UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Evaluación Automática e Interactiva de Destrezas Mediante Distancias entre Cadenas

Proyecto Final de Carrera

Ingeniería Informática

Autor: Alejandro Lucas Villaverde

Director: Carlos Monserrat Aranda

Co-Director: José Hernández-Orallo

5 de Junio del 2012

Resumen

Desarrollo de un método para la evaluación automática e interactiva de destrezas durante la realización de ejercicios quirúrgicos mínimamente invasivos usando técnicas de distancia (similitud) entre cadenas de movimientos, y estudio de su capacidad para discriminar entre sujetos de distintos niveles de experiencia.

Palabras clave: evaluación, destreza, distancia, similitud, cadenas, interactivo

Índice

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Objetivos del Proyecto

El objetivo del proyecto en el que se enmarca este PFC es explorar las posibilidades de un sistema de evaluación automática (sin intervención de un experto) e interactiva (durante la realización de la prueba) de la destreza de los estudiantes de cirugía en técnicas mínimamente invasivas.

Este PFC sirve como primera aproximación a dicho objetivo, desarrollando y validando la parte más importante: el evaluador de destreza interactivo. Como resultado del estudio previo (sección **1.2**) se ha podido ver la ausencia de sistemas de evaluación interactiva en el mercado, ya que todos ellos proporcionan la evaluación al terminar, y la imposibilidad de adaptar estos a un uso interactivo.

Debido a esto, como método para la evaluación se va a usar, de forma novedosa, las medidas de distancia entre cadenas. Este tipo de algoritmos (sección **1.3**) nunca han sido utilizados en la evaluación de destrezas, y menos aun desarrollados para trabajar con información que aun está siendo generada (en lugar de con la información completa de antemano).

Por lo tanto, el objetivo de este PFC es desarrollar y validar un método de evaluación automática e interactiva de destrezas mediante el uso de distancias entre cadenas. Pero para poder cumplir este objetivo habrá que desarrollar también un simulador con el que poder obtener las cadenas de movimientos necesarias para probar y validar dicho método.

1.2. Estado del Arte: Técnicas de Evaluación Automática de Destrezas

La evaluación de destrezas en el aprendizaje de técnicas quirúrgicas, como las mínimamente invasivas (*MIS*), no es ninguna novedad. Debido a la dificultad de este tipo de técnicas su curva de aprendizaje es más larga. Factores como la dificultada de manejo del instrumental, un *feedback* táctil limitado, y trabajar en un área reducida hacen que la adquisición de destrezas en esta técnica sea muy importante.

Debido a esto, en los últimos años se han desarrollando sistemas de entrenamiento en técnicas *MIS* con los que los estudiantes pueden aprender. La parte principal de estos sistemas es el evaluador de destreza, que proporciona una medida objetiva de la experiencia demostrada por el estudiante durante la prueba. A continuación se muestran algunos de estos métodos.

1.2.1. Métodos Basados en Pesos

1.2.1.1. Quantitative Assessment of the Surgical Training Methods with the Suture/Ligature Training System WKS-2RII

[**28**]

Se evalúa la destreza en la realización de suturas. El sistema *WKS-2RII* proporciona los siguientes parámetros tras cada una de las suturas: tiempo requerido para completar la tarea, fuerza ejercida sobre la piel, tensión ejercida en la piel, distancia entre el punto de sutura y el límite de la herida, distancia entre suturas, y apertura resultante en la herida.

Para evaluar la destreza se usa una función que, como resultado, devuelve una puntuación. Para determinar dicha función se usa un método de *Análisis Discriminante (DA)* [**Anexo**] cuyo cometido es clasificar las muestras entrantes en una de las clases (niveles de destreza). El resultado de este método es un conjunto de funciones lineales discriminantes. En concreto, en el caso del autor la función resultante es la siguiente:

$$E_{Sutura} = w_T\, I_T + w_{ShF}\, I_{ShF} + w_{OpF}\, I_{OpF} + w_{SuD}\, I_{SuD} + w_{SuW}\, I_{SuW} + w_{WoA}\, I_{WoA} \qquad \textbf{(1.1)}$$

En esta función, los coeficientes discriminantes (w) multiplican a las variables de entrada (I) por un factor, según su importancia, para obtener la puntuación final. Para determinar el valor de dichos coeficientes, el autor utiliza un conjunto de muestras de entrenamiento de individuos de tres niveles de destreza conocidos: cirujanos experimentados, estudiantes con algo de conocimiento, y gente sin conocimiento alguno. Para clasificar correctamente a estos individuos con la función discriminante anterior, se llego a unos valores de los coeficientes discriminantes determinados.

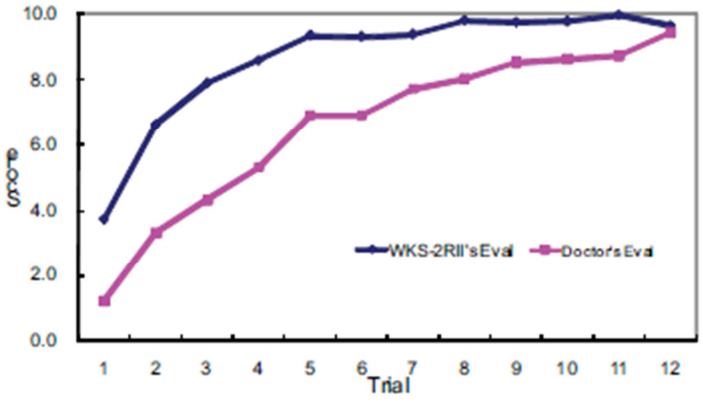

Figura 1.1: Comparativa entre la opinión de un doctor (en morado) sobre 12 suturas realizadas por un alumno, y la puntuación obtenida con el sistema *WKS-2RII* (en azul).

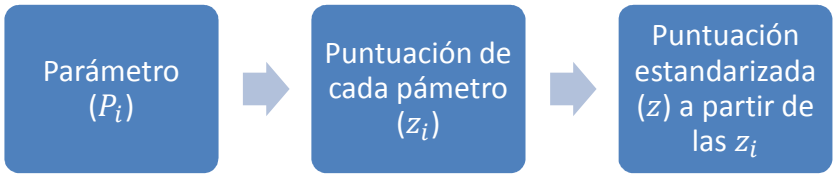

Figura 1.2: Proceso seguido por CELTS para obtener la puntuación que represente la destreza de un individuo.

Con la **Ecuación 1.1** y los pesos resultantes de la fase de entrenamiento se puede evaluar el nivel de destreza en suturas de un sujeto con los parámetros obtenidos con el sistema *WKS-2RII*. En la **Figura 1.1** se puede ver un ejemplo de evaluación.

1.2.1.2. CELTS: A Clinically-Based Computer Enhanced Laparoscopic Training System

[**31**]

En este caso se usan cinco indicadores de destreza: profundidad total recorrida por el instrumento, suavidad de movimiento, orientación del instrumental, longitud del camino seguido y tiempo requerido para terminar la prueba. Para cada estudiante se obtienen dichos indicadores al terminar la prueba y se realizan los siguientes pasos (**Figura 1.2**):

- Para cada indicador se calcula una puntuación (*z-score*) que se obtiene de la siguiente forma:

$$z_i = \frac{P_i^N - \overline{P_i^E}}{\sigma_i^E} \qquad \textbf{(1.2)}$$

P_i^N : medida del estudiante en el indicador *i*

$\overline{P_i^E}$: medida del experto en el indicador *i*

σ_i^E : desviación estándar de los expertos en el indicador *i*

  Como se puede observar en la ecuación anterior, se requiere de un entrenamiento previo por un conjunto de expertos.

- Para obtener la puntuación final se realiza un sumatorio de todas las *z-score* de los 5 indicadores de la siguiente forma:

$$z = 1 - \frac{\sum_{i=1}^{N} a_i z_i}{\sum_{i=1}^{N} a_i z_{max}} - a_0 z_0 \qquad \textbf{(1.3)}$$

a_i : peso asociado a z_i

z_0 : medida del resultado de la tarea

Los pesos se usan para variar la importancia de cada uno de los indicadores según se desee en cada prueba.

Cuando se desee evaluar a un estudiante con este método bastará con obtener las cinco métricas tras realizar una determinada tarea, obtener la puntuación de cada una de ellas (**Ecuación 1.2**) y, por último, la puntuación final (**Ecuación 1.3**).

1.2.2. Métodos Basados en LDA (Análisis Discriminante Lineal)

1.2.2.1. Objective Evaluation of Laparoscopic Surgical Skills Using Waseda Bioinstrumentation System WB-3

[**18**]

El sistema *WB-3* proporciona, tras cada prueba, un vector que contiene 54 datos del estudiante. Este vector $X_i(t)$ será procesado para poder clasificar al individuo según su destreza. El proceso consta de cuatro etapas, tal y como se puede ver en la **Figura 1.3**, que son:

- **Extracción de Características:** se realiza un estudio estadístico para averiguar los parámetros más importantes a extraer a partir de los datos de entrada. El resultado son los 7 parámetros: velocidad angular media de ambos hombros, función de distribución acumulada (CDF) de la velocidad angular de ambos hombros, frecuencia de movimiento de ambos hombros, y ratio de uso del hombro derecho.

- **Normalización de Características:** dado que cada uno de los parámetros característicos tiene unas unidades y rangos distintos, esto puede perjudicar posteriores etapas de clasificación. Por ello se usa la media y desviación de cada parámetro para normalizar las características, como se puede ver en la siguiente ecuación:

$$N_i(k) = \frac{1}{\sigma_i^2}\,(L_i(k) - \mu_i) \qquad \textbf{(1.4)}$$

$$\mu_i = \frac{1}{N}\,L_i(k)$$

$$\sigma_i^2 = \frac{1}{N}\,(L_i(k) - \mu_i)^2$$

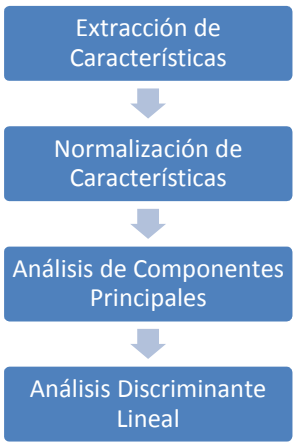

Figura 1.3: Proceso seguido por WB-3 para obtener el nivel de destreza en una laparoscopia.

- **Análisis de Componentes Principales (PCA):** [**Anexo**] las diferentes características extraídas pueden estar midiendo el mismo indicador de destreza. Para verificar la información útil que contiene cada parámetro, y de paso reducir el tamaño de las características, se usa *PCA*.

  Este método consiste en extraer un conjunto de componentes principales, que son combinaciones lineales de las características, y ordenarlos según cuanta varianza muestran (p.e.: el primer componente tendrá la mayor varianza presente en los datos).

  En este caso, con las dos primeras componentes principales se tiene la mayor parte de la varianza acumulada, por lo que sólo se necesitan estas dos componentes principales para la posterior clasificación.

- **Análisis Discriminante Lineal (LDA):** [**Anexo**] se usa para decidir a qué clase/grupo de destreza pertenece el sujeto según sus componentes principales. Este método necesita de un conjunto de datos de entrenamiento, compuesto por componentes principales de sujetos pertenecientes a dos niveles de destreza conocidos: expertos y novatos. A partir de estos, *LDA* aprende la distribución de componentes que clasifican los sujetos en ambos grupos.

Para obtener el grupo (nivel de destreza) al que pertenece un nuevo sujeto basta con obtener los parámetros, extraer las características, normalizarlas, obtener las componentes principales y, por último, *LDA* estimará al grupo al que más probablemente pertenezca el sujeto comparando sus componentes principales con las de las muestras de entrenamiento. En la **Figura 1.4** se puede ver un ejemplo de clasificación de varios sujetos.

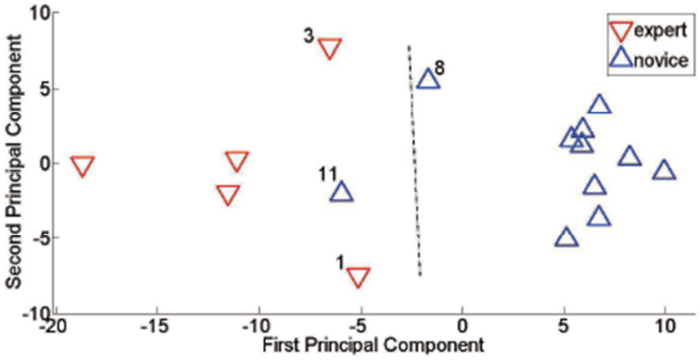

Figura 1.4: Ejemplo de clasificación errónea con WB-3, ya que el sujeto 11 (novato) es clasificado como experto por estar a la izquierda de la línea de decisión de LDA.

1.2.2.2. Objective Classification of Residents Based on their Psychomotor Laparoscopic Skills

[**5**]

El sistema de tracking *TrEndo* proporciona una serie de datos en forma de *MAPs* obtenidos de la prueba laparoscópica. En concreto se obtienen los siguientes 6 *MAPs*: tiempo requerido para realizar la tarea, longitud del camino seguido por la punta de instrumento, profundidad total recorrida por la punta del instrumento a través de su eje, suavidad de movimiento, área angular (distancia entre los puntos más lejanos en los que estuvo la punta del instrumento), y volumen (el área angular teniendo en cuenta también la profundidad).

Al igual que en el método anterior ([**18**]) se utiliza *PCA* para reducir el número de datos (se quedan con las dos primeras componentes) y *LDA* para clasificar en uno de los tres grupos de destreza (experto, intermedio y novato).

1.2.2.3. Towards Automatic Skill Evaluation: Detection and Segmentation of Robot-Assisted Surgical Motions

[**17**]

El sistema pretende detectar y segmentar los movimientos durante una intervención de laparoscopia para así evaluar el nivel de destreza. Como evaluador se va a crear un clasificador haciendo uso de los datos almacenados por el sistema *Da Vinci* (*Intuitive Surgical*).

Se obtienen 72 datos por unidad de tiempo: velocidades-posiciones cartesianas, matrices de rotación, posiciones y velocidades de las articulaciones. La tarea a evaluar es una determinada sutura, de la que un especialista extrae ocho movimientos comunes a realizar por cualquier estudiante. Estos 8 movimientos formarán el vocabulario de la tarea.

El proceso llevado a cabo es el siguiente (**Figura 1.5**):

1. **Extracción de Características Locales:** de un movimiento a otro, durante la laparoscopia, los datos cambian abruptamente. Esto indica que podemos utilizar los datos del movimiento actual junto con los de los predecesores y sucesores para crear un súper-vector de características (**Figura 1.6**), que se usará en etapas posteriores.

2. **Normalización de Características:** dado que datos como la velocidad y la posición tienen unidades y rangos diferentes, es necesario normalizar el súper-vector de características.

3. **Análisis Discriminante Lineal (LDA):** [**Anexo**] se usa este método con el fin de reducir las dimensiones del súper-vector de características.

4. **Clasificador de Bayes:** este clasificador dirá en qué paso de la laparoscopia es más probable que se encuentre el sujeto ($\hat{C}(n)$). Para ello se usan las probabilidades a posteriori de la siguiente forma:

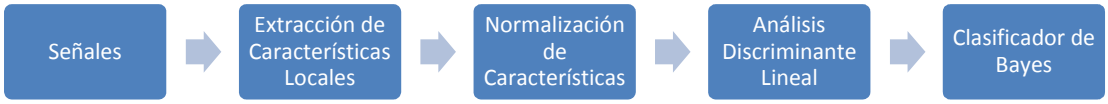

Figura 1.5: Etapas del proceso seguido para obtener en qué paso de una laparoscopia se encuentra el sujeto.

$$L(k_t) = [X(k_{t-m}) \mid X(k_{t-m+1}) \mid \ldots \mid X(k_t) \mid \ldots \mid X(k_{t+m-1}) \mid X(k_{t+m})]$$

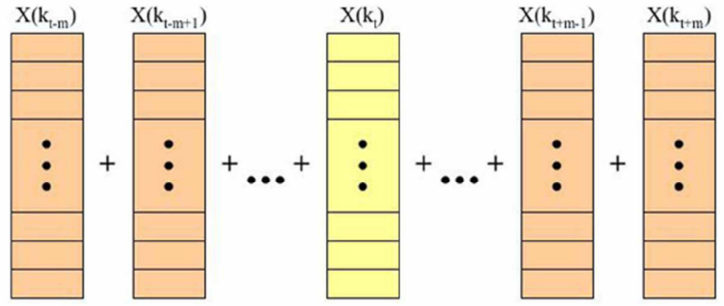

Figura 1.6: Creación del súper-vector de características.

$$\hat{C}(n) = argmax_{C(n)}\, P\big(C(n)\big|Y(k)\big) = \; argmax_{C(n)}\, P\big(Y(k)\big|C(n)\big)\, P\big(C(n)\big) \qquad \textbf{(1.5)}$$

Para crear el clasificador (obtener las probabilidades $C(n)$ de pertenencia a cada uno de los pasos de la laparoscopia) hace falta entrenarlo. Para esto se necesita un conjunto de datos pertenecientes a sujetos de destreza conocida (expertos e intermedios).

Para evaluar a un nuevo sujeto se obtienen todos los datos que ha generado durante su laparoscopia y se detectan los movimientos que ha realizado mediante el clasificador de Bayes. Si estos son los mismos que los dictaminados por los expertos para una correcta operación, se podrá decir que dicho sujeto la ha realizado correctamente.

1.2.3. Métodos Basados en Modelos de Markov

1.2.3.1. Generalized Approach for Modeling Minimally Invasive Surgery as a Stochastic Process Using a Discrete Markov Model

[**24**]

Parámetros de Evaluación

El sistema mide, durante el ejercicio de laparoscopia, los siguientes parámetros sobre el instrumental: posición, orientación, fuerza y torsión. Luego agrupa los posibles valores de fuerza y torsión (provenientes de 7 sensores) en tres niveles de magnitud: alto, medio y bajo.

A continuación, tras analizar ejercicios realizados por expertos, define 14 posibles acciones a realizar con el instrumental y traduce los datos medidos de posición y orientación a una de las 14 posibles acciones que aparecen en la **Figura 1.7**.

Primera Aproximación

En un primer método de evaluación ([**26**] y [**27**]) se representan las acciones a realizar durante una laparoscopia mediante un diagrama finito de estados (*FSD*) como el que se puede ver en la **Figura 1.8**.

Dado que en cada una de las anteriores acciones se ha de realizar una determinada fuerza y tensión, podemos definir una arquitectura *HMM* [**Anexo**] para modelar el paso entre los 3 niveles de magnitud Fuerza/Tensión: alto, medio y bajo (**Figura 1.9**).

Type	State Name	State Acronym	Force / Torque						
			Fx	Fy	Fz	Tx	Ty	Tz	Fg
I	Idle	ID	*	*	*	*	*	*	*
	Grasping	GR							+
	Spreading	SP							-
	Pushing	PS			-				
	Sweeping	SW	+/-	+/-		+/-	+/-		
II	Grasping - Pulling	GR-PL			+				+
	Grasping - Pushing	GR-PS			-				+
	Grasping - Sweeping	GR-SW	+/-	+/-		+/-	+/-		+
	Pushing - Spreading	PS-SP			-				-
	Pushing - Sweeping	PS-SW	+/-	+/-	-	+/-	+/-		
	Sweeping - Spreading	SW-SP	+/-	+/-		+/-	+/-		-
III	Grasping - Pulling - Sweeping	GR-PL-SW	+/-	+/-	+	+/-	+/-		+
	Grasping -Pushing - Sweeping	GR-PS-SW	+/-	+/-	-	+/-	+/-		+
	Pushing - Sweeping - Spreading	PS-SW-SP	+/-	+/-	-	+/-	+/-		-

Figura 1.7: Acciones que se pueden realizar con el instrumental de laparoscopia y la fuerza/torsión ejercida en cada uno de ellos.

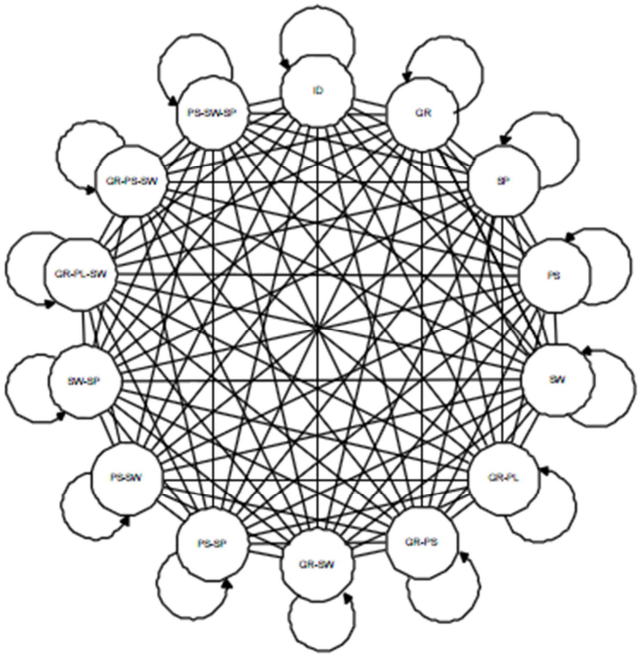

Figura 1.8: FSD que representa una laparoscopia, donde cada estado es una de las posibles acciones a realizar detalladas en la Figura 1.7.

Para cada estado del *FSD* arriba mostrado tendremos dos *HMM* como el anterior, uno que representa a los expertos ($\lambda_{experto}$) y otro que representa a los no-expertos ($\lambda_{no-experto}$). De esta forma, cuando un estudiante esté realizando la laparoscopia se mirará si está siguiendo las acciones definidas por los expertos (si sigue un determinado *FSD*) y si la fuerza/tensión que ejerce se corresponde a las de un experto o a las de un no-experto. En concreto, se usan las siguientes funciones de decisión dada una observación en un determinado estado del *FSD*:

$$\mathrm{E} = \frac{\log(P(observación \mid \lambda_{experto}))}{\log(P(observación \mid \lambda_{observación}))} \tag{1.6}$$

$$\mathrm{NE} = \frac{\log(P(observación \mid \lambda_{no-experto}))}{\log(P(observación \mid \lambda_{observación}))} \tag{1.7}$$

La función E (**Ecuación 1.6**) define cual es la similitud estadística entre la actuación del sujeto bajo estudio y el grupo de expertos y la función NE (**Ecuación 1.7**) y lo mismo para un grupo de no-expertos (**Figura 1.10**). Si se tratase de un sujeto completamente experto la función E devolvería un 1 y la función NE un 0.

