo bien hecho y la otra mitad no, y así sucesivamente hasta llegar a un valor de la sumatoria de 10, lo cuál significaría evidentemente que el movimiento fue excelente. Seguidamente el método *comparar_velocidad();* lo que hace es determinar que tan rápido el usuario hace el movimiento en comparación con el movimiento “ideal”, esto ya que de nada serviría tener un 10 en la función *comparar_angulos();* si se hizo el movimiento muy lento, ya que efectivamente el movimiento sería igual pero no sería lo correcto decirle al usuario que estuvo bien su movimiento si dura mucho tiempo en ejecutarlo,

este método toma la longitud del arreglo de ángulos de los 2 movimientos y los compara, si hay una diferencia de más de un 10% se lo indica al usuario, diciendole que fue muy lento si se tuvo un 10% más o que fue muy rápido si se tuvo un 10% menos.

2.4 Estructuras de datos y complejidad

Como se menciona en gran parte del documento, la librería está basada en arreglos en donde almacenamos y generamos información, por ejemplo la información leída por parte del kinect de cada una de las posiciones de los joints es almacenada en arreglos, al igual generamos información con arreglos que nos dicen que tan bueno fue un movimiento, pues bien es importante analizar que podemos hacer más eficiente dicho almacenamiento y generación de datos utilizando algunas estructuras de datos, las estructuras nos aportaran elegancia, y una forma más eficiente para trabajar con datos, por ejemplo en un nodo de las estructuras, podemos englobar toda la información de una posición, creando variables que nos permite tener un acceso mejor, por ejemplo a las posiciones x,y,z, así como a la hora de generar los ángulos necesarios.
Para este proyecto se utilizaran 3 estructuras de datos las cuales se ejecutaran y luego serán analizadas para así determinar cual es la que más nos sirve, o cual da una respuesta más eficiente y eficaz a nuestro problema. La complejidad de las estructuras de datos fueron un aspecto a considerar muy importante en la realización del proyecto, ya que mediante la complejidad es posible determinar cuáles estructuras son más eficientes que otras y por lo tanto esto permite que se pueda escoger cuál es la estructura que finalmente creará el algoritmo más eficiente posible. Para el proyecto se escogieron 3 estructuras de datos: La lista doblemente enlazada, Cola y Pila. Se escogieron estas ya que son las más utilizadas cuando se trabaja con muchos datos que se van creando y almacenando, como es el caso de los datos provenientes del kinect. Primeramente con la pila se tenía una complejidad de almacenamiento de O(n), el problema con esta estructura de datos es que al ser una estructura de tipo LIFO, primero se debía desapilar todos las datos para luego recorrer toda la pila ya que esta sería la única manera de recorrer todos los datos provenientes del kinect de manera ordenada (osea en el orden que provienen del kinect). En el caso de la lista se tiene que al insertar cada dato nuevo en la lista el algoritmo tiene una comlplejidad de O(1), pero en los métodos que se implementa la lista como un todo (cuando ya ha sido almacenada en memoria) el algoritmo se convierte en un O(n), la lista permite recorrer todos los datos en orden ya que los punteros idican el orden en que se deben recorrer los datos. Algo parecido sucede con la cola, ya que cada iteración al ir guardando los datos provenientes del kinect en memoria es un O(1) pero cuando se recorre la cola en cualquiera de los métodos utilizados se convierte en un O(n), pero al ser la cola una estructura de tipo FIFO, esto permitiría asegurarse de una manera mas eficiente (sin punteros como en la lista) que los datos provenientes del kinect se van a recorrer de una manera ordenada.

2.5 Complejidad del algoritmo de comparación

Para determinar la complejidad del algoritmo de comparación total, es decir, toda la librería, es necesario dar un análisis de todas y cada una de las funciones que ocurren en este. Al ser tantas, hemos decidido dar algunos ánalisis en ciertas partes importantes del código y suficientes para encontrar el orden de complejidad del mismo. Como primer punto, tenemos la clase *archivos* cuya función general corresponde a introducir dentro del código los datos emulados por el Kinect. Esto sucede de la siguiente manera: primero se crea el arreglo $A[n]$, sumando de 1, luego inicia un *for* en $i = 0$ hasta *n*, el número de datos, $i++$, sumando de 1 + *n*. Finalmente dentro del *for* sucede que cada línea i es igual a la posición $A[i]$, sumando de 3n. Lo que da un total de 4n + 2, que indica que su orden de complejidad en este algoritmo es de O(n).

Otro de las secciones por analizar es la encargada de generar el arreglo de unos y ceros. En esta método sucede lo siguiente: primero se crea el arreglo de 10 posiciones, 1. Luego inicia un *for* para esas 10 posiciones, 1 + 10. Dentro del *for*

se inicia una variable y otro nuevo *for* de otras 10 posiciones, pero como ya están dentro de un *for* el sumado es 10 + 10 + 100. En este segundo *for* se produce la suma de los promedios, 100 + 100. Finalizado el segundo *for* se almacena el dato obtenido en el arreglo de los promedios del primer *for* en sus diez diferentes posiciones, 10 + 10. Y finalizado este primer *for*, se retorna el arreglo, 1. Esto, para dar con una constante 353. Como se aprecia a lo largo del código, y posterior al método en la clase *archivos*, se presentan órdenes de complejidad constantes, por lo que se puede mencionar que, en general, el algoritmo de comparación corresponde a un orden de complejidad de O(n).

3 Referencias

1. Richard, J. Computer Science Division. University of California at Berkeley. Triangle. A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator. Encontrado el 13 de abril del 2014 en: http://www.cs.cmu.edu/ quake/triangle.html
2. Escenografía Intermedial. Nuevos medios y tecnologías afines a la escena. (15 de mayo del 2012). Nube de puntos (Point Cloud) con Kinect. Encontrado el 13 de abril del 2014 en: http://escenografiaaumentada.wordpress.com/2012/05/15/148/
3. OPENKINECT. Encontrado el 13 de abril del 2014 en: http://openkinect.org/wiki/Main_Page
4. Joyanes, L., Sánchez, L. & Zahonero, I. (2007). Estructuras de datos en C++ (1ra ed.) Madrid: McGraw-Hill / Interamericana de España, S.A.
5. Ramis fuambuena, A. (2012). Cámaras de reconocimiento Recuperado el 08 de Julio del 2014, de http://personales.alumno.upv.es/alraf
6. Abrego, M. (2011, 31 de Diciembre). Project Natal (KINECT) PASADO-PRESENTE-FUTURO | MalenyMSP Recuperado el 08 de Julio del 2014, de http://malenyabrego.wordpress.com/2011/12/31/mundo-kinect/