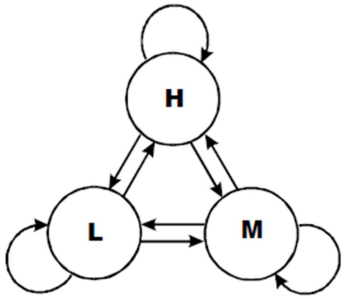

Figura 1.9: HMM que representa las 3 magnitudes Fuerza/Tensión posibles.

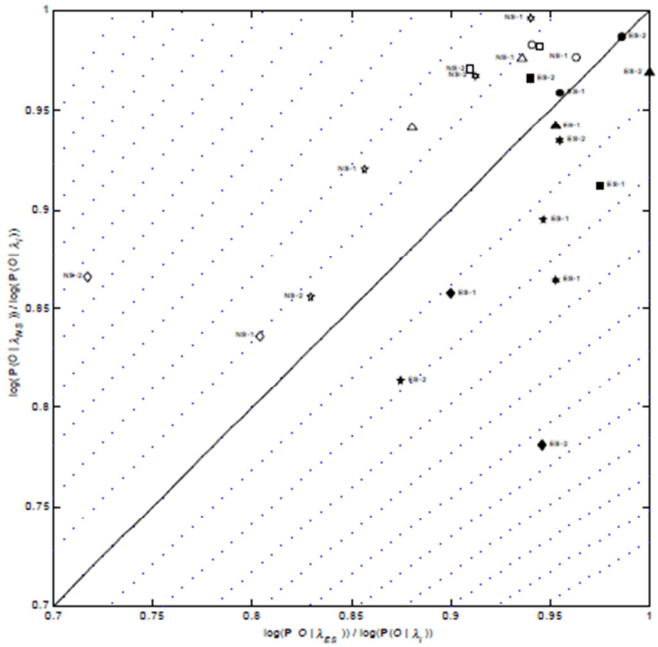

Figura 1.10: Los ejes X e Y representan valores obtenibles por las ecuaciones 1.6 y 1.7, tanto por expertos (ES) como por no-expertos (NS). La línea diagonal separa ambos grupos, por lo que hay 3 expertos que fueron evaluados como no-expertos.

Segunda Aproximación

Para saber cómo de lejos está un nuevo sujeto de hacerlo como los expertos, primero se ha de entrenar el sistema experto. En el artículo [**24**] se han usado 30 sujetos divididos en 6 grupos de destreza (5 en cada uno), cuya evaluación dependerá de la distancia estadística entre sus *HMM* y los de los expertos.

Para cada uno de estos 30 sujetos se obtiene un *HMM*. Para obtener esta distancia estadística lo primero que se ha de definir es cómo se calcula la distancia entre dos *HMM* (λ_1 *y* λ_2):

$$D(\lambda_1, \lambda_2) = \frac{1}{T_{O_2}}[logP(O_2|\lambda_1) - logP(O_2|\lambda_2)] \quad \textbf{(1.8)}$$

$$D(\lambda_2, \lambda_1) = \frac{1}{T_{O_1}}[logP(O_1|\lambda_1) - logP(O_1|\lambda_2)] \quad \textbf{(1.9)}$$

Esta distancia (**Ecuación 1.8**) es una medida de cómo de bien el modelo *HMM* λ_1 iguala las observaciones generadas por el modelo *HMM* λ_2 relativo a cómo de bien el modelo *HMM* λ_2 iguala las observaciones generadas por si mismo. Pero, como tanto (**Ecuación 1.8**) y (**Ecuación 1.9**) son no-simétricas, la versión simétrica es la siguiente:

$$D_S(\lambda_1, \lambda_2) = \frac{D(\lambda_1, \lambda_2) + D(\lambda_2, \lambda_1)}{2} \quad \textbf{(1.10)}$$

Se normaliza la distancia simétrica (**Ecuación 1.10**) entre un sujeto y un experto usando como referencia la media de la distancia entre los expertos, como se puede ver en la siguiente ecuación:

$$\bar{D}_S(\lambda_{NE_i}, \lambda_{EX_i}) = \frac{D(\lambda_{NE_i}, \lambda_{EX_i})}{\frac{1}{l}\sum_{u=1;v=1}^{u=5;v=5} D(\lambda_{EX_u}, \lambda_{EX_v})} \qquad (u \neq v) \quad \textbf{(1.11)}$$

El significado práctico de esta distancia (**Ecuación 1.11**) es cómo de lejos está el sujeto de hacerlo como cada uno de los expertos. En la **Figura 1.11** se puede ver un ejemplo.

Tercera Aproximación

En el diagrama finito de estados que se mostró en la primera aproximación se representaban los posibles pasos que se pueden realizar durante una laparoscopia con una sola mano. Ahora bien, en una operación de este tipo se usan ambas manos de forma independiente. Para modelar esto se necesita un diagrama finito de estados con dos sub-diagramas, uno para la mano izquierda y otro para la mano derecha. Las transiciones entre los estados de cada sub-diagrama siguen significando lo mismo: cambios de estado/acción durante la operación.

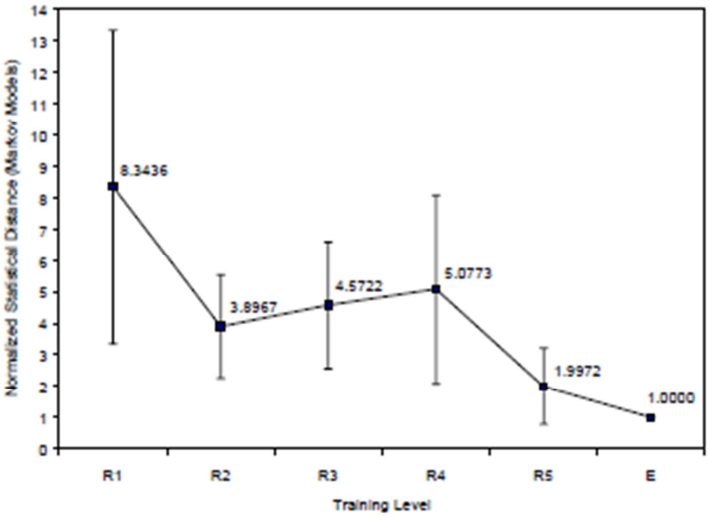

Figura 1.11: Evolución de un grupo de residentes durante 5 años usando la distancia normalizada definida en la Ecuación 1.11. Por ejemplo, durante el primer año se reduce drásticamente la distancia entre estos estudiantes y los expertos

Pero también hace falta representar el uso de ambas manos durante una misma etapa de la operación (que podrán estar haciendo acciones distintas), por ello se comunican los estados de ambos sub-diagramas entre si, como se muestra en la **Figura 1.12**.

Este nuevo modelo modifica ligeramente lo que ya se tenía en las aproximaciones anteriores. Ahora hay un *HMM* para la mano izquierda y otro para la derecha (λ_L y λ_R), y una matriz de distribución de probabilidades de transiciones entre ambos sub-diagramas (C). De forma que la probabilidad de observar un determinado estado es la siguiente:

$$P(observación|\lambda_L, \lambda_R, C) \quad \textbf{(1.12)}$$

Las probabilidades por definición tienen un rango numérico de 0 a 1. En poco tiempo la probabilidad calculada con la ecuación anterior converge exponencialmente a 0, por lo que excede el rango de precisión de casi cualquier máquina. Por esta razón se usa una transformación logarítmica, de forma que los valores anteriores de 0 a 1 se mapean en valores de -∞ a 0, dando mayor precisión en los valores que anteriormente se situaban cerca de 0.

$$Log(P(observación|\lambda_L, \lambda_R, C)) \quad \textbf{(1.13)}$$

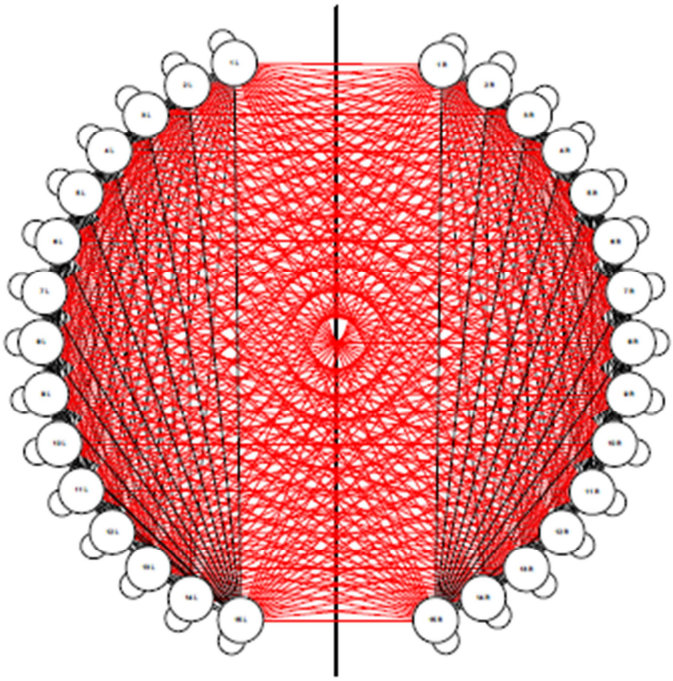

Figura 1.12: FSD que representa un ejercicio laparoscópico teniendo en cuenta el uso de ambas manos simultáneamente.

Conclusión

Al final, lo que se tiene es un sistema que evalúa, para un nuevo sujeto de estudio, si éste realiza las etapas requeridas durante la operación (siguiendo el diagrama finito de estados), tanto con la mano izquierda como con la mano derecha y, para cada etapa, si aplica la fuerza/tensión necesarias según la distancia entre el *HMM* que se obtiene de él y los *HMM*s que se obtuvieron durante el entrenamiento del sistema. Si, por ejemplo, obtienen las menores distancias con el grupo experto durante cada una de las etapas de la laparoscopia, se puede afirmar que tiene un nivel de destreza experto.

1.2.3.2. Psychomotor Skills Assessment in Laparoscopic Surgery Using Augmented Reality Scenarios

[**15**]

Este sistema mide para cada sujeto la trayectoria que realiza con la herramienta de laparoscopia para tocar dos esferas. Se utiliza *Dynamic Time Warping* (*DTW*) para sincronizar las trayectorias de los sujetos (no todos tardan lo mismo) y para obtener una medida, invariante al tiempo, de la similitud entre ellas.

Para cada una de las 16 trayectorias obtenidas por el entrenamiento se obtiene un *HMM* [**Anexo**] y luego se clasifica en cada grupo según la distancia estadística (o similitud) entre ambos (**Ecuación 1.11**). Las distancias entre novatos y expertos se promedian y, con las dos distancias resultantes (de grupo novato a grupo experto y de grupo experto a grupo novato), se obtiene la distancia simétrica (**Ecuación 1.10**) que permite clasificar nuevos sujetos.

Para evaluar un nuevo sujeto, se obtienen sus *HMM* y se calcula la distancia simétrica a ambos grupos de entrenamiento. La distancia más pequeña indicará que el sujeto pertenece a ese nivel de destreza.

1.2.3.3. Modeling and Evaluation of Surgical Performance Using Hidden Markov Models

[**20**]

Su principal diferencia respecto al método [**24**] radica en que no especifica los estados o etapas de la laparoscopia, sino que dejan que el *HMM* los obtenga durante la fase de entrenamiento, convergiendo a una definición de los estados y de las transiciones que describan de la mejor forma posible la laparoscopia.

Se usa el simulador de laparoscopia *LapSim Basic Skill 1.5* para adquirir las medidas. Los datos proporcionados consisten en dos series temporizadas de puntos 3D, una para cada instrumento, que describen la trayectoria seguida por el instrumental durante un determinado ejercicio.

Para obtener el *HMM* se usa primero una *Short-Time Fourier Transform* (*STFT*) para extraer las características más importantes de los gestos realizados a partir de los datos antes mencionados.

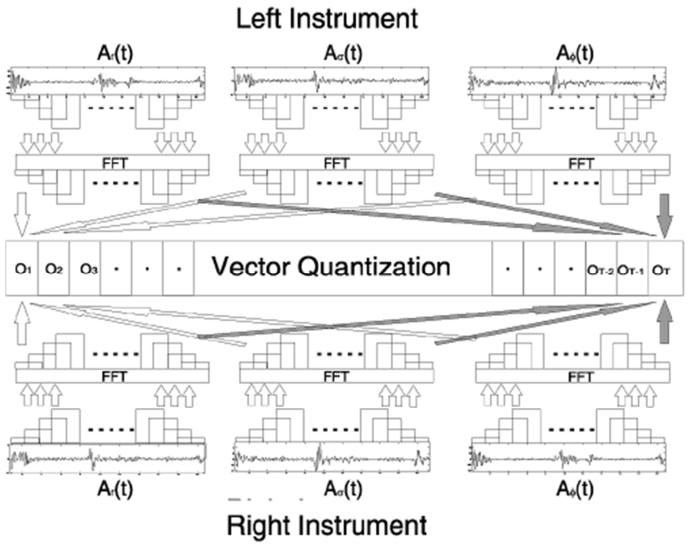

Figura 1.13: Proceso de obtención de datos, a partir de las medidas tomadas en ambas manos y en las tres coordenadas.

El resultado de aplicar *STFT* es que para cada instante de tiempo se obtienen 6 N-tuplas (N frecuencias por cada coordenada, 2 manos * 3 coordenadas). Sin embargo, para usar estos datos en un *HMM* se necesita tener sólo una dimensión, por lo que se usa un algoritmo de clustering (*K-Means*) que devuelve un valor representativo para cada conjunto de datos. Se puede observar todo este proceso en la **Figura 1.13**.

Para modelar el sistema experto se probaron varias topologías usando las medidas obtenidas por dos expertos. Se calcula la distancia entre ambos en cada topología (D_{sm} en la **Ecuación 1.14**) y al final se usa la de menor distancia.

$$D_{sm}\,(\lambda_{EX1},\lambda_{EX2}) = \frac{\frac{D\,(\lambda_{EX1},\lambda_{EX2})}{D_m\,(\lambda_{EX1},\lambda_{EX1})} + \frac{D\,(\lambda_{EX2},\lambda_{EX1})}{D_m\,(\lambda_{EX2},\lambda_{EX2})}}{2} \tag{1.14}$$

$$D_m\,(\lambda_{EX},\lambda_{EX}) = \frac{1}{nT_{O_{EX}}} \sum_{i=2}^{n+1} |\log P(O_i^{EX}\,|\lambda_{EX}) - \log P(O_{i-1}^{EX}\,|\lambda_{EX})|$$

$$D\,(\lambda_{EX1},\lambda_{EX2}) = \frac{1}{T_{O_{EX2}}}\,[\log P(O^{EX2}\,|\lambda_{EX1}) - \log P(O^{EX2}\,|\lambda_{EX2})]$$

Sólo queda definir una métrica para la evaluación de la destreza laparoscópica. Para ello, primero se necesita la medida de similitud logarítmica que da la probabilidad de que la observación (O) haya sido generada por el sistema experto (λ_{EX}). Además, para evitar que esta medida se vea influida por el tiempo que haya tardado, la observación se normaliza:

$$LL_m(O,\lambda_{EX}) = \frac{1}{n}\sum_{i=1}^{n}[\log P(\hat{O}_i^{EX}|\lambda_{EX})] \tag{1.15}$$

Y a continuación se usa esta medida en la métrica de evaluación:

$$S_m(O,\lambda_{EX}) = \frac{|\log P(O,\lambda_{EX}) - LL_m(O,\lambda_{EX})|}{\frac{1}{n}\sum_{i=1}^{n}|\log P\left(\hat{O}_i^{EX}|\lambda_{EX}\right) - LL_m(O,\lambda_{EX})|} \tag{1.16}$$

Con esta métrica se puede obtener lo cerca que está un nuevo sujeto, a partir de los datos de las trayectorias que ha seguido con ambas manos, de hacerlo como los expertos que se usaron de entrenamiento.

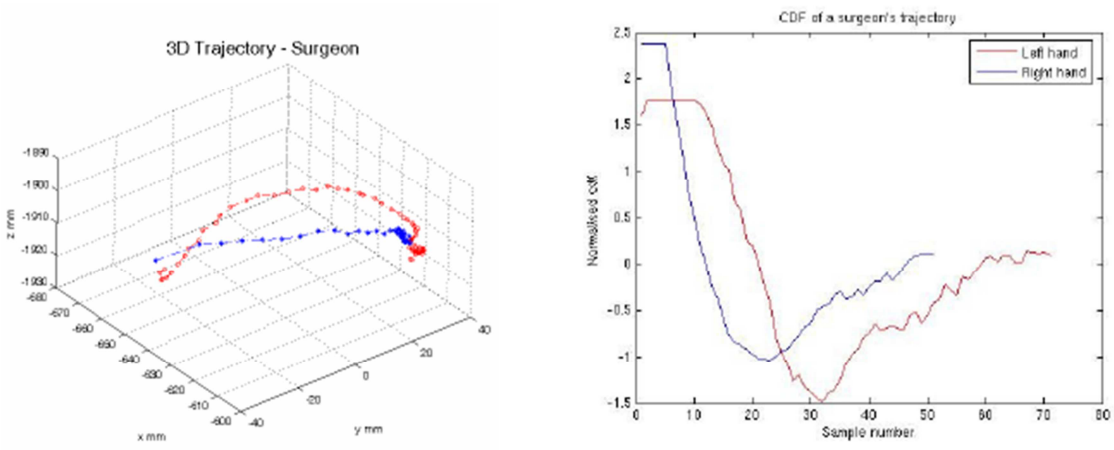

Figura 1.14: En la gráfica de la izquierda se muestran las trayectorias 3D tomadas de un experto para mano izquierda y derecha. En la gráfica de la derecha se muestra la representación CDF de estas.

1.2.3.4. HMM Assessment of Quality of Movement Trajectory in Laparoscopic Surgery

[**16**]

Como datos, este sistema usa las trayectorias 3D seguidas por el instrumental durante varios ejercicios. Estas trayectorias se mapean a una representación invariante a la vista basada en *Centroid Distance Function* (*CDF*) que normaliza los puntos respecto del centroide de la trayectoria (**Figura 1.14**). El *CDF* de cada trayectoria se usará como *input* del *HMM*.

Se usa una topología de *HMM* con un número fijo de estados. Las probabilidades de observación se modelan con *Gaussian Mixture Model* (*GMM*). Para inicializar los parámetros de este *GMM* se usa el algoritmo *K-Means* (para obtener las medias y covarianzas). Se usa también el algoritmo *Expectation Maximization* (*EM*) para calcular la máxima similitud de los parámetros del *HMM* y del *GMM*.

Con una serie de datos de expertos se entrena un *HMM* que permitirá que, para un nuevo sujeto, se pueda calcular la similitud entre este y los expertos mediante distancias (igual que en [**24**]). En la **Figura 1.15** se puede apreciar la diferencia entre la trayectoria de un estudiante, la de los profesores y la aprendida por el *HMM*.

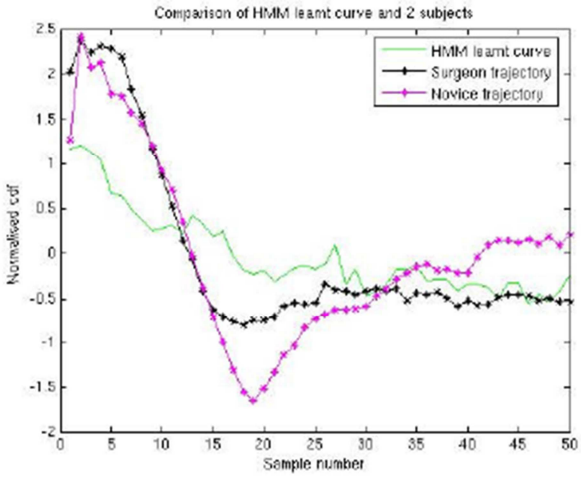

Figura 1.15: Trayectorias medias CDF seguidas por expertos (en negro) y novatos (en rosa), mientras que en verde es la trayectoria aprendida por el HMM.

1.2.4. Métodos Basados en Clasificadores Fuzzy

1.2.4.1. Fuzzy Set Theory for Performance Evaluation in a Surgical Simulator

[7]

Se obtienen tres medidas: tiempo requerido para completar la prueba, deformación máxima de la superficie (o forzado máximo del hilo, según la prueba), y número de errores. Lo primero que se hace, ya que estas medidas tienen unidades y rangos distintos, es normalizar estas medidas para que su rango vaya de 0 a 1.

Para categorizar cada variable de entrada se definen tres niveles: low, medium y high. Para obtener este nivel se usa *Fuzzy Logic Toolbox* de *Matlab*, entrenado con unas muestras de entrenamiento (producidas por 8 expertos, 8 intermedios y 8 novatos), con el que además se obtienen los centros de cada nivel (**Figura 1.16**).

Para cada sujeto de entrenamiento, perteneciente a uno de los tres niveles de destreza (novatos, intermedios y expertos), se obtienen tres niveles, uno para cada uno de las tres medidas (**Figura 1.17**). Gracias a estos niveles se pueden crear reglas que permiten clasificarlo correctamente dentro del grupo al que pertenece como, por ejemplo: *si TIEMPO es LOW, DEFORMACION_MAX es LOW y ERRORES es MEDIUM, entonces su puntuación es la de un EXPERTO.*

Si se crean reglas para todos los sujetos de entrenamiento y se juntan todas, se obtienen algunas reglas repetidas. Si una regla se repite significa que es buena para clasificar correctamente para ese nivel de destreza, por lo que se le asigna un peso mayor (según cuántas veces ha aparecido). También existirán reglas que se contradigan, pero éstas se anularán durante el proceso *Fuzzy*. Este conjunto de reglas formarán, junto con sus pesos, el conjunto de reglas del modelo *Fuzzy* que se usará.

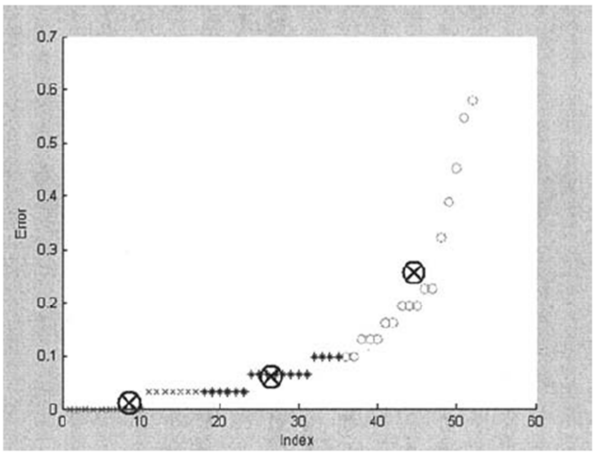

Figura 1.16: Clústeres formados para la variable de número de errores.

	Time	Max thread overstretch	Number of errors
Performance value	0.067	0	0.064
Class of data	Low	Low	Medium

Figura 1.17: Categorizado de los parámetros obtenidos para la clase experta.

Para un nuevo sujeto, basta con obtener los tres niveles (low, médium o high) de los tres parámetros medidos y pasarlos por el modelo *Fuzzy*, que devolverá una conclusión teniendo en cuenta el conjunto de reglas obtenidas en el entrenamiento.

1.2.4.2. Fuzzy Classification: Towards Evaluating Performance on a Surgical Simulator

[**11**]

Este método es parecido al anterior [**7**]. Con un simulador de realidad virtual se obtiene: tiempo para completar la tarea, número de errores producidos, economía de los movimientos y puntuación final (obtenida con las tres medidas anteriores y sus respectivos pesos).

En este método se crea un clasificador *Fuzzy* con un conjunto de entrenamiento compuesto por sujetos con unos niveles de destreza conocidos: novatos, intermedios, y expertos. Para crear dicho clasificador se usa la herramienta de *Matlab's Fuzzy Toolbox* (basado en *Fuzzy C-Means*) llamada *Neuro-Fuzzy Inference System, que* usa una red neuronal para entrenar el clasificador gracias a la habilidad de aprendizaje de estos sistemas para detectar patrones.

Al clasificador *Fuzzy* resultante de este proceso se le pueden pasar nuevas muestras de estudiantes y devolverá el nivel de destreza de dichas muestras clasificándolas en una de las tres clases.

1.3. Estado del Arte: Búsqueda de Similitud entre Cadenas

Como se ha podido ver en la sección anterior sobre evaluadores de destreza, no existe en la actualidad ningún sistema que sea capaz de proporcionar la evaluación durante la realización del ejercicio, sino que todos ellos operan con la información obtenida del ejercicio una vez éste ha finalizado.

Dado que adaptar cualquier de estas técnicas de evaluación al funcionamiento interactivo podría resultar muy complejo (en muchos casos imposible), se vio la necesidad de buscar otra técnica de evaluación. Uno de los métodos que se barajaron fue el cálculo de distancias (o similitud) entre cadenas. La idea es representar los movimientos que realice el estudiante durante el ejercicio mediante caracteres, que serán comparados con movimientos que realizó el experto.

El problema es que estos métodos de búsqueda de similitud entre cadenas nunca han sido utilizados para la evaluación de destreza. Tampoco se han utilizado estos algoritmos trabajando carácter a carácter, sino que suelen hacerlo con las cadenas ya completas de antemano. Por esta razón, hace falta estudiar dichos métodos para ver si realmente son adecuados para la evaluación interactiva de destrezas.

A continuación se describen las principales métricas que se utilizan para calcular la similitud entre dos cadenas y algunos de los algoritmos más importantes de una de ellas (que será la que se use en este proyecto).

1.3.1. Approximate String Matching

[**8**]

Se trata de una rama del problema de *String Matching* en la que se permite un cierto error en la búsqueda, dando lugar a cadenas que, si no son iguales (*Exact String Matching*, que sería la otra rama), se parecen bastante al patrón de búsqueda.

Dada una cadena s sacada de un conjunto S de posibles cadenas (compuestas por caracteres de un alfabeto A), el objetivo es encontrar una cadena t que se aproxime a s. La tarea consiste en obtener el subconjunto T de cadenas de S que sean suficientemente parecidas o encontrar las N cadenas de T más parecidas.

Los términos "*suficientemente parecidas*" o "*más parecidas*" son subjetivos, hace falta una función de distancia o métrica que, dadas dos cadenas, nos diga cómo de parecidas son. Visto de otra manera, se necesita saber cuántos errores contiene una de las cadenas suponiendo que es la otra que ha sido corrompida.

1.3.2. Edit Distance

[**32**] [**22**]

Una forma de ver la similitud entre dos cadenas es cuántos errores hacen que una de las cadenas no sea completamente igual a la otra. Si se quieren corregir estos errores se han de realizar una serie de operaciones (por ejemplo eliminar algún carácter) que harán que sean iguales.

La distancia de edición entre dos cadenas X e Y se define como el mínimo número de operaciones elementales (insertar, borrar, sustituir y, en algunos casos, transponer) requeridas para transformar la cadena origen X en la cadena destino Y.

Cada operación elemental tiene un coste (por ejemplo: la sustitución en algunas métricas suele valer el doble que la inserción y el borrado), por lo que la distancia de edición puede verse como el mínimo sumatorio de los costes de las operaciones requeridas para transformar X en Y.

Si todas las operaciones tienen un coste diferente se le llama *General Edit Distance*, mientras que si todas las operaciones valen 1 se le llama *Simple Edit Distance* (o directamente *Edit Distance*). Otra variedad es la *Normalized Edit Distance*, en la que el coste de las operaciones tiene que ver con la longitud de las cadenas. Para esta última distancia no es lo mismo tener que realizar 2 operaciones en cadenas de longitud 3, que estas mismas 2 operaciones en cadenas de longitud 9, ya que en el primer caso previsiblemente la cadena será muy diferente a la original.

Según qué operaciones se permitan y el coste de éstas se definen unas métricas. Las más famosas, que se verán en el siguiente punto, son las siguientes: Levenshtein Distance, Damerau-Levenshtein Distance, Hamming Distance y Longest Common Subsequence.

1.3.3. Métricas

1.3.3.1. Levenshtein Distance

[**22**]

Es la más común de las métricas, por lo que muchas veces se la suele llamar directamente Distancia de Edición. Esta métrica permite las siguientes operaciones elementales: inserción, borrado y sustitución de un único carácter. En su forma básica, las tres operaciones tienen un coste de 1. Por ejemplo, la distancia de Levenshtein entre las cadenas "*KITTEN*" y "*SITTING*" es de 3:

1. *KITTEN* --> *SITTEN* (sustitución de *S* por *K*)
2. *SITTEN* --> *SITTIN* (sustitución de *E* por *I*)
3. *SITTIN* --> *SITTING* (inserción de *G* al final)

Para obtener esta distancia se suele usar un algoritmo de programación dinámica, que se apoya en una matriz de $(n+1)x(m+1)$, siendo n y m las longitudes de las cadenas X e Y respectivamente. En la **Figura 1.18** se muestra el pseudo-código de dicho algoritmo.

En la **Tabla 1.1** se muestra la matriz obtenida tras calcular la distancia de Levenshtein con el algoritmo anterior para las cadenas "*KITTEN*" y "*SITTING*". Las celdas de color gris son igualdades $(X[i] = Y[j])$, las de color azul sustituciones $(M[i-1][j-1] + 1)$, y las de color verde inserciones $(M[i][j-1] + 1)$. Por último, en rojo se puede ver que la distancia de Levenshtein.

```
Desde i = 0 hasta n
        M[i][0] := i
Desde j = 0 hasta m
        M[0][j] := j
Desde i = 1 hasta n
        Desde j = 1 hasta m
                Si X[i] = Y[j]  ----> coste := 0
                Sino   ----> coste := 1
                M[i][j] := mínimo (M[i-1][j] + 1,          // borrado
                                   M[i][j-1] + 1,          // inserción
                                   M[i-1][j-1] + coste)    // sustitución
distancia := M[n][m]
```

Figura 1.18: Algoritmo para calcular la distancia de Levenshtein.

		k	**i**	**t**	**t**	**e**	**n**
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	**3**

Tabla 1.1: Matriz de distancias Levenshtein obtenida para las cadenas "*KITTEN*" y "*SITTING*".

La distancia de Levenshtein además tiene algunas características interesantes como que: siempre será al menos la diferencia entre los tamaños de ambas cadenas, será como mucho el tamaño de la cadena más larga y será 0 si y sólo si las cadenas son iguales

El algoritmo básico, mostrado anteriormente, tiene un coste de $O(nm)$, aunque existen varias optimizaciones que permiten reducir este coste. Muchas de estas optimizaciones se verán en el punto sobre algoritmos de *LCS* (*Longest Common Subsequence*), ya que ambos se basan en la programación dinámica y sus algoritmos no son demasiado distintos.

1.3.3.2. Damerau-Levenshtein Distance

[**6**]

Esta métrica permite, además de las operaciones de inserción, borrado y sustitución de un único carácter, la transposición (o intercambio) entre dos caracteres adyacentes. Es esta última operación lo que la diferencia de la distancia de Levenshtein y fue introducida por el autor, *Frederick J. Damerau*, porqué según decía las 4 operaciones permiten corregir más del 80% de los fallos humanos en escritura con una sola operación.

El algoritmo (**Figura 1.19**) es parecido al de la distancia de Levenshtein, mostrado anteriormente, pero con unas modificaciones en el bucle principal (marcadas en rojo) para incorporar la nueva operación.

```
Desde i = 1 hasta n
        Desde j = 1 hasta m
                Si X[i] = Y[j]  ----> costeS := 0
                Sino   ----> costeS := 1
                Si X[i] = Y[j-1] y X[i-1] = Y[j]  ----> costeT := 1
                Sino  ----> costeT := ∞
                M[i][j] := mínimo(M[i-1][j] + 1,              // borrado
                                  M[i][j-1] + 1,              // inserción
                                  M[i-1][j-1] + costeS,       // sustitución
                                  M[i-2][j-2] + costeT)       // transposición
distancia := M[n][m]
```

Figura 1.19: Algoritmo para calcular la distancia de Damerau-Levenshtein.

1.3.3.3. Hamming Distance

[**22**]

A diferencia de las anteriores, esta métrica sólo permite una operación, la de sustitución, lo que implica que ambas cadenas han de tener una misma longitud (de lo contrario habría que insertar o eliminar caracteres). Esta restricción es la culpable de que prácticamente esté en desuso, puesto que en aplicaciones de *String Matching* las cadenas no suelen tener la misma longitud.

La distancia de Hamming entre dos cadenas X e Y es el número de elementos en los que difieren. Por ejemplo, la distancia de Hamming entre las cadenas *TONED* y *ROSES* es de 3, ya que hay que sustituir en la primera las letras *T-N-D* por las letras *R-S-S* para obtener la segunda.

1.3.3.4. Longest Common Subsequence (LCS)

[**3**]

A diferencia de las anteriores métricas, esta no permite la operación de sustitución, pero sí las de inserción y borrado. Muchos autores dicen que *LCS* es un caso especial de la distancia de Levenshtein, en el que la sustitución tiene un coste del doble que las otras dos operaciones. La siguiente ecuación muestra la relación entre ambas distancias (siendo n y m las longitudes de X e Y):

$$Levenshtein(X,Y) = n + m - 2 * |LCS(X,Y)| \qquad (1.17)$$

Dadas las cadenas de entrada *X[1..m]* e *Y[1..n]*, la sub-secuencia *S[1..s]* de *X* se obtiene borrando $m - s$ símbolos de *X*. Esta sub-secuencia será común, $CS(X, Y)$, si *S* también es una sub-secuencia de Y, obtenida borrando $n - s$ símbolos de *Y*. La $LCS(X, Y)$ será la sub-secuencia común de mayor longitud de todas las posibles entre ambas cadenas.

Para abordar el problema de la *LCS* de dos cadenas se debe tener en cuenta las siguientes propiedades:

- **Propiedad 1:** si ambas cadenas acaban en el mismo elemento, podemos eliminar éste de las dos y volver a comprobar cómo terminan. Cuando no podamos eliminar más elementos encontramos la *LCS* de las sub-cadenas resultantes y le añadimos todos los elementos que quitamos previamente, ya que al ser comunes forman parte de la *LCS* final. Por ejemplo, "*BANANA*" y "*ATANA*" terminan ambas en "*ANA*", así que se elimina de ambas. Se resuelve $LCS(BAN, AT)$, que devuelve "*A*". Por lo tanto, $LCS(BANANA, ATANA)$ es *AANA*.
- **Propiedad 2:** si las cadenas no terminan igual podemos quitar de una su último elemento, ya que si este elemento formase parte de la *LCS* ambas terminarían igual. De esta forma, la $LCS(X_n, Y_m)$ será la más larga de las dos secuencias $LCS(X_n, Y_{m-1})$ y $LCS(X_{n-1}, Y_m)$.
- **Propiedad 3:** si una de las cadenas tiene longitud nula no puede haber una sub-secuencia común.

Gracias a estas tres propiedades se puede definir la función *LCS* de la siguiente forma (siendo x_i el carácter *i*-ésimo de la cadena *X* e y_j el carácter *j*-ésimo de la cadena *Y*):

$$LCS(X_i, Y_j) = \begin{cases} 0 & , \quad si\ i = 0\ o\ j = 0 \\ (LCS(X_{i-1}, Y_{j-1}), x_i) & , \quad si\ x_i = y_j \\ longest\left(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)\right), & \quad si\ x_i \neq y_j \end{cases} \qquad \textbf{(1.18)}$$

Dado que la función anterior **(1.18)** opera con cadenas de menor tamaño cada vez, va reduciendo un problema mayor (encontrar la *LCS* entre *X* e *Y*) en problemas más pequeños (encontrar *LCS* de sub-cadenas), se suele utilizar programación dinámica para su resolución.

1.3.4. Algoritmos LCS

[29] [30]

1.3.4.1. Wagner-Fischer, 1974

[33]

El primer algoritmo que surgió para resolver el problema de encontrar la *LCS* hace uso de la técnica de programación dinámica. Su objetivo es rellenar una tabla con la que obtener luego tanto la distancia *LCS* (evaluando la última celda) como la propia cadena *LCS* (aplicando *back-tracking*).

Las celdas de dicha tabla pueden verse como vértices de un grafo y las dependencias entre los valores de la tabla (de dónde viene su valor) como aristas. Por esta razón, muchas veces el problema de *back-tracking* (encontrar la cadena *LCS*) se resuelve mediante la búsqueda del camino más corto en dicho grafo que llegue al último vértice (la última celda).

Cada celda (i, j) contiene la longitud de la *LCS* entre las sub-cadenas *X[1..i]* e *Y[1..j]*. Su valor se obtiene mediante la siguiente ecuación (basada en las tres propiedades de la *LCS*):

$$R[i][j] = \begin{cases} 0 & , \quad si\ i = 0\ o\ j = 0 \\ R[i-1][j-1] + 1 & , \quad si\ x_i = y_j \\ \max(R[i-1][j], R[i][j-1]), & si\ x_i \neq y_j \end{cases} \qquad \textbf{(1.21)}$$

Una vez rellenada la tabla, la longitud de la *LCS* (la distancia *LCS*) de las cadenas originales *X[1..m]* e *Y[1..n]* se encuentra en la celda $R[m][n]$.

La propia cadena *LCS* se obtiene realizando *back-tracking* desde $R[m][n]$. En muchos de los algoritmos que se verán en este documento la fase de llenado de la tabla no se modifica, aunque algunos aplican un pre-procesamiento para reducir su coste de obtención, sino que es ésta estrategia de *back-tracking* lo que se optimiza para mejorar el coste de encontrar la cadena *LCS*.

En este caso, el algoritmo parte de $R[m][n]$ y comprueba de cuál de estos tres vecinos proviene su valor: de la izquierda (cuando $X[i] \neq Y[j]$ y $R[i][j-1]$ fue el máximo), de arriba (cuando $X[i] \neq Y[j]$ y $R[i-1][j]$ fue el máximo), o de arriba a la izquierda (cuando $X[i] = Y[j]$). El vecino que haya producido el valor de *R[m][n]* será el siguiente elemento en camino seguido por el *back-track*. Este proceso se repetirá hasta llegar a un elemento de la primera fila o columna.

En la **Tabla 1.2** se muestra un ejemplo de la matriz de distancias obtenida para las cadenas "*ABCDBB*" y "*CBACBAABA*". En ella se pueden observar varias cosas:

- En rojo se marcan todos los elementos en los que se cumple $X[i] = Y[j]$ (su valor proviene del vecino superior-izquierdo), llamados *k-matches*.

			C	***B***	***A***	***C***	***B***	***A***	***A***	***B***	***A***
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	0	0	0	0	0	0	0	0	0	0	0
A	**1**	0	0	0	**1**	1	1	**1**	**1**	1	**1**
B	**2**	0	0	**1**	1	1	**2**	2	2	**2**	2
C	**3**	0	**1**	1	1	**2**	2	2	2	2	2
D	**4**	0	1	1	1	2	2	2	2	2	2
B	**5**	0	1	**2**	2	2	**3**	3	3	**3**	3
B	**6**	0	1	**2**	2	2	**3**	3	3	**4**	4

Tabla 1.2: Matriz de distancias LCS obtenida para las cadenas "*CBACBAABA*" y "*ABCDBB*".

- Hay unos *k-matches* dominantes, los de la celda coloreada, que cumplen que no hay más *k-matches* de su misma clase *k* ni a su izquierda (en la misma fila) ni arriba (en la misma columna).
- La longitud de la *LCS* es 4, que es el valor contenido en la celda $R[6][9]$.

Del camino obtenido por el *back-track* sólo interesan, para obtener la cadena *LCS*, aquellos vértices o celdas cuyo valor proviene del vecino superior-izquierdo (los *k-matches*). Además, se cumple que el algoritmo de *back-track* pasa por un único *k-match* dominante por clase. Son estos *k-matches* dominantes los que indican qué caracteres pertenecen a la cadena *LCS*. El pseudo-código del algoritmo de *back-track* de Wagner-Fischer se puede ver en la **Figura 1.20**.

Los costes computacionales para la obtención de la tabla de distancias de este algoritmo suponen el límite superior del problema *LCS* (el resto de autores han buscado reducirlos), y son los siguientes:

- **Coste Temporal:** la condición de igualdad $X[i] = Y[j]$ se evalúa nxm veces dentro de un bucle anidado, por lo que el coste temporal es de: $O(nm)$.
- **Coste Espacial:** dado que hay que almacenar toda la matriz en memoria para ir operando con ella, el coste espacial también es de: $O(nm)$.

```
i := m
j := n
Mientras i > 0 y j > 0 Hacer
        Si (i, j) proviene de (i-1, j)
          i := i-1
        Sino Si (i, j) proviene de (i, j-1)
        j := j-1
          Sino
                    Imprimir (i, j)
                    i := i-1
                    j := j-1
```

Figura 1.20: Algoritmo de Wagner-Fischer para calcular la distancia LCS.

El algoritmo de *back-track* de este autor ha sido tomado como base por otros autores (Ukkonen [**32**], Myers [**21**] y Wu [**34**]) que han intentado optimizar la obtención de la cadena LCS. Pero como en este proyecto sólo se ha utilizado la versión más básica de *back-track*, se ha decidido no incluir en esta memoria la descripción de estos algoritmos.

1.3.4.2. Hirschberg, 1975

[**9**]

En el algoritmo de Wagner-Fischer, cuando se está trabajando con celdas de una determinada fila *i* de la matriz, sólo se accede a información de dicha fila (para leer el valor de $R[i][j-1]$) y de la anterior (para leer los valores de $R[i-1][j-1]$ y $R[i-1][j]$).

Teniendo en cuenta esto, si se modifica el algoritmo para que, en lugar de trabajar con una matriz de mxn todo el rato, trabaje con dos filas de tamaño m (la fila actual y la anterior) se reducirá el coste espacial.

Adicionalmente, sobre la optimización anterior se puede aplicar una estrategia de *divide and conquer*, que consiste en dividir recursivamente la cadena X en dos mitades y, para cada mitad, ($X1$ y $X2$) encontrar el prefijo y sufijo correspondientes de Y ($Y1$ e $Y2$) para los que la $LCS(X1, Y1)$ concatenada con la $LCS(X2, Y2)$ es igual a la LCS original:

$$LCS(X1, Y1) \cdot LCS(X2, Y2) = LCS(X, Y) \qquad \textbf{(1.22)}$$

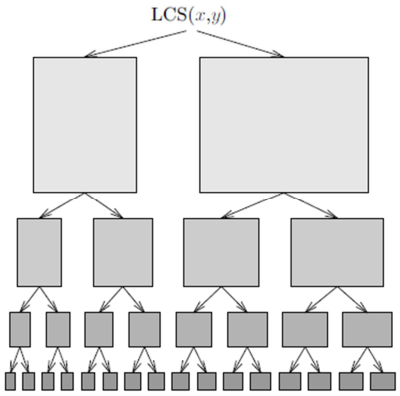

Figura 1.21: Funcionamiento de la estrategia 'divide and conquer' aplicada a la búsqueda de la cadena LCS. El problema original se descompone en problemas más pequeños y fáciles de resolver.

El teorema seguido por el autor dice que cuando X se divide en dos mitades, en cualquier punto de la recursión, el valor máximo (a través de todas las divisiones de Y en dos mitades) de la suma de las longitudes de una LCS de los prefijos ($X1$ e $Y1$) y de los sufijos ($X2$ e $Y2$) es igual a la longitud de la LCS original (X e Y).

La subdivisión (**Figura 1.21**) termina al llegar a uno de los casos triviales: se alcanza una longitud de cadena indivisible (0 o 1). Entonces se deshace la recursión, concatenando las $LCS(X1, Y1)$ y $LCS(X2, Y2)$ máximas, hasta llegar a $LCS(X, Y)$.

El resultado de aplicar ambas optimizaciones es que el coste temporal no se modifica, sigue siendo $O(nm)$, pero el coste espacial pasa de ser cuadrático a ser lineal: $O(n + m)$. Esto hace que este algoritmo sea especialmente interesante si se va a trabajar con cadenas de gran tamaño, que en el algoritmo de Wagner-Fischer daría lugar a una matriz demasiado grande.

1.3.4.3. Hunt-Szymanski, 1976

[**12**]

La clave de este algoritmo es el uso de un vector de valores umbral, T, en el que $T_{i,k}$ contiene la mínima j para la que $X[1:i]$ e $Y[1:j]$ contienen una sub-secuencia común de longitud k. Es decir, esta $T_{i,k}$ se obtiene mediante la siguiente fórmula:

$$T_{i,k} = min\{\, j \; tal \; que \; |LCS(X[1:i], Y[1:j])| = k \,\} \qquad \textbf{(1.23)}$$

Por ejemplo, para las cadenas "*ABCBDDA*" y "*BADBABD*" se tiene que:

$$T_{5,1} = 1$$

$$T_{5,2} = 3$$

$$T_{5,3} = 6$$

$$T_{5,4} = 7$$

$$T_{5,5} = sin\, definir$$

Cada $T_{i,k}$ puede verse como un puntero que dice cuánto de *Y* se necesita para producir una subsecuencia común de longitud k con los *i* elementos primeros de *X*. Además, se cumple que los valores de *T* son estrictamente incrementales para una misma *i*:

$$T_{i,1} < T_{i,2} < \ldots < T_{i,p} \qquad \textbf{(1.24)}$$

Si en un momento dado se tienen calculados todos los valores k para una i dada, $T_{i,k}$, y se quieren obtener todos los valores de $i+1$, $T_{i+1,k}$, se cumplirá que:

$$T_{i,k-1} < T_{i+1,k} \leq T_{i,k} \qquad \textbf{(1.25)}$$

Gracias a esta propiedad se puede calcular $T_{i+1,k}$ a partir de $T_{i,k-1}$ y $T_{i,k}$ mediante la siguiente función:

$$T_{i+1,k} = \begin{cases} \min\{ j \mid x_{i+1} = y_j \ \wedge \ T_{i,k-1} < j \leq T_{i,k}\} \\ T_{i,k} \qquad\qquad , \quad si\ no\ existe\ dicha\ j \end{cases} \qquad \textbf{(1.26)}$$

El algoritmo desarrollado por los autores obtiene el vector *T* mediante esta función. Una vez calculado *T* se puede devolver la longitud de la *LCS* buscando la k más grande, $T[k]$, en dicho vector que tenga un valor definido, es decir, que tenga alguna *j* asignada.

Este método reduce el coste temporal de $O(nm)$, de los anteriores métodos, a $O((r+n)\log n)$ para cadenas de igual longitud y usando un alfabeto finito. Siendo r el número de pares (i,j) en los que se cumple $x_i = y_j$ (*matches*).

En el peor caso, en el que todos los elementos de *X* hacen *matching* con los de *Y*, el coste seria de $O(n^2 \log n)$. Sin embargo, en la mayoría de aplicaciones reales r es cercana a n, por lo que el coste medio es $O(n \log n)$. El coste espacial es de $O(r+n)$.

El algoritmo de *back-track* de estos autores fue tomado como referencia por Kuo-Cross [**14**] para intentar mejorar la obtención de la cadena LCS.

1.3.4.4. Hirschberg, 1977

[**10**]

El autor hace uso de dos propiedades de la matriz de distancias *LCS*: los valores se incrementan monótonamente en *i* y en *j*, y las entradas adyacentes difieren como mucho en una unidad. Gracias a estas propiedades la tabla puede ser dividida en regiones disjuntas, una para cada valor de k desde 0 hasta p, siendo $p = |LCS(X, Y)|$.

El contorno entre la región k y la región $k - 1$ está definido por el conjunto de *k-candidatos* posicionados en las esquinas convexas de dicho contorno, como se puede ver en la **Tabla 1.3**.

Para almacenar estos *k-candidatos* mínimos se usa una matriz *D*, en la que $D[k][i]$ es el valor *j* del *k-candidato* mínimo y único con un valor *i* o 0 si no hay *k-candidato* mínimo. De esta forma $D[k][i]$ describe los contornos dando el número de la primera columna de la fila *i* que está en la región k si el número es diferente al de $D[k][i-1]$.

El algoritmo del autor obtiene esta matriz *D* y luego la recorre para recuperar la sub-secuencia común. Para ello, basta con obtener los índices *i* de aquellos *k-candidatos* mínimos que generaron un *(k+1)-candidato* mínimo hasta llegar a la distancia de la *LCS*. Este método consigue encontrar la *LCS* con un coste temporal $O(pn + n \log n)$, siendo *p* la longitud de la cadena *LCS*.

Este algoritmo de *back-track* ha sido tomado como base por Apostolico-Guerra [**2**], que han mejorado el coste obtención de la cadena LCS.

	c	**b**	**a**	**c**	**b**	**a**	**a**	**b**	**a**
a	0	0	1	1	1	1	1	1	1
b	0	1	1	1	2	2	2	2	2
c	1	1	1	2	2	2	2	2	2
d	1	1	1	2	2	2	2	2	2
b	1	2	2	2	3	3	3	3	3
b	1	2	2	2	3	3	3	4	4

Tabla 1.3: Contornos formados en la matriz de distancias LCS obtenida para las cadenas "*CBACBAABA*" y "*ABCDBB*". Cada región, desde 0 hasta *p* (siendo *p* la longitud de la cadena LCS), tiene varios k-candidatos (en gris) situados en sus esquinas convexas.

1.3.4.5. Masek-Paterson, 1980

[**19**]

El algoritmo desarrollado por estos autores usa la técnica *Four Russians* (Arlazarov et al., 1970) sobre el algoritmo desarrollado por Wagner-Fischer. Esta técnica acelera el trabajo sobre matrices, pero requiere que el alfabeto sea finito y que los costes de edición (de las operaciones elementales) sean todos múltiplos enteros de un determinado número real.

La idea principal de la técnica *Four Russians* es dividir la matriz de distancias en sub-matrices cuadradas (*blocks*) y realizar todos los cálculos posibles sobre dichas sub-matrices durante una fase de pre-procesamiento.

La sub-matriz (i, j, k), de dimensiones $(k+1)x(k+1)$, es aquella cuyo elemento de la esquina superior-izquierda es $R[i][j]$. Por ejemplo, en las **Tabla 1.4** se muestra la matriz original y en la **Tabla 1.5** las siguientes cuatro sub-matrices formadas a partir de ésta: (0,0,3), (0,3,3), (3,0,3) y (3,3,3).

La primera parte de algoritmo consiste en calcular todas las posibles sub-matrices con el alfabeto y las funciones de coste dadas. Almacenará para cada sub-matriz los vectores que definen su contorno (**Tabla 1.6**). Para poder obtener todas las sub-matrices y numerarlas es necesario que el alfabeto sea finito.

Dado que los valores de celdas adyacentes no difieren en más de una unidad, se pueden almacenar sus valores como la diferencia respecto del vecino izquierdo (*step*), pudiendo ser esta diferencia -1, +1 o 0.

Si sólo almacenamos valores de las celdas que pertenecen a un contorno o que son la esquina superior izquierda de un bloque, habrá menos información a examinar en posteriores etapas del algoritmo. De esta forma, podemos definir cada sub-matriz con su valor inicial (i, j) y con dos vectores de *steps*: uno horizontal y otro vertical. En la **Tabla 1.7** se puede ver la definición de las sub-matrices de la **Tabla 1.5** siguiendo esta notación.

	c	**b**	**a**	**c**	**b**	**a**	**a**
a	0	0	1	1	1	1	1
b	0	1	1	1	2	2	2
c	1	1	1	2	2	2	2
d	1	1	1	2	2	2	2
b	1	2	2	2	3	3	3
b	1	2	2	2	3	3	3
c	1	2	2	3	3	3	3

Tabla 1.4: Matriz de distancias LCS obtenida para las cadenas CBACBAA y ABCDBBC.

	c	b	a	c	b	a	a
a	0			1			
b							
c							
d	1			2			
b							
b							
c							

Tabla 1.5: Sub-matrices (0,0,3), (0,3,3), (3,0,3) y (3,3,3) formadas a partir de la matriz de la Tabla 1.4. La celda con número es la esquina superior izquierda de la sub-matriz.

	c	b	a	c	b	a	a
a	0	0	1	1	1	1	1
b	0			1			2
c	1			2			2
d	1	1	1	2	2	2	2
b	1			2			3
b	1			2			3
c	1	2	2	3	2	2	3

Tabla 1.6: Contornos de las cuatro sub-matrices formadas en la Tabla 1.5.

La segunda fase del algoritmo obtiene la matriz de distancias, en forma de matriz de *steps*, usando la información contenida en todas estas sub-matrices. Una vez obtenida, el coste de edición se puede calcular sumando los *steps* a lo largo de cualquier camino desde $(0, 0)$ hasta (n, m).

	c	b	a	c	b	a	a
a	0	0	+1	1	0	0	0
b	0			1			
c	+1			2			
d	1	0	0	2	0	0	0
b	0			2			
b	0			2			
c	0			3			

Tabla 1.7: Definición de las cuatro sub-matrices de la Tabla 1.5 mediante una esquina (celda gris) y dos vectores de steps (indicados con flechas): vertical y horizontal.

Para obtener la propia cadena *LCS* basta con aplicar algún algoritmo de *back-tracking*, modificado para operar con matriz de *steps* en lugar de la matriz de distancias tradicional, como por ejemplo el de Wagner-Fischer. El coste temporal del algoritmo es $O(\frac{n^2}{\log n})$ para cadenas de igual longitud. El coste espacial también es $O(\frac{n^2}{\log n})$.

1.3.4.6. Allison-Dix, 1986

[**1**]

Los autores usan una representación binaria de la matriz de distancia *LCS* en la que los 1*s* marcan los contornos de las *k*-clases. El algoritmo computa dicha matriz por columnas, usando operaciones lógicas sobre las cadenas binarias que representan cada columna.

Por definición, los valores en una misma fila de la matriz de distancias *LCS* varían como mucho en una unidad respecto al valor de la celda anterior, esto hace que se pueda representar la matriz de distancias de forma binaria.

Además, se cumple que en cualquier fila habrá el mismo número o uno más de bits (celdas) a 1 respecto de la fila anterior. Gracias a estas dos propiedades se puede obtener la distancia *LCS* como la suma de bits a 1 de la última fila de la matriz binaria.

Durante la fase de pre-cálculo se define una cadena de bits para cada letra del alfabeto mediante la comparación de estas letras con las presentes en la cadena *X*. Esta operación tiene un coste temporal de O(n) para un alfabeto fijo. Un ejemplo de esto para la cadena X="*BTCTDAC*" puede verse en la **Tabla 1.8**.

Para obtener la fila i de la matriz binaria se usa el carácter *Y[i]* para seleccionar qué cadena binaria del alfabeto usar. Por ejemplo, si *Y* es "*ABCDEF*" y estamos trabajando con la fila 3, correspondiente a *C*, usaremos la representación obtenida en la **Tabla 1.8** de *0010001* (la de *C*)

Con esta cadena (T_i) y la de la fila anterior $(fila_{i-1})$ se aplican las siguientes operaciones binarias (siendo << un *shift* hacia la izquierda) para obtener la fila actual $(fila_i)$:

$$fila_i = T_i \; AND \; ((T_i - (fila_{i-1} \ll 1)) \neq T_i) \qquad \textbf{(1.27)}$$

Por ejemplo, si en un momento dado tenemos la $fila_{10}$ (1000000100011111) y queremos obtener la $fila_{11}$, lo que hacemos es seleccionar con el carácter *Y[11]* qué cadena binaria usar del alfabeto obtenido por comparación con *X* (T_i=*0101100010001100*). Con ambas cadenas de bits podemos obtener la $fila_{11}$ mediante la **Ecuación 1.27**.

$$fila_{i-1} = 1000000100011111$$

$$T_i = 0101100010001100$$

$$fila_i = 0000100010011111$$

0	B	T	C	T	D	A	C
A	0	0	0	0	0	1	0
B	1	0	0	0	0	0	0
C	0	0	1	0	0	0	1
D	0	0	0	0	1	0	0
…							

Tabla 1.8: Alfabeto, en forma de cadenas de bits, obtenido a partir de la cadena "*BTCTDAC*".

Dado que el valor de una fila se calcula a partir de la anterior con operaciones de bit, la obtención de toda la matriz es un proceso muy rápido. Gracias a esto, encontrar la distancia (sumar el número de bits a 1 de la última fila) tiene un coste temporal de $O(\left\lceil \frac{m}{w} \right\rceil n)$, en el que w es el tamaño de palabra del computador. También requiere de un pre-procesamiento de cada cadena de $O(\left\lceil \frac{m}{w} \right\rceil |C| + m)$, siendo C el alfabeto finito.

Si se quiere obtener también la cadena *LCS* se puede aplicar algún método de *back-track*, como el de Hirschberg o Wagner-Fischer, con el coste adicional de cómputo.

1.3.4.7. Kumar-Rangan, 1987

[**13**]

Este algoritmo usa la técnica vista anteriormente de *divide and conquer*, que irá partiendo las cadenas originales en partes más pequeñas mediante recursión.

Dadas las cadenas $A[1{:}m]$ y $B[1{:}n]$, se define $L_i(k)$ como la más larga h para la que $A[i{:}m]$ y $B[h{:}n]$ tienen una *LCS* de longitud k. Cada $L_i(k)$ puede verse como un puntero que nos dice cuánto de la parte derecha de B hace falta para producir una *LCS* de longitud k con los últimos $(m - i + 1)$ símbolos de A. No todos los $L_i(k)$ están definidos.

Análogamente, se define $L_i^*(k)$ como la más pequeña h para la que $A[1{:}i]$ y $B[1{:}h]$ tienen una *LCS* de longitud k. Cada $L_i^*(k)$ puede verse como un $L_i(k)$ pero por la parte izquierda de B.

Se define el par ordenado (u, v) como *corte válido* de la *LCS* C si existen sub-cadenas C1 y C2 que cumplen:

$$C_1 = LCS(A[1:n], B[1:v])$$
$$C_2 = LCS(A[u+1:m], B[v+1:n]) \quad \textbf{(1.28)}$$
$$C = C_1 \cdot C_2$$

Se define $W(A, B)$ como el sobrante de A respecto de la *LCS* C de A y B, es decir, el número de símbolos de A que no están en C. Se obtiene:

$$W(A, B) = |A| - |C| = m - p \quad \textbf{(1.29)}$$

El par ordenado (u, v) será un *corte perfecto* si, además de ser un *corte válido*, cumple que:

$$W\big(A(1:u), B(1:v)\big) = [\frac{m-p}{2}] \quad \textbf{(1.30)}$$

El autor prueba que para cualquier par de cadenas A y B existe un corte perfecto, y que (u, v) será dicho corte perfecto si y sólo si:

$$L_{u+1}(m - u - w') > v \geq L_u^*(u - w) \quad \textbf{(1.31)}$$

Dadas todas estas definiciones, el algoritmo, para obtener la cadena *LCS*, lo que hace es ir partiendo las cadenas originales en dos sub-cadenas mediante cortes perfectos. De cada sub-cadena se obtiene un nuevo *LCS*, que a su vez se obtendrá con otro corte perfecto de forma recursiva y se concatena con el de la otra mitad (obtenida de la misma forma). Al final se obtendrá el *LCS* de las cadenas originales como concatenación de muchos *LCS* de sub-cadenas formadas por cortes perfectos.

El coste temporal de este algoritmo es $O(n(m - p))$, donde p es la longitud de la *LCS*.

1.4. Conclusión

En el estado del arte sobre evaluadores de destreza se ha visto como no existe en la actualidad ningún evaluador preparado para trabajar de forma interactiva, proporcionando evaluaciones durante la realización de ejercicios. Por esta razón se decidió probar con el cálculo de distancias entre cadenas como técnica de evaluación de destreza.

Sin embargo, como se ha visto en el estado del arte sobre la búsqueda de similitud entre cadenas, no existe en la actualidad ninguna aplicación de estas técnicas en la evaluación de destrezas. Tampoco existe un algoritmo de cálculo de distancias entre cadenas que esté pensado para trabajar con una de las cadenas incompleta (que se está generando durante el cálculo).

Por estas razones, en este proyecto no sólo se va a desarrollar un método novedoso de cálculo de distancias interactivo, sino que además se va a aplicar dicha técnica en la evaluación de destrezas. Por ello, la fase de validación será de gran importancia.

2. Sistema de Evaluación Automática e Interactiva de Destrezas Mediante Distancias entre Cadenas

2.1. Introducción

El objetivo principal de este proyecto es desarrollar y validar un método de evaluación interactiva de destrezas mediante distancias entre cadenas, por lo que el área de mayor importancia es el cálculo de la distancia que represente el nivel de experiencia.

Sin embargo, para poder probar y validar dicho método hace falta disponer de gran cantidad de cadenas de movimientos, por lo que necesitamos de un sistema que sea capaz de proporcionar los movimientos (en forma de caracteres) durante la realización de un determinado ejercicio (para poder evaluar interactivamente). Al no disponer de un sistema que sea capaz de proporcionar este tipo de información se ha tenido que desarrollar un prototipo de simulador laparoscópico parecido a los vistos en la sección **1.2**.

Por esta razón, en esta sección se describirá en profundidad el método de cálculo de las distancias (sección **2.6**), mientras que se explicará por encima el resto de componentes que permiten realizar esta evaluación, para dar al lector una visión de todo el trabajo realizado en este proyecto.

2.2. Vista Global

Para poder evaluar correctamente el nivel de experiencia de un usuario según los movimientos que realiza, primero se necesita de un método que permita capturar dichos movimientos. El proceso encargado de transformar estos movimientos reales en información digital es el Tracker.

De nada sirve capturar los movimientos del usuario si está realizando un ejercicio diferente al que realizó el experto. Por esta razón, se necesita de un marco común de trabajo para que ambos hagan (o traten de hacer) las mismas acciones. Este marco común es un Escenario, en el que experto y estudiante realizan un mismo ejercicio con los movimientos capturados por el Tracker.

La parte principal de este proyecto es el cálculo de distancias entre cadenas de movimientos, por lo que parece obvio que se necesita de: un proceso que genere cadenas a partir de los movimientos capturados por el Tracker, y un proceso que compare los movimientos del estudiante con los que realizó el experto.

En esta sección se describirán todos estos procesos (cuyo funcionamiento puede verse en la **Figura 2.1**), poniendo especial énfasis en el cálculo de distancias entre cadenas.

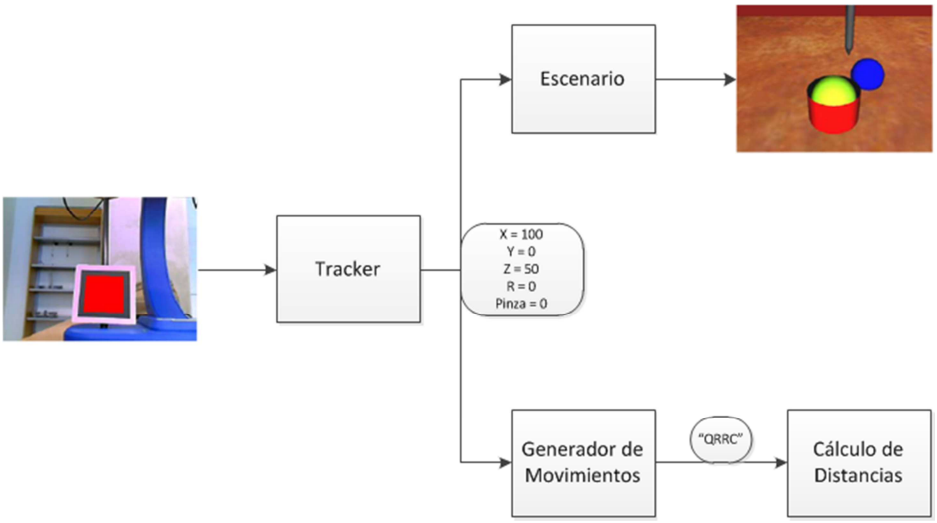

Figura 2.1: Vista global de los componentes principales del sistema desarrollado.

2.3. Tracker

Es el proceso encargado de capturar los movimientos realizados por el estudiante para poder luego transformarlos en caracteres. El tracker es una pieza importante de este proyecto, ya que influye en cómo de representativos van a ser los movimientos capturados de los que se realizaron. Si un estudiante realiza un ejercicio correctamente, pero los movimientos capturados no se corresponden a los reales, difícilmente se evaluará su destreza correctamente cuando se comparen estos movimientos con los del experto. Lo deseable es que el tracker no influya en la evaluación de destreza.

2.3.1. Funcionamiento del Tracker

En sistemas comerciales de entrenamiento en cirugía mínimamente invasiva (*MIS*) existen varios métodos de tracking [**4**], utilizados todos ellos para conocer la posición del instrumento, o instrumentos, durante la realización de un ejercicio. Lo que los diferencia entre sí es la tecnología empleada para obtener dicha posición, pudiendo ser esta: mecánica, óptica, acústica o electromagnética.

Para este proyecto se dispone de un sistema comercial de tracking mecánico llamado *Virtual Laparoscopic Interface* (**Figura 2.2**), de la empresa *Immersion*. Este sistema captura los movimientos realizados con el instrumental mediante unos sensores en el mecanismo *gimbal* que dirige los instrumentos, permitiendo así reconocer hasta cinco grados de libertad: *pitch, yaw, roll*, inserción y pinzado. Sin embargo, actualmente estos sensores no funcionan. Esto no supone problema, ya que desde un primer momento queríamos que el sistema fuese óptico.

Figura 2.2: Sistema de tracking mecánico *Virtual Laparoscopic Interface*.

Ya que otros sistemas comerciales usan tracking óptico con éxito, se decide colocar una marca en uno de los instrumentos del sistema *Virtual Laparoscopic Interface* para que pueda ser capturada con alguna herramienta software de tracking óptico y una webcam.

Existen varias alternativas de software libre para hacer tracking óptico: *OpenCV, ARToolKit, Mixed Reality Toolkit, NyARToolkit,...* Se decidió utilizar *ARToolKit* principalmente por su sencillez de uso y que se programa en *C*. Esta librería está pensada para aplicaciones de realidad aumentada, por lo que dispone de funciones que, dado un patrón (el de la marca que se coloca en el instrumental) y una imagen (la capturada), permiten obtener la posición-orientación en el mundo real de la marca.

A partir de uno de los ejemplos suministrados con las librerías se ha desarrollado una aplicación que muestra por pantalla lo que recibe la webcam, indica dónde ha localizado la marca, y va proporcionando información de posición (*X, Y, Z*) y de orientación (respecto del eje Y) de la marca (**Figura 2.3**). Todos estos datos son obtenidos a partir de la matriz 4x4 que representa la translación y rotación que ha sufrido la marca respecto de la cámara.

En el sistema comercial *Virtual Laparoscopic Interface* hay un quinto grado de libertad: el estado de la pinza. Al estar estropeado también este sensor y no poder captar el movimiento ópticamente, se optó simularlo con una tecla.

La tarea de transformar los cambios de estado en un movimiento la realizará otro módulo: el de Generación de Movimientos (se verá en la sección **2.5**). El proceso completo de obtención de los 5 grados de libertad puede verse en la **Figura 2.4**.

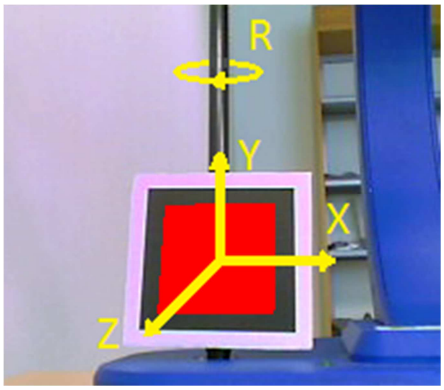

Figura 2.3: Representación de los 4 grados de libertad obtenidos con *ARToolKit* a partir de una marca colocada en el instrumento.

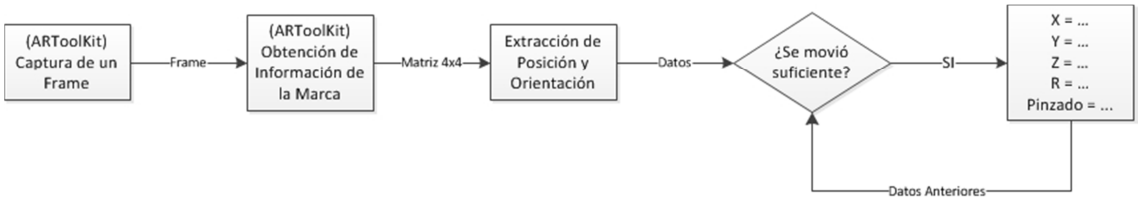

Figura 2.4: Proceso seguido por el Tracker para la obtención de los cinco grados de libertad.

2.3.2. Precisión

Un aspecto importante del tracking es la precisión con la que se detectan los cambios de posición y orientación. La librería *ARToolKit* está pensada para realidad aumentada, por lo que tiene una precisión adecuada para este tipo de aplicaciones, pero mejorable para aplicaciones de tracking 3D que requieran gran exactitud. Para llegar a la precisión requerida en este proyecto se tuvo que:

- Aumentar el tamaño de la marca (equilibrio entre tamaño y precisión adecuado).
- Recalibrar varias veces hasta conseguir unos parámetros de la cámara óptimos.
- Añadir algo de memoria al proceso de tracking, de forma que si la marca se ha movido menos de un umbral respecto de su anterior posición (inevitable) estos nuevos datos son ignorados.
- En la obtención de la rotación, aumentar el umbral de "imprecisión" en ángulos cercanos a 0º (la marca mirando a la cámara) en los que los valores devueltos por *ARToolKit* fluctúan demasiado.

2.4. Escenarios

Si se quiere evaluar la destreza de los estudiantes comparando sus movimientos con los que realizó el experto hace falta que todos ellos realicen un mismo ejercicio. Dicho de otra forma, es necesario que el estudiante trate de realizar los mismos movimientos que el experto. La forma de conseguir esto es mediante un escenario común.

La idea inicial era desarrollar un único escenario, puesto que era suficiente para lograr los objetivos del proyecto. Sin embargo, tras desarrollar dicho escenario se vio que, con unas pocas modificaciones, era posible desarrollar un segundo escenario y, por tanto, realizar una validación más amplia del proyecto.

El objetivo de estos escenarios es que el estudiante tenga que enfrentarse a los mismos problemas que el experto, para ver si los resuelven de la misma forma. Tratando de resolver estos ejercicios realizarán unos movimientos con el instrumental que serán capturados por el tracker y, posteriormente, utilizados para calcular la destreza que se está demostrando.

Para la elaboración de estos escenarios se ha tomado como referencia los de un simulador comercial llamado *LapSim*. Se han buscado escenarios que sean adecuados para un usuario no demasiado experto, pero permitiéndole demostrar experiencia en su resolución y que le obliguen a usar todos, o casi todos, los grados de libertad.

2.4.1. Escenario 1

Este escenario se basa en uno de *LapSim* llamado *Camera Navigation* (**Figura 2.5**), cuyo objetivo es que el estudiante adquiera destreza en el manejo del instrumental laparoscópico, especialmente en el uso del instrumento-cámara, que es una de las primeras destrezas que ha de adquirir el estudiante de laparoscopia.

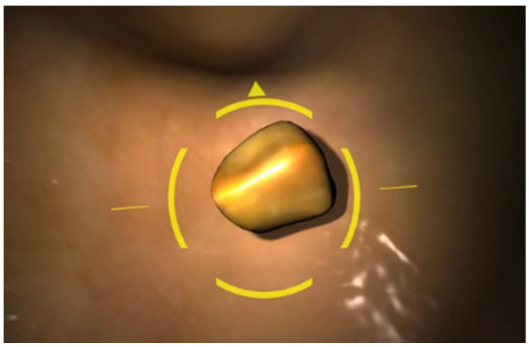

Figura 2.5: Escenario *Camera Navigation* del simulador comercial *LapSim.*

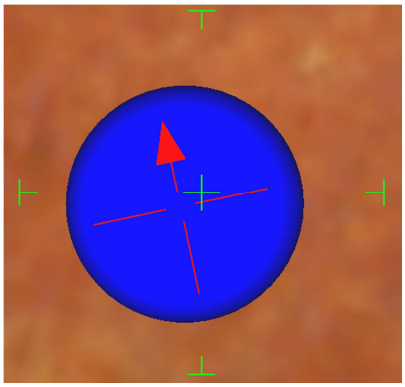

Figura 2.6: Captura del escenario 1 en la que se puede ver la mirilla (en verde) y la flecha de orientación (en rojo) que permiten al estudiante saber cuándo ha de pinzar.

Este escenario obliga al estudiante a utilizar los cinco grados de libertad ya que, no solo ha de posicionar el instrumento cerca del objeto, sino que también ha de orientarlo de forma adecuada para poder hacerlo desaparecer cuando pince.

Como dificultad añadida, en este ejercicio no se muestra dónde está el objeto a pinzar, por lo que los movimientos que realice el estudiante durante su localización también serán evaluados, pudiendo buscar con los movimientos del experto (que dicta qué está bien y qué no) una economía de movimientos o, por contra, que el usuario realice sus movimientos suficientemente alejado del suelo (que equivaldría a un órgano o tejido) para evitar dañar al paciente.

En nuestro escenario, la única ayuda de la que dispone el estudiante es una mirilla en la cámara (**Figura 2.6**) que indica cómo de cerca ha de estar del objetivo, y una flecha en la pelota que indica el ángulo adecuado. Obviamente, si el estudiante trata de pinzar un objeto sin estar colocado correctamente fallará y el tracker habrá capturado tanto este pinzado fallido como los movimientos posteriores de recolocación y reorientación. Este fallo repercutirá negativamente en la posterior evaluación del estudiante, puesto que el experto no habrá fallado.

En resumen, este escenario se centra en la precisión a la hora de posicionarse y orientarse con el instrumento. Además, permite evaluar la búsqueda y navegación hasta nuevos objetivos. Por lo tanto, se trata de un buen escenario para adquirir una destreza básica con el instrumental laparoscópico.

2.4.2. Escenario 2

Este escenario se basa en uno de *LapSim* llamado *Grasping* (**Figura 2.7**), cuyo objetivo es coger unos vasos sanguíneos con la pinza del instrumento y depositarlos en una bolsa de recuperación endoscópica. A diferencia del escenario anterior la cámara está fija, de manera que sólo se mueve el instrumento. Esto hace que tenga mayor similitud con una operación real.

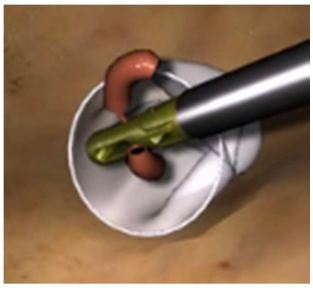

Figura 2.7: Escenario *Grasping* del simulador comercial *LapSim*.

Se ha elegido este ejercicio porque busca que el estudiante adquiera destreza en la manipulación de objetos (**Figura 2.8**), una práctica muy común en laparoscopia. De esta forma no sólo se evalúa que se realicen los movimientos correctamente, sino que además han de realizarse con la pinza en un estado determinado: cerrada (transportando un objeto) o abierta (sin objeto).

Su principal dificultad es que, al no estar colocada la cámara en el instrumento, los movimientos son más difíciles, ya que la perspectiva y lejanía de la cámara produce que muchas veces los objetos estén más cerca o lejos de lo que parecen. Si el estudiante aprende a trabajar con esta característica mejorará su destreza, puesto que una de las mayores dificultades de esta técnica quirúrgica es que los movimientos se transmiten a través de otro objeto (el instrumento) y no directamente, como cuando se usa el bisturí, dificultando así su control.

En este escenario los fallos de pinzado son mas críticos que en el anterior, puesto que se realizan más movimientos de pinzado y las transiciones se han de realizar en un determinado estado. Por ejemplo, si se despinza transportando un objeto el resto de movimientos serán erróneos (como ya se verá en la sección **2.5.1**) hasta que corrija el estado de la pinza.

En resumen, este escenario se centra en la manipulación de objetos y en acostumbrarse a los movimientos del instrumento cuando se está observando desde una cámara.

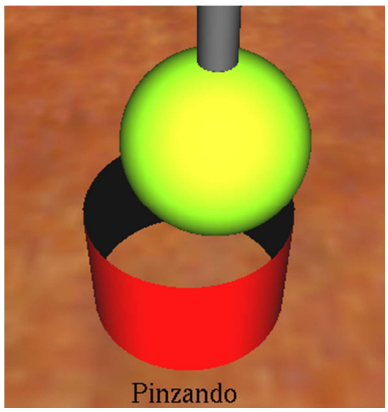

Figura 2.8: Captura del escenario 2 en la que podemos ver como el instrumento tiene una pelota cogida y está tratando de dejarla en el recipiente.

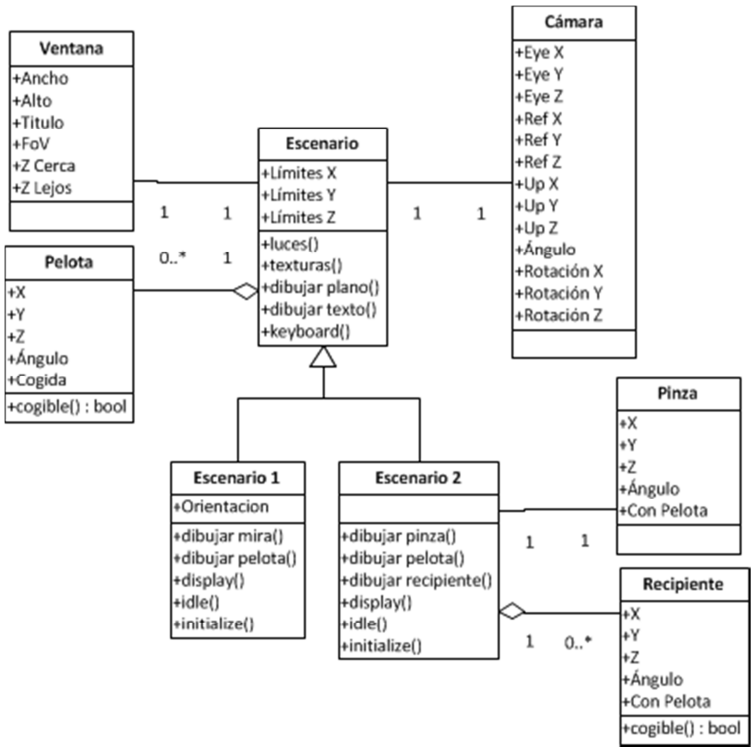

Figura 2.9: Diagrama de Clases de los escenarios.

2.4.3. Desarrollo de los Escenarios

Para el desarrollo de los escenarios se ha utilizado el lenguaje de programación *C++* y la *API* de gráficos *OpenGL*, que es una librería destinada a la creación de aplicaciones en 2D y 3D.

Como ya se comentó anteriormente, la idea inicial era desarrollar un único escenario, pero tras desarrollar el primer escenario se vio que gran parte del código podría servir para desarrollar nuevos escenarios. Por esta razón se creó una clase padre para que los hijos heredaran de ella lo común de todos los escenarios y añadieran los elementos específicos. En la **Figura 2.9** se puede ver el diagrama de clases resultante.

2.4.4. Funcionamiento de los Escenarios

Cada vez que el escenario recibe datos del tracker realiza el siguiente pre-proceso: convertir las unidades del mundo real (tracker) en unidades del mundo virtual (escenario), y comprobar que no se salgan de rango (p.e.: traspasar el suelo).

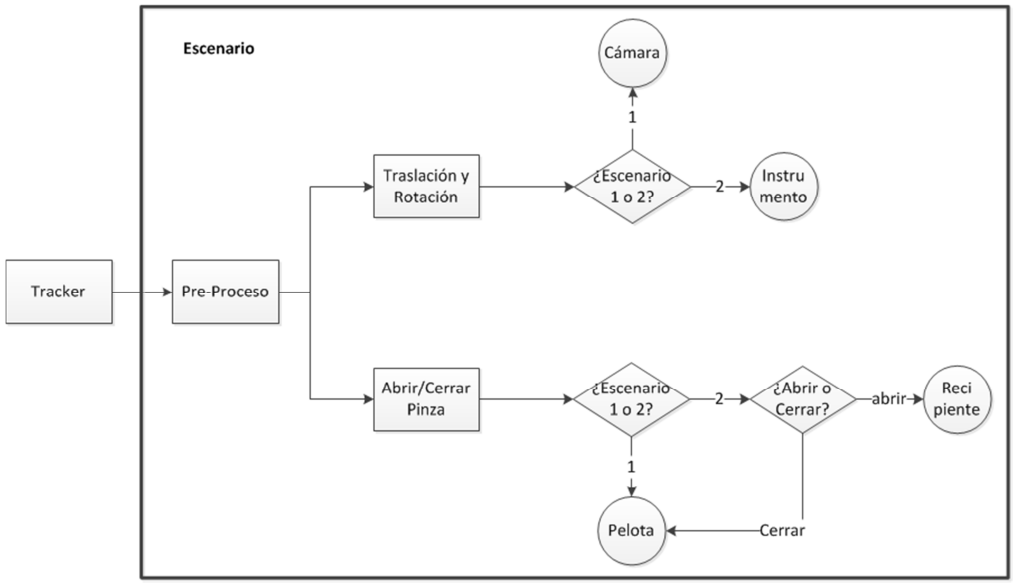

Figura 2.10: Diagrama de interacción de los escenarios.

Los datos resultantes del pre-proceso son transformados en una traslación y una rotación de *OpenGL*. Estas operaciones no afectan igual en ambos escenarios: en el primero mueven-giran la cámara (que está situada en el propio instrumento), mientras que en el segundo mueven-giran sólo el instrumento (la cámara está fija).

En cuanto al estado de la pinza, si éste es distinto del anterior estado significa que se produjo un movimiento de pinza. En la **Figura 2.10** se muestra cómo afecta este movimiento de forma distinta en cada escenario, puesto que en el primero la interacción se produce entre la cámara y la pelota, mientras que en el segundo se produce entre el instrumento, la pelota (para coger) y el recipiente (para dejar).

2.5. Generador de Movimientos

Este proceso se encarga de comprobar periódicamente los datos de posición, orientación y estado de pinza proporcionados por el tracker para detectar si se produjo algún movimiento. Además se encarga de traducir estos movimientos a caracteres, para que puedan ser utilizados en el cálculo de distancias.

Inicialmente, esta tarea era llevada a cabo en el propio módulo de tracking, pero luego se vio que era mejor separar la detección de la marca de la generación de movimientos, principalmente, porque un movimiento-carácter requiere de un mayor número de cambios de estado que un movimiento en el escenario (que es más directo). Además, permite en un futuro usar otro módulo de tracking (por ejemplo mecánico) sin que afecte al resto del proyecto.

2.5.1. Movimientos Representados

Antes de explicar cómo se genera un movimiento hace falta definir cuáles son los movimientos que van a ser representados. Dado que hay cinco grados de libertad, parece obvio que se va a tener que representar por lo menos diez movimientos:

- **En X:** hacia la izquierda (L) y hacia la derecha (R).
- **En Y:** hacia arriba (U) y hacia abajo (D).
- **En Z:** hacia delante (C) y hacia detrás (F).
- **De orientación:** giro hacia la izquierda (H) y hacia la derecha (A).
- **De cambio de estado de la pinza:** pasa a estar abierta (Q) y a estar cerrada (P).

Como ya se ha comentado la sección **2.4**, no es lo mismo en un ejercicio laparoscópico mover transportando un objeto en la pinza (está cerrada), que hacerlo sin objeto (pinza abierta). Por eso se pensó que en la posterior etapa de similitud entre cadenas de movimientos se debía penalizar no estar realizando los movimientos con la pinza en el mismo estado que el experto.

Usando sólo los 10 movimientos anteriores se tendría que acudir al último carácter de cambio de estado de la pinza en la cadena para saber si un determinado movimiento se realiza con la pinza abierta o cerrada. Esto supondría un trabajo adicional (previsiblemente costoso) y chocaba con la filosofía de los algoritmos de cálculo de similitud. Se descartó también acompañar cada movimiento explícitamente con el estado de la pinza en ese instante (p.e.: mover hacia la derecha con la pinza abierta sería "*QR*") porque duplicaría el tamaño de las cadenas de movimientos, lo cual nunca es deseable.

La solución adoptada es representar con un carácter diferente los movimientos con la pinza abierta y los movimientos con la pinza cerrada (**Tabla 2.1**). Por ejemplo, diferenciar entre un movimiento hacia la derecha con la pinza abierta (R) que con la pinza cerrada (S). De esta forma se duplica el número de caracteres pero no el tamaño de las cadenas.

	Pinza Abierta		**Pinza Cerrada**	
	-	+	-	+
X	L	R	K	S
Y	D	U	E	T
Z	C	F	J	I
Orientación	H	A	G	B
Estado Pinza	Q		P	

Tabla 2.1: Caracteres que representan los movimientos en los 5 grados de libertad según se esté pinzando en ese momento o no.

Para ver la importancia de diferenciar entre movimientos con la pinza abierta y cerrada veamos un ejemplo: un estudiante está en el segundo escenario transportando una pelota hacia su recipiente y a mitad camino despinza sin querer. A continuación puede ocurrir que:

- Trate de corregir el error (cierre la pinza), por lo que generaría la cadena "*QP*" (abre y cierra la pinza). El experto no ha realizado estos dos movimientos, por lo que el estudiante va a ser penalizado mínimamente, pero el resto de movimientos podrán ser correctos.

- No se dé cuenta y siga moviéndose, por ejemplo hacia la derecha, por lo que generaría la cadena: "*QSSS*" (abre la pinza y se mueve hacia la derecha). Esta cadena será comparada con la del experto: "*RRR*" (con la pinza cerrada se mueve hacia la derecha). Al no tener las cadenas nada en común la penalización va a ser considerablemente mayor que la anterior, mayor cuantos más movimientos realice sin corregir el error.

Otro aspecto importante a tener en cuenta para evaluar la destreza de un estudiante según sus movimientos es saber cómo y cuándo los realizó. No es lo mismo realizar unos movimientos durante un segundo, que hacerlos durante 5 segundos. Tampoco es lo mismo realizar una serie de movimientos de forma continua y fluida (sin pausas), que realizarlos de forma escalonada (parando cada dos por tres). Por esta razón se decidió incluir unos caracteres nuevos: los de temporización. No son movimientos per se, ya que el usuario no los realiza explícitamente y no son capturados por el tracker, pero sí son movimientos implícitos que se dan durante la realización de un ejercicio.

El primero de estos caracteres temporales es la *X*, que se genera obligatoriamente cada segundo. Sirve para conocer la velocidad con la que se realizan el resto de movimientos. Por ejemplo, no es lo mismo realizar tres movimientos hacia la derecha en menos de un segundo que tardar tres segundos. Ambos casos serían iguales ("*RRR*") sin este carácter, pero gracias a él se pueden distinguir, ya que en el primer caso se obtiene la cadena "*XRRRX*" mientras que en el segundo se genera la cadena "*XRXRXRX*".

El otro carácter temporal es la *N*. Esta letra se produce sólo cuando pasa un determinado tiempo (p.e.: medio segundo) sin haberse registrado algún movimiento. Sirve para saber cómo de fluido están siendo los movimientos. Una persona con experiencia moverá el instrumento de forma fluida, sin realizar casi pausas, mientras que una persona con poca experiencia irá poco a poco, realizando varias micro-pausas (imperceptibles al ojo).

Por ejemplo, el experto realiza tres movimientos hacia la derecha y genera la cadena "*RRR*". Sin embargo, un estudiante novato no realizará un movimiento tan marcado y perfecto. Sin el carácter *N* ambas cadenas serán iguales, pero con éste carácter no ocurrirá esto, ya que la cadena que se generará será como "*RNRNR*", que se asemeja a la del experto pero no es igual.

Además, variando el tiempo necesario para generar el carácter *N* se puede controlar la importancia que se le está dando a la fluidez, ya que no es lo mismo generar esta letra cada medio segundo sin mover que hacerlo cada 0.1 segundos (en este último caso cobra más importancia).

2.5.2. Cómo se Generan los Movimientos

El tracker está proporcionando datos varias veces por segundo (cada periodo de tracking). No se puede generar un carácter cada vez que uno de los cinco grados de libertad cambia su estado respecto del anterior periodo (salvo en el caso de la pinza) porque se estarían generando cadenas de gran tamaño, sino que hay que esperar a que se dé un cambio suficientemente grande para considerarlo movimiento. Estos cambios no se suelen dar de un periodo de tracking al siguiente (salvo en movimientos bruscos), por lo que se ha de llevar la cuenta de los cambios que se van produciendo en cada periodo. Además, hay que intentar evitar que la imprecisión del tracker llegue a la generación de caracteres, por lo que se ha de filtrar los datos que llegan a esta etapa.

Por ejemplo, se decide que en el eje X se generará un carácter cada 10 unidades de movimiento. Un estudiante empieza a mover y el tracker proporciona los siguientes valores de X (en cada periodo): 5, 8 y 11. Una primera solución sería esperar a que se sobrepase el umbral de 10, en el tercer periodo, para generar el carácter correspondiente. Sin embargo, podría ocurrir que un estudiante estuviese moviendo de derecha a izquierda el instrumento sin que ninguno de estos desplazamientos fuese superior a 10 unidades, por lo que no generaría ningún carácter.

Para que esto no ocurra se pueden ir acumulando los desplazamientos para que varios movimientos pequeños conformen uno grande. En el ejemplo anterior, en el primer periodo se acumularían 5 unidades, en el segundo 3 unidades, y en el tercero otras 3 unidades. Sería en este último periodo cuando el acumulador sobrepasase el umbral y se generase el carácter.

Para X harán falta dos acumuladores (**Figura 2.11**): para movimientos hacia la derecha y para movimientos hacia la izquierda. Otros dos acumuladores para Y, para Z y para la orientación. En total 8 acumuladores.

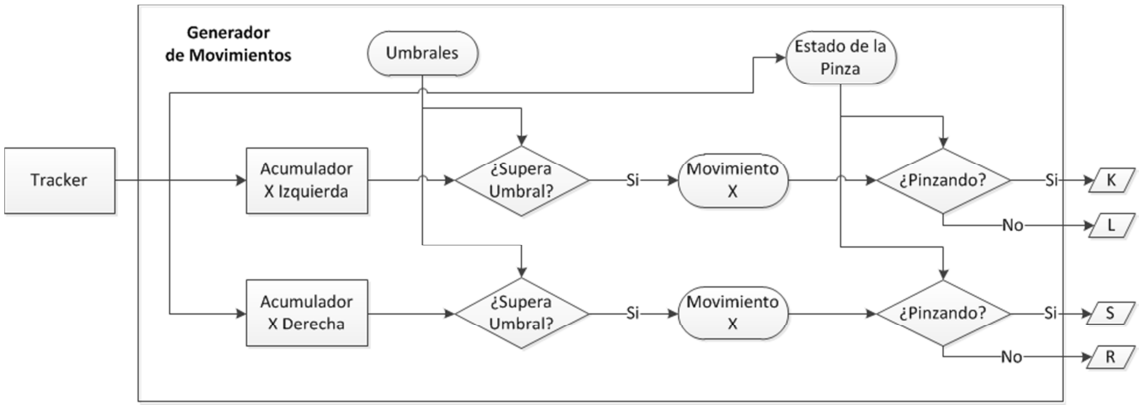

Figura 2.11: Representación de cómo se generan los movimientos de X en el Generador de Movimientos. De forma homóloga se haría para Y, Z y la orientación.

2.5.3. Desarrollo del Generador de Movimientos

El generador de movimientos está compuesto por dos procesos (*threads* de *C++*):

- El primero se encarga de recibir los datos del tracker, calcular el movimiento relativo que hubo entre ese periodo y el anterior y añadirlo al acumulador que corresponda.
- El segundo se encarga de comprobar si alguno de estos acumuladores ha superado los umbrales establecidos para generar el carácter correspondiente.

Para movimientos de la pinza basta con comprobar si el estado actual de la pinza difiere del anterior. No sólo se generará el movimiento de pinza correspondiente (*P* o *Q*), sino que además el estado actual se tendrá en cuenta a la hora de generar el resto de movimientos (sección **2.5.1**). En cuanto a los caracteres temporales, basta con contar el número de periodos del segundo proceso que pasan (**Figura 2.12**):

- Desde la última *X* para generar este carácter.
- Desde el último movimiento no temporal para generar la *N*.

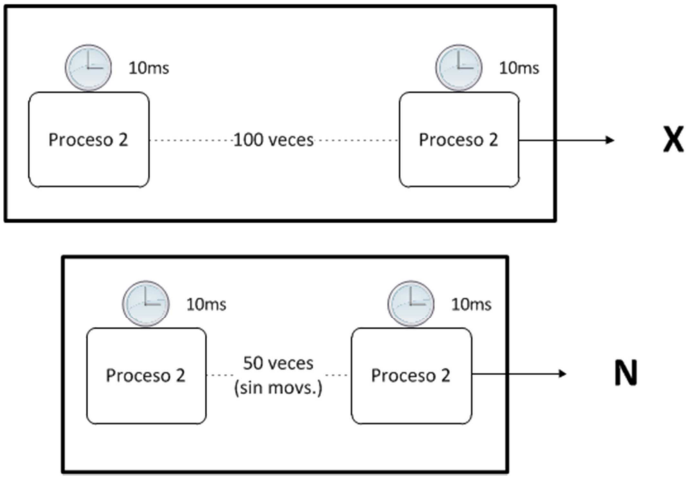

Figura 2.12: Obtención de los caracteres temporales mediante la cuenta de los periodos de muestreo que trascurren.

2.5.4. Los Umbrales de Movimiento

Los umbrales de movimiento son más importantes de lo que parecen. Al controlar cómo de grandes han de ser los movimientos para generar el carácter correspondiente, también se está controlando cómo de grandes van a ser las cadenas y la precisión con la que estas cadenas van a representar los movimientos reales.

Por ejemplo, un umbral de 1 unidad para X provoca que un movimiento de 10 unidades (muy pequeño) genere 10 caracteres. Esto implica dos cosas: se está representando el movimiento con una gran precisión, pero también se está generando una cadena de gran tamaño (con estos umbrales en algunos escenarios se generarían 10000 caracteres). No se puede alcanzar una gran precisión sin generar cadenas de gran longitud, ni generar pocos caracteres sin perder precisión.

Además, tener demasiada precisión transfiere demasiado bien las pequeñas imprecisiones cometidas en dos movimientos casi idénticos (p.e.: producidas por el pulso). Con un sistema demasiado preciso es más difícil encontrar dos movimientos iguales.

Por esta razón, para obtener los umbrales definitivos se realizaron varios experimentos. Uno de ellos trataba de homogeneizar los umbrales, ya que el tracker no es igual de sensible en todos los grados de libertad (p.e.: un movimiento de igual magnitud generaba más caracteres en Z que en X). Como resultado de este experimento un mismo movimiento generará el mismo número de caracteres en Z que en X, ya que sus umbrales serán equivalentes.

Otro experimento trataba de encontrar un umbral a mitad camino entre una fiel representación de los movimientos (sin llegar a la sobre-precisión) y una longitud de las cadenas adecuada. Para ello se obtenía la similitud entre varias cadenas con diferentes umbrales y se representaba con gráficas similitud-umbral. De esta forma es más fácil seleccionar el umbral más adecuado.

2.6. Distancias Entre Cadenas

Todos los algoritmos de cálculo de similitud que se han visto en la sección **1.3** tienen una cosa en común: requieren de las cadenas completas de antemano para calcular la distancia entre ambas. Esto quiere decir que los algoritmos, tal cual, sólo sirven en este proyecto para evaluar la destreza tras realizar el ejercicio (*offline*). En nuestro caso, es necesario que el cálculo de la similitud se produzca cada vez que el estudiante genera un movimiento, para poder proporcionarle una evaluación continua durante el ejercicio.

En el estudio previo no se pudo encontrar ningún algoritmo que trabajase carácter a carácter. Esto puede ser debido a que en las principales aplicaciones de estos métodos (p.e.: obtención de similitud entre secuencias RNA) ambas cadenas están completas de antemano. Por lo tanto, se necesita desarrollar un nuevo algoritmo, o adaptar uno de los ya existentes, para que trabaje con un flujo de caracteres (*online*) en lugar de con cadenas completas.

2.6.1. Cálculo de la Distancia Online

2.6.1.1. Elección del Algoritmo Base

Existen varias métricas para el cálculo de la distancia de edición (sección **1.3.3**). Para poder desarrollar un algoritmo de cálculo de similitud online hay que seleccionar una de estas métricas, ya que no todas ellas cuentan con las mismas operaciones ni representan lo mismo.

Se descartó Damerau-Levenshtein porque al contemplar cuatro operaciones elementales (inserción, borrado, sustitución y trasposición) podría resultar demasiado complejo de pasar a una versión online. Se descartó Hamming por todo lo contrario, ya que sólo contempla la operación de sustitución. Por lo tanto, hay que decidir entre usar Levenshtein o Longest Common Subsequence.

Tras evaluar los algoritmos base de ambas métricas se vio que LCS podría ser más adecuado para un trabajo carácter a carácter y más sencillo de adaptar, puesto que contempla sólo dos operaciones elementales (inserción y borrado) frente a las tres de Levenshtein (inserción, borrado y sustitución). Además, la operación de sustitución no es más que una inserción y un borrado, por lo que LCS podría verse como la generalización de Levenshtein.

La mayoría de los algoritmos de LCS vistos en sección **1.3.4** tratan de mejorar el coste temporal o de memoria para obtener la matriz de distancias, mientras que otros se centran en agilizar la fase de *back-trace*. Sin embargo, en el algoritmo online de este proyecto ninguno de estos aspectos es demasiado importante debido a que:

- El tiempo que se tarda en generar un carácter (usuario -> tracker -> generador) es mayor del que se tarda en calcular su distancia y obtener la cadena LCS.
- Las cadenas de movimientos tienen un tamaño relativamente pequeño, por lo que ni el coste en memoria ni el cálculo del back-trace van a ser problemáticos.

Además, alguno de estos algoritmos optimizados de LCS son realmente complejos, por lo que adaptarlos a una versión online resultaría una tarea difícil y en muchos casos imposible.

Por todas estas razones, se decidió utilizar el algoritmo más básico de LCS, el de Wagner-Fischer (sección **1.3.4.1**), como punto de partida para desarrollar el algoritmo online. Se ha llegado incluso a aplicar una pequeña optimización, que es trabajar con dos vectores en lugar de con una matriz (Hirschberg, sección **1.3.4.2**).

2.6.1.2. Longest Common Subsequence Online

En el algoritmo de Wagner-Fischer obtiene la distancia LCS entre dos cadenas mediante el uso de una matriz de programación dinámica. Cada celda $R[i][j]$ de esta matriz contiene la distancia entre las sub-cadenas $X[1..i]$ e $Y[1..j]$ y se calcula con la siguiente ecuación:

$$R[i][j] = \begin{cases} 0 & , \quad si\ i = 0\ o\ j = 0 \\ R[i-1][j-1] + 1 & , \quad si\ x_i = y_j \\ \max(R[i-1][j], R[i][j-1]), & si\ x_i \neq y_j \end{cases} \qquad \textbf{(2.1)}$$

Cuando finaliza el algoritmo (**Figura 2.13**) la distancia entre ambas cadenas es el valor de la última celda, $R[m][n]$ (siendo m el tamaño de X y n el tamaño de Y), ya que ésta contiene la distancia entre las sub-cadenas $X[1..m]$ e $Y[1..n]$, que son las cadenas completas.

Por ejemplo, un estudiante realiza un ejercicio y obtiene la cadena de movimientos "*RUURUR*". Para ver la destreza que demostró se compara esta cadena con la del experto, que es "*RUCURR*". La distancia LCS resultante es 5, como puede verse en la **Tabla 2.2**. Es decir, 5 de 6 caracteres del estudiante no requirieron operaciones de inserción o borrado, por lo que estos caracteres ("*RUURR*") son comunes entre ambas cadenas. Se podría decir que la cadena del estudiante es un 83'3% (5/6) similar a la del experto, ya que la distancia obtenida es 5 y la máxima distancia que se podría dar es el tamaño de la cadena más pequeña, que es 6. Si ambas cadenas fuesen iguales la similitud sería de 100% (6/6), mientras que si no tuviesen nada en común sería de 0% (0/6).

```
Desde i=0 hasta m
        Desde j=0 hasta n
                Si X[i] = Y[j] entonces
                        R[i][j] := R[i-1][j-1] + 1
                Sino
                        R[i][j] := max(R[i][j-1],R[i-1][j])

Devolver R[m][n]
```

Figura 2.13: Pseudocódigo del algoritmo LCS de Wagner-Fischer.

		R	**U**	**C**	**U**	**R**	**R**
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2
U	0	1	2	2	3	3	3
R	0	1	2	2	3	4	4
U	0	1	2	2	3	4	4
R	0	1	2	2	3	4	5

Tabla 2.2: Matriz de distancias LCS resultante de la comparación entre las cadenas "*RUURUR*" (estudiante) y "*RUCURR*" (experto).

Sin embargo, este algoritmo no nos vale tal cual, ya que en el de LCS online no se dispone de la cadena completa del estudiante de antemano (el primer bucle *Desde* en la **Figura 2.13**), sino que ésta se va generando poco a poco con los caracteres proporcionados por el generador de movimientos.

Siguiendo con el ejemplo anterior, el primer carácter generado sería *R*, por lo que la cadena de movimientos hasta ese momento sería "*R*". Con el segundo carácter la cadena sería "*RU*". Lo mismo pasaría con los siguientes movimientos: "*RUU*", "*RUUR*", "*RUURU*" y "*RUURUR*". Una primera aproximación a la obtención de la distancia online es calcular la distancia LCS de cada sub-secuencia del estudiante (obtenida tras cada nuevo movimiento) con la cadena completa del experto. En este ejemplo se calcularía la distancia LCS seis veces, obteniendo 5 tablas intermedias (**Tabla 2.3**, **Tabla 2.4**, **Tabla 2.5**, **Tabla 2.6** y **Tabla 2.7**) y la tabla LCS final (**Tabla 2.2**).

		R	**U**	**C**	**U**	**R**	**R**
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1

Tabla 2.3: Matriz de distancias LCS resultante de la comparación entre las cadenas "*R*" (sub-cadena del estudiante tras el primer movimiento) y "*RUCURR*" (experto).

		R	**U**	**C**	**U**	**R**	**R**
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2

Tabla 2.4: Matriz de distancias LCS resultante de la comparación entre las cadenas "*RU*" (sub-cadena del estudiante tras el segundo movimiento) y "*RUCURR*" (experto).

		R	**U**	**C**	**U**	**R**	**R**
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2
U	0	1	2	2	3	3	3

Tabla 2.5: Matriz de distancias LCS resultante de la comparación entre las cadenas "*RUU*" (sub-cadena del estudiante tras el tercer movimiento) y "*RUCURR*" (experto).

		R	U	C	U	R	R
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2
U	0	1	2	2	3	3	3
R	0	1	2	2	3	4	4

Tabla 2.6: Matriz de distancias LCS resultante de la comparación entre las cadenas "*RUUR*" (sub-cadena del estudiante tras el cuarto movimiento) y "*RUCURR*" (experto).

		R	U	C	U	R	R
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2
U	0	1	2	2	3	3	3
R	0	1	2	2	3	4	4
U	0	1	2	2	3	4	4

Tabla 2.7: Matriz de distancias LCS resultante de la comparación entre las cadenas "*RUURU*" (sub-cadena del estudiante tras el quinto movimiento) y "*RUCURR*" (experto).

Las similitudes obtenidas, mirando la última celda de cada tabla, son: 16'6% (1/6), 33'3% (2/6), 50% (3/6), 66'6% (4/6), 66'6% (4/6) y 83'3% (5/6). Estas similitudes no parecen muy significativas de la destreza del estudiante puesto que lo único que han hecho es crecer hasta alcanzar la distancia LCS tradicional. Esto es debido a que se ha comparando cada sub-cadena del estudiante con la cadena completa del experto, es decir, se ha evaluado al estudiante con movimientos que aun no ha tenido la oportunidad de realizar.

Por ejemplo, cuando el estudiante lleva dos movimientos ("*RU*") se compara con toda la cadena del experto ("*RUCURR*"). La similitud en este caso será como mucho de 33'3% (2/6), puesto que el estudiante aun no ha introducido más caracteres. Sin embargo, lo que lleva movido el estudiante es idéntico al experto ("*RU*"), es decir, lo está haciendo perfecto (similitud 100%).

Por esta razón se pensó que, a la hora de obtener la similitud, sólo se tendría que tener en cuenta del experto tantos caracteres como el usuario hubiese introducido hasta ese momento para ser justos en su evaluación (no estar evaluando con caracteres que aun no ha podido generar). Es decir, las matrices de distancia serán siempre cuadradas, puesto que habrá el mismo número de caracteres del estudiante que del experto. Se comparan sub-cadenas del estudiante con sub-cadenas del experto.

Siguiendo con el ejemplo anterior, con las 5 tablas intermedias (**Tabla 2.8**, **Tabla 2.9**, **Tabla 2.10**, **Tabla 2.11** y **Tabla 2.12**) y la tabla LCS final (**Tabla 2.2**) se obtendrían las siguientes similitudes: 100% (1/1), 100% (2/2), 66'6% (2/3), 75% (3/4), 80% (4/5) y 83'3% (5/6). Estas similitudes sí que dan una mejor idea de cómo lo está haciendo el estudiante.

		R
	0	0
R	0	1

Tabla 2.8: Matriz de distancias LCS resultante de la comparación entre las cadenas "*R*" (sub-cadena del estudiante tras el primer movimiento) y "*R*" (sub-cadena del experto tras el primer movimiento).

		R	**U**
	0	0	0
R	0	1	1
U	0	1	2

Tabla 2.9: Matriz de distancias LCS resultante de la comparación entre las cadenas "*RU*" (sub-cadena del estudiante tras el segundo movimiento) y "*RU*" (sub-cadena del experto tras el segundo movimiento).

		R	**U**	**C**
	0	0	0	0
R	0	1	1	1
U	0	1	2	2
U	0	1	2	2

Tabla 2.10: Matriz de distancias LCS resultante de la comparación entre las cadenas "*RUU*" (sub-cadena del estudiante tras el tercer movimiento) y "*RUC*" (sub-cadena del experto tras el tercer movimiento).

		R	**U**	**C**	**U**
	0	0	0	0	0
R	0	1	1	1	1
U	0	1	2	2	2
U	0	1	2	2	3
R	0	1	2	2	3

Tabla 2.11: Matriz de distancias LCS resultante de la comparación entre las cadenas "*RUUR*" (sub-cadena del estudiante tras el cuarto movimiento) y "*RUCU*" (sub-cadena del experto tras el cuarto movimiento).

		R	U	C	U	R
	0	0	0	0	0	0
R	0	1	1	1	1	1
U	0	1	2	2	2	2
U	0	1	2	2	3	3
R	0	1	2	2	3	4
U	0	1	2	2	3	4

Tabla 2.12: Matriz de distancias LCS resultante de la comparación entre las cadenas "*RUURU*" (sub-cadena del estudiante tras el quinto movimiento) y "*RUCUR*" (sub-cadena del experto tras el quinto movimiento).

El algoritmo anterior resuelve el problema de la obtención de la distancia de forma interactiva mediante de forma "tosca", ya que simplemente se está calculando la distancia LCS tradicional tantas veces como movimientos. Teniendo en cuenta que en los escenarios pueden generarse más de 100 caracteres, se generarían más de 100 tablas, algunas de ellas con un tamaño superior a 100x100.

El algoritmo desarrollado (**Figura 2.14**) se utiliza una única tabla durante todo el proceso, proporcionando así una solución óptima al problema del cálculo de distancias online. Su principal característica es que la matriz va aumentando el número de filas conforme se reciben caracteres del estudiante. Su funcionamiento es el siguiente:

1. La matriz está inicialmente vacía debido a que no ha llegado aún ningún carácter del estudiante (la fila y columna del carácter vacío son obviadas en esta explicación).

2. Llega el primer carácter, por lo que se crea una fila para él y se calcula la distancia de este carácter con todos los del experto. La similitud que se devuelve en esta etapa corresponde a la última celda de la matriz 1x1, ya que llevamos un único carácter (para evitar el problema comentado anteriormente).

3. Llega el segundo carácter, por lo que se crea una nueva fila para él y se calcula la distancia de este carácter con todos los del experto. La distancia que se devuelve es la de la última celda de la matriz 2x2

4. Se procede de igual forma con el resto de caracteres hasta llegar a la matriz 6x6, que es la misma que la obtenida por el algoritmo LCS tradicional.

A continuación pueden verse los diferentes estados que atraviesa la matriz de distancias durante el cálculo de similitudes online con las cadenas del ejemplo anterior. En rojo se muestra la distancia que se utiliza para la obtención de la similitud en cada etapa, que es la última celda de la matriz cuadrada (en azul claro) correspondiente al número de caracteres que el estudiante lleva.

```
i := 0

Mientras haya movimientos del estudiante & i<n

        usuario[i] := nuevo movimiento

        Desde j=0 hasta n

                Si usuario[i] = experto[j] entonces

                        R[i][j] := R[i-1][j-1] + 1

                Sino

                        R[i][j] := max(R[i][j-1],R[i-1][j])

        similitud[i] := R[i][i];

        i++
```

Figura 2.14: Pseudocódigo del algoritmo LCS online desarrollado.

		R	**U**	**C**	**U**	**R**	**R**
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?
¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?
¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?
¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?
¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?

Tabla 2.13: Estado de la matriz de distancias LCS tras el primer carácter del estudiante.

		R	**U**	**C**	**U**	**R**	**R**
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2
U	¿?	¿?	¿?	¿?	¿?	¿?	¿?
R	¿?	¿?	¿?	¿?	¿?	¿?	¿?
U	¿?	¿?	¿?	¿?	¿?	¿?	¿?
R	¿?	¿?	¿?	¿?	¿?	¿?	¿?

Tabla 2.14: Estado de la matriz de distancias LCS tras el segundo carácter del estudiante.

		R	U	C	U	R	R
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2
U	0	1	2	2	3	3	3
R	¿?	¿?	¿?	¿?	¿?	¿?	¿?
U	¿?	¿?	¿?	¿?	¿?	¿?	¿?
R	¿?	¿?	¿?	¿?	¿?	¿?	¿?

Tabla 2.15: Estado de la matriz de distancias LCS tras el tercer carácter del estudiante.

		R	U	C	U	R	R
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2
U	0	1	2	2	3	3	3
R	0	1	2	2	3	4	4
U	¿?	¿?	¿?	¿?	¿?	¿?	¿?
R	¿?	¿?	¿?	¿?	¿?	¿?	¿?

Tabla 2.16: Estado de la matriz de distancias LCS tras el cuarto carácter del estudiante.

		R	U	C	U	R	R
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2
U	0	1	2	2	3	3	3
R	0	1	2	2	3	4	4
U	0	1	2	2	3	4	4
R	¿?	¿?	¿?	¿?	¿?	¿?	¿?

Tabla 2.17: Estado de la matriz de distancias LCS tras el quinto carácter del estudiante.

Como se puede observar en las tablas anteriores (**Tabla 2.13**, **Tabla 2.14**, **Tabla 2.15**, **Tabla 2.16** y **Tabla 2.17**), la matriz en realidad no es dinámica, sino que es cuadrada (del tamaño de la cadena del experto). De esta forma, en lugar de insertar filas, lo que se hace es rellenar filas vacías, que es más cómodo y tiene menor coste computacional. Sin embargo, esto tiene un punto negativo, y es que se ha de tener en cuenta dos posibles casos: cuando el estudiante introduce menos caracteres que el experto (se quedan filas sin rellenar), y cuando introduce más caracteres (faltan columnas).

Para solucionar el primer caso, en el que el estudiante se ha quedado corto de movimientos, lo que se hace es seguir proporcionando similitudes con los caracteres restantes del experto. Estas similitudes corresponderán a fallos, puesto que el estudiante no ha introducido más caracteres (sería como comparar caracteres vacíos del estudiante con los restantes del experto). Por ejemplo, si en lugar de tener la cadena del experto "*RUCURR*" se tuviese "*RUCURRAA*", el estado de la tabla tras introducir el estudiante su sexto carácter sería:

		R	**U**	**C**	**U**	**R**	**R**	**A**	**A**
	0	0	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1	1	1
U	0	1	2	2	2	2	2	2	2
U	0	1	2	2	3	3	3	3	3
R	0	1	2	2	3	4	4	4	4
U	0	1	2	2	3	4	4	4	4
R	0	1	2	2	3	4	5	5	5

Tabla 2.18: Estado de la matriz de distancias LCS tras el sexto carácter del estudiante cuando la cadena del experto es "*RUCURRAA*".

Como puede observarse en la **Tabla 2.18** sobran dos caracteres del experto (o le faltan dos caracteres al estudiante). La solución adoptada es proporcionar tras la última similitud (83'3% o 5/6) dos nuevas similitudes correspondientes a los caracteres *A* del experto: 71'43% (5/7) y 62'5% (5/8).

Para el segundo caso, en el que el estudiante realizó movimientos de más, se procede de forma parecida. Los caracteres adicionales del estudiante son tratados como errores ya que se comparan con caracteres vacíos del experto. Por ejemplo, si el estudiante introduce la cadena "*RUURURAA*" en lugar de la original "*RUURUR*", la tabla que se obtendría tras introducir el sexto carácter sería:

		R	**U**	**C**	**U**	**R**	**R**
	0	0	0	0	0	0	0
R	0	1	1	1	1	1	1
U	0	1	2	2	2	2	2
U	0	1	2	2	3	3	3
R	0	1	2	2	3	4	4
U	0	1	2	2	3	4	4
R	0	1	2	2	3	4	5
A	¿?	¿?	¿?	¿?	¿?	¿?	¿?
A	¿?	¿?	¿?	¿?	¿?	¿?	¿?

Tabla 2.19: Estado de la matriz de distancias LCS tras el sexto carácter del estudiante cuando la cadena de este es "*RUURURAA*".

Las similitudes obtenidas serían las mismas que cuando el estudiante terminó antes que el experto. De esta forma, en ambos casos estamos penalizando no utilizar el mismo número de movimientos que el experto, ya sea por exceso o por defecto. Una alternativa menos severa sería tener en cuenta estos movimientos adicionales, pero esto implicaría cambiar bastante código por lo que se deja como una posible mejora futura (sección **4**).

2.6.2. Feedback al Usuario

El objetivo de un sistema de evaluación de destrezas online es proporcionar al usuario en todo momento durante la realización del ejercicio una medida de lo bien o mal que lo está haciendo. Es esto lo que lo diferencia de los sistemas de evaluación existentes. Con el algoritmo LCS online mostrado anteriormente (**Figura 2.14**) se obtiene una medida de la similitud tras cada movimiento. Esta forma de retroalimentación o *feedback* es suficiente, puesto que permite al estudiante conocer en todo momento cómo de bien lo esta haciendo mediante una puntuación de 0 a 100%. Sin embargo, podría ocurrir que a un usuario avanzado (un estudiante con experiencia o el profesor que supervisa el ejercicio) esta puntuación le pareciese insuficiente y quisiese más información para hacerse una idea del porqué de su puntuación.

Como ya se vio en el estado del arte, recorriendo la matriz de distancias se puede obtener la cadena LCS, que contiene los movimientos comunes de experto y estudiante. A este proceso se le llama *back-trace* y es lo que se usa en este proyecto como método avanzado de feedback. Así el usuario avanzado puede conocer también qué movimientos realizó correctamente.

Para lograr esto se ha modificado el algoritmo de back-trace de Wagner-Fischer (sección **1.3.4.1**) para que permita obtener la cadena LCS partiendo desde cualquier posición de la matriz, en lugar de hacerlo desde la última celda, para poder usarlo en cada etapa del algoritmo online.

Sin embargo, esta forma de feedback en el prototipo final se ha dejado como opcional, para que los usuarios avanzados la activen si desean, mientras que al usuario normal sólo se le muestra la similitud. De esta forma se evita abrumar al usuario novato o aquel que sólo desea conocer cómo de bien lo está haciendo, con cadenas de movimientos comunes que terminarán siendo bastante grandes (si lo está haciendo bien).

2.6.3. Pre-Proceso de las Cadenas

Inicialmente, todas las cadenas del estudiante comenzaban por una secuencia de caracteres temporales, de la forma "*XNNXNNX...*", hasta que finalmente se producía un movimiento no-temporal. Esto es debido a que desde que se empieza a trackear y generar movimientos hasta que el estudiante realiza su primer movimiento con el instrumento pasan unos segundos.

Esta información resulta perjudicial a la hora de calcular las distancias, puesto que se está evaluando no sólo los movimientos realizados sino cuándo se comenzó a mover. De forma que si, por ejemplo, el experto empieza a mover antes que el estudiante, cuando este último comience el ejercicio lo hará con una penalización de partida (aunque luego lo haga perfecto no obtendrá 100%). Para evitar esto, basta con descartar en el cálculo de distancias los caracteres iniciales de tiempo, para que la primera corresponda a los primeros caracteres no-temporales de ambos.

También se descartan los movimientos de giro cuando se está realizando el escenario 2, ya que en este ejercicio no se tiene en cuenta la orientación (el estudiante no tiene referencia de qué giros ha de realizar), por lo que si luego ésta se tiene en cuenta en el cálculo de similitud podría llegar a confundir al usuario (no relacionaría malas similitudes con fallos de giro) y tampoco sería justo (no se quiere evaluar ese movimiento).

El último pre-procesamiento realizado tiene que ver con el carácter temporal *X*. En la sección del **2.5.1** se comentó que se introducía este carácter para dar una medida de la velocidad a la que se realizan los movimientos. Por ejemplo: un estudiante que genera "*XLLX*" está moviendo a 2 caracteres por segundo. Si el experto generó "*XLXL*" (1 carácter por segundo) entonces el estudiante va demasiado rápido.

Lo que ocurre es que cuando estos caracteres llegan al cálculo de distancias lo que hacen es difuminar el resto de movimientos, ya que al generarse cada segundo son bastante comunes en las cadenas y, por esta misma razón, producen muchos matches. Por ejemplo, el experto genera "*XLLLX*" y el estudiante "*XRRRX*". Lo lógico sería pensar que el estudiante ha de obtener una similitud del 0%, ya que ha errado en todos sus movimientos. Sin embargo, las similitudes que va a obtener, como se puede ver en la **Tabla 2.20**, le hacen pensar que no lo está haciendo tan mal, ya que obtendría: 100% (1/1), 50% (1/2), 33'3% (1/3), 25% (1/4), y 40% (2/5).

Para evitar esto basta con ignorar las *X* en el cálculo de la distancia. Esto no quiere decir que no se vayan a utilizar, ya que en la sección **2.6.5** se verá cómo usar estos caracteres para su propósito inicial, que es informar de la diferencia de velocidad entre el estudiante y el experto, sin que afecte al cálculo de la similitud.

		X	**L**	**L**	**L**	**X**
	0	0	0	0	0	0
X	0	1	1	1	1	1
R	0	1	1	1	1	1
R	0	1	1	1	1	1
R	0	1	1	1	1	1
X	0	1	1	1	1	2

Tabla 2.20: Matriz de distancias obtenida con las cadenas "*XLLLX*" y "*XRRRX*" en la que se da una mejor puntuación de la que debería.

2.6.4. Caracteres con Peso

En la sección **2.5.1** se comentó que se introducía el carácter *N* para representar cómo de fluido se producen los movimientos. Con este carácter se pretende diferenciar entre un estudiante que realiza unos movimientos de forma escalonada o con micro-pausas (p.e.: "*RNRNR*") de un experto que los realiza de forma continua (p.e.: "*RRR*"). Tras los primeros test con el algoritmo de LCS online se vio que cumplía esta función pero que tenía algunos aspectos negativos importantes.

Por ejemplo, en un ejercicio el experto obtiene la cadena "*LNR*" y el estudiante la cadena "*NLR*". La similitud obtenida al final es 66'6% (2/3), como puede verse en la **Tabla 2.21**. Esta puntuación podría parecer demasiado baja teniendo en cuenta que sin el carácter *N* (sin tener en cuenta la continuidad de los movimientos) ambas cadenas son iguales ("*LU*").

		L	**N**	**R**
	0	0	0	0
N	0	0	1	1
L	0	1	1	1
R	0	1	1	2

Tabla 2.21: Matriz de distancias LCS obtenida con las cadenas "*LNR*" y "*NLR*".

En este caso la penalización es debida a que la pausa se realizó en instantes de tiempo diferentes. El problema es que si el error fuese uno movimiento no-temporal incorrecto la similitud va a ser la misma. Por ejemplo, el experto obtiene "*LAR*" porque gira a la izquierda en lugar de realizar la pausa, y el estudiante "*HLR*" porque gira incorrectamente a la derecha. La similitud obtenida (**Tabla 2.22**) también es 66'6%, cuando el error parece más grave.

		L	**A**	**R**
	0	0	0	0
H	0	0	0	0
L	0	1	1	1
R	0	1	1	2

Tabla 2.22: Matriz de distancias LCS obtenida con las cadenas "*LAR*" y "*HLR*".

En otro ejercicio el experto obtiene la cadena "LNU", mientras que el usuario obtiene "*NLD*". Se ha cometido un fallo grave: se movió hacia abajo (*D*) en lugar de hacia arriba (*U*). La similitud (**Tabla 2.23**) que se obtiene es 33'3% (1/3). Esta puntuación podría parecer demasiado baja teniendo en cuenta que sin el carácter *N* la similitud sería 50% (1/2), dado que sólo tienen un carácter en común (la *L*).

		L	N	U
	0	0	0	0
N	0	0	1	1
L	0	1	1	1
D	0	1	1	1

Tabla 2.23: Matriz de distancias LCS obtenida con las cadenas "*LNU*" y "*NLD*".

En este caso ocurre igual que en el anterior, si el error en lugar de ser una pausa realizada en mal momento fuese un movimiento incorrecto se obtendría la misma similitud. Por ejemplo, si el experto obtiene la cadena "LAU" y el estudiante "HLD", el nuevo fallo es un movimiento hacia la derecha en lugar de hacia la izquierda. Como se puede observar en la **Tabla 2.24**, la similitud obtenida también es 33.3%, cuando la gravedad del error es mayor.

		L	A	U
	0	0	0	0
H	0	0	0	0
L	0	1	1	1
D	0	1	1	1

Tabla 2.24: Matriz de distancias LCS obtenida con las cadenas "*LAU*" y "*HLD*".

Con estos ejemplos se puede ver como el carácter *N* no sólo se está penalizando en exceso cuando una pausa no se realiza en el mismo instante que el experto, sino que también se le está dando la misma importancia a este tipo de fallo que a uno por movimiento (real) incorrecto. Para evitar esto se ha de quitar importancia (peso) a los fallos causados por este carácter.

2.6.4.1. ¿Cómo se resta importancia a la N?

La causa de la excesiva penalización del carácter *N* es que en la **Ecuación 2.1** todos los fallos afectan de la misma forma (no aumentando la distancia). Si hubiese un caso adicional que contemplase los fallos producidos por una *N*, que no penalizase tanto como el resto de fallos, se le estaría quitando peso a este tipo de fallos.

Para lograrlo se modifica esta ecuación para que a la hora de asignar la distancia en caso de no haber matching compruebe si es debido a una *N*, en cuyo caso se le sumara a la distancia obtenida un pequeño peso para no penalizar tanto este fallo. La ecuación quedaría así (en rojo las modificaciones):

$$R[i][j] = \begin{cases} 0 & , \quad si\ i = 0\ o\ j = 0 \\ R[i-1][j-1] + 1 & , \quad si\ x_i = y_j \\ \max(R[i-1][j], R[i][j-1]) + coste, & si\ x_i \neq y_j \end{cases} \tag{2.2}$$

$$coste = \begin{cases} peso & , \ si\ x_i\ o\ y_j\ es\ 'N' \\ 0 & , \ sino \end{cases}$$

Las modificaciones en el código del algoritmo LCS online son mínimas, como puede verse en la **Figura 2.15** en rojo. Sin embargo, tiene algunas implicaciones como que la matriz pasa a tener números decimales y a ser algo menos legible.

2.6.4.2. ¿Cuánta Importancia Restarle a la N?

El siguiente paso es seleccionar el peso a utilizar, que es el que indica cuánta importancia se le resta a los fallos producidos por el carácter *N*. Por ejemplo, un peso 1 implicaría que un fallo causado por este carácter se ignora completamente (es un match), mientras que un peso 0 es como si no existiese esta modificación de los pesos (se penalizaría igual que en el resto de fallos).

```
Desde i=0 hasta m
        Desde j=0 hasta n
                Si X[i] = Y[j] entonces
                        R[i][j] := R[i-1][j-1] + 1
                Sino
                        Si X[i]='N' o Y[j]='N' entonces
                                coste := peso
                        Sino
                                coste := 0
                        R[i][j] := max(R[i][j-1],R[i-1][j]) + coste

Devolver R[m][n]
```

Figura 2.15: Pseudocódigo del algoritmo LCS online con pesos.

Para determinar el peso se realizaron muchas pruebas. La primera para encontrar cuál es el límite superior del peso para que la similitud nunca supere el valor 100% (no tiene sentido). El límite inferior es 0, que es la distancia que se suma cuando hay un fallo en el algoritmo LCS tradicional.

Por ejemplo, en un ejercicio el experto mueve hacia la izquierda ("*XLLX*") y el estudiante no mueve ("*XNNX*"). Como referencia, las similitudes que sin pesos (o con peso 0) se obtendrían son (**Tabla 2.25**): 100% (1/1), 50% (1/2), 33'3% (1/3) y 50% (2/4).

		X	**L**	**L**	**X**
	0	0	0	0	0
X	0	1	1	1	1
N	0	1	1	1	1
N	0	1	1	1	1
X	0	1	1	1	2

Tabla 2.25: Matriz de distancias LCS obtenida con las cadenas "*XLLX*" y "*XNNX*" sin pesos.

Como puede verse en la **Tabla 2.26**, con un peso 1 se obtienen las siguientes similitudes durante el ejercicio: 100% (1/1), 150% (3/2), 166'6% (5/3) y 150% (6/4). Dado que varias son superiores a 100% se puede descartar este peso.

		X	**L**	**L**	**X**
	0	0	0	0	0
X	0	1	1	1	1
N	0	2	3	4	5
N	0	3	4	5	6
X	0	1	4	5	6

Tabla 2.26: Matriz de distancias LCS obtenida con las cadenas "*XLLX*" y "*XNNX*" con peso 1.

El siguiente peso a probar es 0'5, a medio camino entre un match y un fallo completo. Con el mismo ejemplo obtendríamos, como puede verse en la **Tabla 2.27**, las siguientes similitudes: 100% (1/1), 100% (2/2), 100% (3/3) y 100% (4/4). Si bien es verdad que no se ha superado la similitud de 100%, se ha obtenido una similitud máxima sin el estudiante haber realizado los movimientos hacia la izquierda. Es decir, tratando de no perjudicar con los fallos producidos por la letra *N* se han pasado por alto fallos más graves. Por lo tanto, este peso no es nada recomendable pero nos permite asegurar que cualquier peso inferior será válido (en % al menos).

		X	L	L	X
	0	0	0	0	0
X	0	1	1	1	1
N	0	1’5	2	2’5	3
N	0	2	2’5	3	3’5
X	0	1	2’5	3	4

Tabla 2.27: Matriz de distancias LCS obtenida con las cadenas “*XLLX*” y “*XNNX*” con peso 0.5.

Continuando con la búsqueda se prueba con 0’25. Las similitudes obtenidas en este caso, como puede observarse en la **Tabla 2.28**, son: 100% (1/1), 75% (1.5/2), 66’6% (2/3) y 75% (3/4). Estas similitudes ya son más parecidas a lo que se esperaba obtener con la modificación de los pesos. Por lo tanto, el rango de valores que puede tomar el peso va desde 0 hasta 0’25.

		X	L	L	X
	0	0	0	0	0
X	0	1	1	1	1
N	0	1’25	1’5	1’75	2
N	0	1’5	1’75	2	2’25
X	0	1	1’75	2	3

Tabla 2.28: Matriz de distancias LCS obtenida con las cadenas “*XLLX*” y “*XNNX*” con peso 0.25.

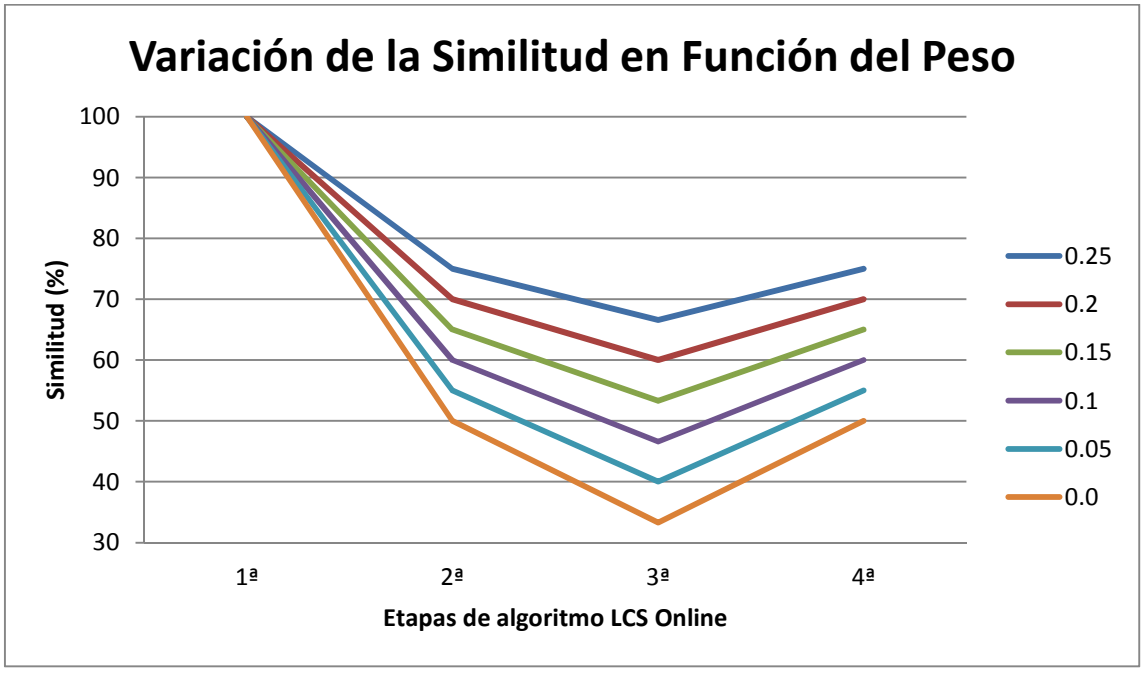

Figura 2.17: Gráfica que muestra la variación de la similitud LCS online según el peso utilizado con las cadenas “*XLLX*” y “*XNNX*”.

En la **Figura 2.17** puede verse como varían las similitudes obtenidas variando el peso entre una excesiva penalización (0) hasta una penalización casi inexistente (0'25). El peso a elegir dentro de este rango depende de la importancia que se quiera dar al tiempo en el ejercicio. A un usuario novel no se le puede pedir que realice los movimientos de forma continua como un experto, por lo que se podría relajar los requisitos temporales (peso cercano a 0'25). Mientras que a un estudiante experimentado sí se le puede exigir que realice las pausas y los movimientos igual que el experto (peso cercano a 0).

Por esta razón, se optó porque el peso fuese un parámetro de los ejercicios que se proporcionase al algoritmo LCS online, de forma que si se asigna un nivel bajo de exigencia temporal se estarán evaluando sobretodo los movimientos explícitos, mientras que si se asigna un nivel alto se evaluarán tanto los movimientos explícitos como los implícitos (pausas, fluidez,...).

2.6.5. Velocidad de Movimientos

En la sección **2.6.3** se explicó por qué se ignora el carácter *X* durante el cálculo de la distancia. Ahora bien, esto no quiere decir que no se vaya a usar este movimiento, ya que la información temporal que aporta (la velocidad a la que se están realizando los movimientos) puede resultar de gran utilidad para el estudiante durante el ejercicio.

Por ejemplo, un experto realiza un ejercicio en el que pinza, mueve hacia la izquierda y luego despinza, por lo que genera "*PXLLXLLXLLXQ*". Se podría decir que el experto mueve a una velocidad 2 caracteres por segundo (cps). Un estudiante realiza el mismo ejercicio y obtiene la cadena "*PXLLLXLXLXLQ*". Los movimientos realizados son los mismos ("*PLLLLLLQ*"), por lo que la similitud es del 100%. Sin embargo, la velocidad a la que se ha realizado el ejercicio no es la misma, ya que en algún momento ha ido a 3 cps y en otros 1 cps.

La velocidad de movimiento influye en las cadenas generadas: un estudiante que va lento genera más caracteres *N* que uno que va rápido (es más fácil que ocurran micro-pausas), mientras que uno que va demasiado rápido puede no generar suficientes *N* (no deja pasar tiempo suficiente entre movimientos). Si no se mostrase la diferencia de velocidad sería más fácil pasar por alto esto.

Para poder mostrar al usuario la velocidad a la que está realizando sus movimientos hace falta conocer la velocidad a la que se movía el experto, e ir obteniendo la velocidad a la que se mueve el estudiante. En ambos casos, para representar esta velocidad se utiliza un vector en el que la posición *i* contiene la velocidad a la que se mueve el estudiante/experto durante el segundo *i*. Por ejemplo, en la cadena anterior del experto, el vector obtenido sería $\{1, 2, 2, 2, 1\}$, mientras que en el caso del estudiante se obtendría al terminar el ejercicio $\{1, 3, 1, 1, 2\}$.

Gracias a estos vectores, se puede obtener la velocidad relativa mediante la comparación de cada posición del vector del estudiante con su homóloga del experto. Por ejemplo:

- **1er Segundo:** tanto el estudiante como el experto se mueven a 1 cps, por lo que van a la misma velocidad.

- **2º Segundo:** el estudiante se mueve a 3 cps mientras que el experto lo hace a 2 cps, por lo tanto va más rápido.

- **3er Segundo:** el estudiante se mueve a 1 cps mientras que el experto lo hace a 2 cps, por lo tanto va más lento.

- **4º Segundo:** el estudiante se mueve a 1 cps mientras que el experto lo hace a 2 cps, por lo tanto va más lento.

- **5º Segundo:** el estudiante se mueve a 2 cps mientras que el experto lo hace a 1 cps, por lo tanto va más rápido.

En el algoritmo de cálculo de similitud LCS online, con cada nuevo carácter que no sea *X* del estudiante no sólo se calcula la similitud, sino que además se actualiza el estado del vector de tiempos del estudiante en ese segundo. Si el carácter recibido es una *X* no se calcula similitud, pero si se compara los vectores de tiempos para obtener la velocidad relativa del estudiante.

2.6.6. Cálculo de Distancia Offline

Pese a que la parte más importante del proyecto es el cálculo de la distancia online, desde un principio se quiso proporcionar otras medidas de distancia al termino del ejercicio para que el estudiante dispusiese de más información a la hora de valorar cómo realizo el ejercicio. Esta evaluación de destreza a posteriori (offline) es la misma que realizan los sistemas comerciales vistos en la sección **1.2**, pero en este caso se sigue utilizando las distancias entre cadenas.

Se pensó en Damerau-Levenshtein para este propósito, ya que al disponer de dos operaciones adicionales podría corregir algunos errores típicos, que son excesivamente penalizados en LCS, como la trasposición entre dos caracteres. Por ejemplo, se realiza un movimiento en diagonal hacia arriba y hacia la izquierda: se puede obtener tanto "*LU*" como "*UL*" según qué umbral se alcanzó antes.

Pero dado que la distancia de Damerau-Levenshtein se obtiene añadiendo a la de Levenshtein las trasposiciones, se decidió incluir también este algoritmo y así proporcionar varias medidas. De esta forma la diferencia de similitud entre ambos es equivalente al número de errores de trasposición.

2.6.6.1. Similitud de Levenshtein y Damerau-Levenshtein

La similitud en estos dos algoritmos se obtiene de forma distinta que en LCS, ya que la distancia viene dada como el coste de las operaciones que hay que realizar sobre la cadena del estudiante para obtener la cadena del experto, mientras que en LCS la distancia es el número de caracteres que no requieren de operaciones. Por ejemplo, si el estudiante obtiene la cadena "*XLDPX*" y el experto "*XRDPX*" la distancia obtenida por LCS es 4 (**Tabla 2.29**), mientras que con Levenshtein se obtiene la distancia 1 (**Tabla 2.30**).

		X	**L**	**D**	**P**	**X**
	0	0	0	0	0	0
X	0	1	1	1	1	1
R	0	1	1	1	1	1
D	0	1	1	2	2	2
P	0	1	1	2	3	3
X	0	1	1	1	1	4

Tabla 2.29: Matriz de distancias LCS obtenida con las cadenas "*XLDPX*" y "*XRDPX*" (sin pesos).

		X	**L**	**D**	**P**	**X**
	0	1	2	3	4	5
X	1	0	1	2	3	4
R	2	1	1	2	3	4
D	3	2	2	1	2	3
P	4	3	3	2	1	2
X	5	4	4	3	2	1

Tabla 2.30: Matriz de distancias Levenshtein obtenida con las cadenas "*XLDPX*" y "*XRDPX*" (sin pesos).

Para entender cómo se obtiene la similitud de Levenshtein y de Damerau-Levenshtein hay que tener en cuenta los límites de las distancias que proporcionan: si ambas cadenas no tienen nada en común se tendrá que realizar tantas operaciones como el tamaño de la cadena más grande, mientras que si son idénticas la distancia es 0. Esto es casi inverso a la distancia LCS: si ambas cadenas no tienen nada en común la distancia es 0, mientras que si son idénticas la distancia es el tamaño de la más pequeña (por eso la relación no es totalmente inversa).

Teniendo en cuenta estos límites, la similitud de Levenshtein y Damerau-Levenshtein se obtiene mediante la siguiente ecuación (siendo m y n los tamaños de las cadenas):

$$Similitud = \left(1 - \frac{distancia}{\max(m,n)}\right) x100 \quad \textbf{(2.3)}$$

En el ejemplo anterior, en el caso de LCS se obtendría una similitud de 80% (4/5), que es la misma que con Levenshtein (1 − 1/5). Sin embargo, esta igualdad no siempre se cumplirá (como se verá luego).

2.6.6.2. Con Pesos

Es importante, para ser coherentes con los resultados de LCS online, que las similitudes proporcionadas por los algoritmos offline incluyan también los pesos o, de lo contrario, se podría confundir al estudiante mostrando medidas en las que el tiempo juega un papel diferente.

Para añadir pesos a estos algoritmos hay que fijarse primero en la ecuación de obtención de los valores de cada celda (**Ecuación 2.4**): cuando hay match se le asigna el menor valor de los vecinos (la distancia no aumenta), mientras que cuando hay fallo la distancia aumenta en una unidad respecto del menor de los vecinos.

$$R[i][j] = \begin{cases} \min(R[i-1][j]+1, R[i][j-1]+1, R[i-1][j-1]) & , \quad si\ x_i = y_j \\ \min(R[i-1][j]+1, R[i][j-1]+1, R[i-1][j-1]+1) & , \quad si\ x_i \neq y_j \end{cases} \quad \textbf{(2.4)}$$

Al contrario que en LCS con pesos, en este caso hay que impedir que la distancia aumente en una unidad cuando se produzca fallo por carácter *N*, ya que se penaliza de igual forma que por fallo no-temporal. Para evitar esto lo que se ha de hacer es aumentar la distancia menos de una unidad. La ecuación anterior quedaría así:

$$R[i][j] = \begin{cases} \min(R[i-1][j]+1, R[i][j-1]+1, R[i-1][j-1]) & , \quad si\ x_i = y_j \\ \min(R[i-1][j]+1, R[i][j-1]+1, R[i-1][j-1]+1) - coste & , \quad si\ x_i \neq y_j \end{cases} \quad \textbf{(2.5)}$$

$$coste = \begin{cases} peso & , \ si\ x_i\ o\ y_j\ es\ 'N' \\ 0 & , \ sino \end{cases}$$

Por ejemplo, el experto obtiene la cadena "*LNP*", mientras que el estudiante obtiene "*NLP*". Sin pesos (**Tabla 2.31**) se obtendría una similitud Levenshtein del 33'3% (1 − 2/3). Pero se quiere restar importancia a los fallos temporales por lo que se decide utilizar un peso de 0'25. La similitud que obtendría, como se puede ver en la **Tabla 2.32**, sería del 50% (1 − 1'5/3).

		L	N	P
	0	1	2	3
N	1	1	1	2
L	2	1	2	2
P	3	2	2	2

Tabla 2.31: Matriz de distancias Levenshtein obtenida con las cadenas "*LNP*" y "*NLP*" (sin peso).

		L	N	P
	0	1	2	3
N	1	0'75	0'75	1'5
L	2	0'75	1'5	1'75
P	3	1'75	1'5	1'5

Tabla 2.32: Matriz de distancias Levenshtein obtenida con las cadenas "*LNP*" y "*NLP*" (con peso 0'25).

2.6.6.3. Similitudes Desiguales

Se ha de tener en cuenta a la hora de observar los resultados de similitud de Levenshtein y Damerau-Levenshtein que estos no tienen que coincidir con los de LCS puesto que no trabajan con las mismas operaciones ni miden la distancia de la misma forma. Por ejemplo, con las cadenas anteriores "*LNP*" y "*NLP*" sin pesos no coinciden, puesto que la similitud LCS obtenida (**Tabla 2.33**) es 66'6% (2/3), mientras que en el caso de Levenshtein (**Tabla 2.31**) era 33'3%. Lo mismo ocurre cuando introducimos los pesos, ya que la similitud LCS que se obtiene (**Tabla 2.34**) es 75% (2'25/3), cuando la obtenida por Levenshtein (**Tabla 2.32**) era 50%.

		L	N	P
	0	0	0	0
N	0	0	1	1
L	0	1	1	1
P	0	1	1	2

Tabla 2.33: Matriz de distancias LCS obtenida con las cadenas "*LNP*" y "*NLP*" (sin pesos)

		L	N	P
	0	0	0	0
N	0	0'25	1	1'25
L	0	1	1'25	1'25
P	0	1	1'5	2'25

Tabla 2.34: Matriz de distancias LCS obtenida con las cadenas "*LNP*" y "*NLP*" (con peso 0'25)

No sólo ocurre en este ejemplo, sino que tras realizar varias pruebas (**Figura 2.18** y **Figura 2.19**) se vio que, normalmente, tanto la similitud Levenshtein como la Levenshtein-Damerau son inferiores a la obtenida por LCS (aunque no siempre). Lo que sí se cumple siempre es que la similitud Damerau-Levenshtein va a ser igual o superior a la de Levenshtein debido a que soluciona errores de trasposición: si el estudiante cometió muchos obtendrá una similitud Levenshtein-Damerau superior, mientras que si no cometió ninguno obtendrá la misma similitud.

2.6.6.4. Versiones Sin Tiempo

Aunque con los pesos se le pueda restar importancia al tiempo en la evaluación de destreza, siempre afectará, en mayor o menor medida a la similitud obtenida. Por ejemplo, en las cadenas anteriores ("*LNP*" y "*NLP*") con un peso de 0'25, que es el máximo, se obtiene una similitud Levenshtein del 50%. Sin embargo, evaluando sólo los movimientos no-temporales la similitud que se obtendría es del 100%, ya que ambas cadenas son "*LP*".

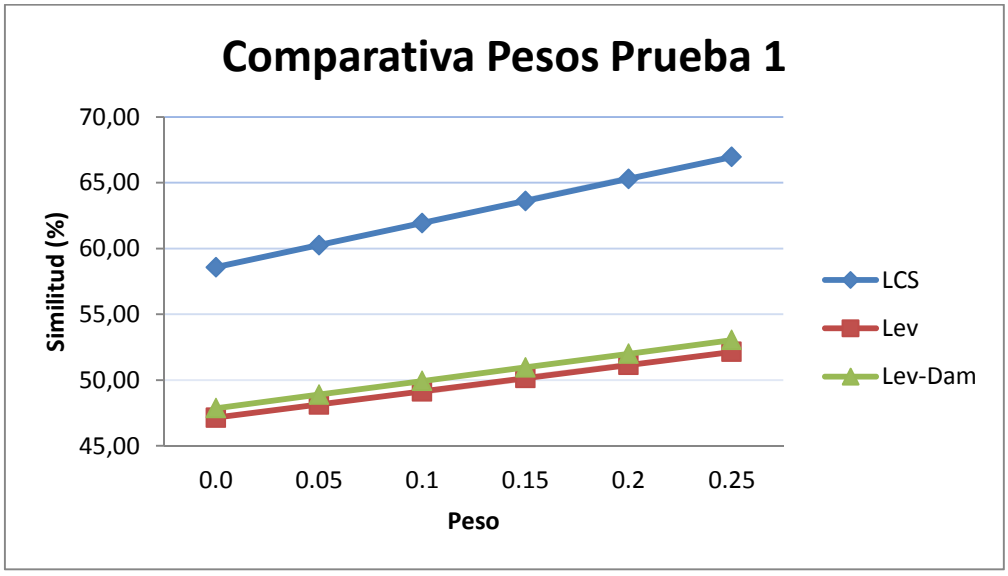

Figura 2.18: Gráfica que muestra las similitudes obtenidas con distintitas métricas y variando el peso en una determinada prueba.

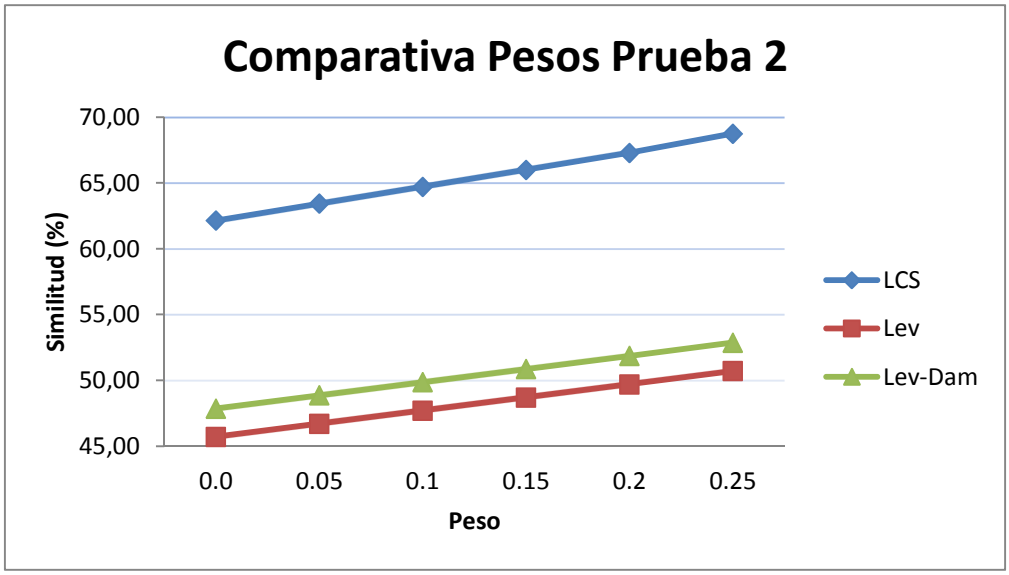

Figura 2.19: Gráfica que muestra las similitudes obtenidas con distintitas métricas y variando el peso en una determinada prueba.

También puede ocurrir que el tiempo beneficie a la similitud del estudiante si la mayoría de matches corresponden a caracteres *N*, de forma que sin ellos la similitud sería menor. Por ejemplo, las cadenas "*NPN*" y "*NQN*" tienen una gran similitud debido a los dos matches de las *N*, pero sin estos caracteres temporales sería 0%.

Debido a esto, se pensó que a un usuario avanzado le podría interesar conocer qué similitud obtuvo sólo con sus movimientos explícitos. Para proporcionar esta información se crearon unas versiones de los algoritmos de Levenshtein y Damerau-Levenshtein que ignoran caracteres temporales. De esta forma, si la similitud de estas versiones es superior a la de las versiones con tiempo significa que cometieron bastantes fallos temporales, mientras que si es inferior significa que cometieron bastantes fallos explícitos.

2.7. Controlando el Correcto Funcionamiento del Sistema

En secciones anteriores de este documento se ha descrito cómo funciona cada uno de los cuatro módulos que componen el proyecto, pero falta por explicar cómo funcionan en conjunto. Lo más lógico a priori sería crear un único programa que realizase todas las tareas, pero se descartó esta idea por las siguientes razones:

- Los escenarios y el tracker son incompatibles en un mismo programa. Esto es debido a que *ARToolkit* trabaja internamente con *OpenGL* pero no permite su manipulación, lo que impide tener dos ventanas en la misma aplicación.

- El programa resultante sería muy "pesado", lo que dificultaría su mantenimiento, su depuración, el testeo y el control/detección de errores.

- Si se desea utilizar, por ejemplo, otro tracker u otra *API* para los gráficos de los escenarios, habría que cambiar código del resto de partes, mientras que si funcionan por separado, basta con modificar el módulo correspondiente manteniendo su interfaz.

Por estas razones se adoptó el diseño actual, en el que cada módulo funciona por separado, como un programa independiente, pero se comunica con el resto de módulos.

2.7.1. Comunicando Módulos

Se optó por utilizar sockets para la comunicación entre módulos (en lugar de mediante ficheros, tuberías, memoria compartida, mensajes, llamadas a procedimiento remoto,...) debido a su simplicidad, ya que son fáciles de implementar y de utilizar.

Se crearon dos clases de socket con *Winsock2* (librería de *Windows*): uno que recibe datos (servidor) y otro que los envía (cliente). El funcionamiento en ambos casos es parecido: se conecta el socket con el otro extremo, se envían o reciben datos y se cierra la conexión.

En la **Figura 2.20** puede observarse como se integran ambas clases de sockets con el resto de módulos. La información que envía el tracker es numérica, mientras que la que envía el generador de movimientos son caracteres.

Con los sockets, además, se controla el inicio sincronizado del flujo de datos, ya que cada módulo se queda bloqueado hasta que sus sockets logren conectarse. De esta forma, hasta que todos los sockets estén conectados, ningún módulo comenzará a enviar/recibir datos, evitando así que se pueda perder información.

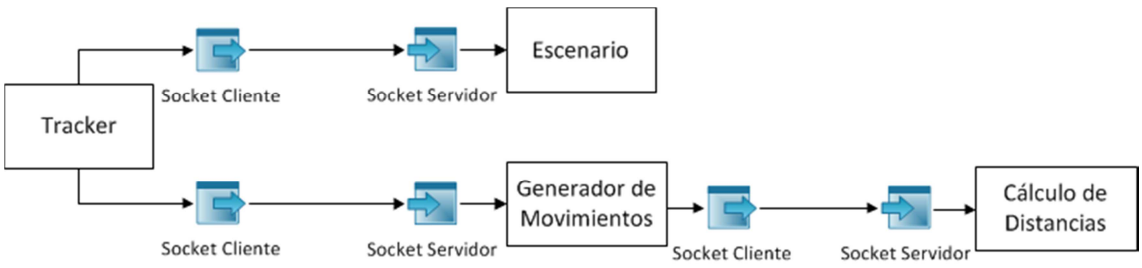

Figura 2.20: Envío y recepción de datos entre los distintos módulos gracias al uso de sockets.

2.7.2. Controlando la Finalización (In)Correcta

¿Qué ocurre cuando alguno de los módulos deja de funcionar? Si, por ejemplo, el módulo de generación de movimientos dejase de funcionar, la información que le envía el tracker se perdería, y el módulo de cálculo de distancias se quedaría parado puesto que no recibe datos.

¿Cuándo finaliza correctamente el sistema? Es el escenario *OpenGL* el que marca la finalización del cálculo de distancia online, por lo que es este módulo el encargado de decirle al tracker que pare de obtener datos, al generador de movimientos que pare de generar caracteres y al módulo de cálculo de distancias que pase a modo offline.

Tanto para evitar que el sistema se colapse, porque algún módulo deja de funcionar, como para que los módulos finalicen correctamente cuando el escenario termina, se necesita de un método de supervisión que controle el funcionamiento conjunto de todas las partes del sistema.

Se descartó usar un socket de control porque su implementación hubiese resultado demasiado compleja. La solución adoptada es añadir otro proceso al sistema que se encargue de supervisar el correcto funcionamiento del resto de procesos. A priori puede parecer igual de complejo que los sockets de control, pero es en realidad bastante sencillo (como puede verse en la **Figura 2.21**). Este proceso (llamado *controlador*) se encarga de:

- Iniciar todos los módulos (es el único ejecutable que el usuario ha de iniciar explícitamente para comenzar a usar el sistema).
- Comprobar periódicamente el estado de ejecución de cada uno de los procesos que ha iniciado.
- Si alguno de los procesos ha terminado, el controlador se encargará de detener el resto de procesos (salvo si fue OpenGL y terminó correctamente, que no detendrá el módulo de cálculo de distancia).

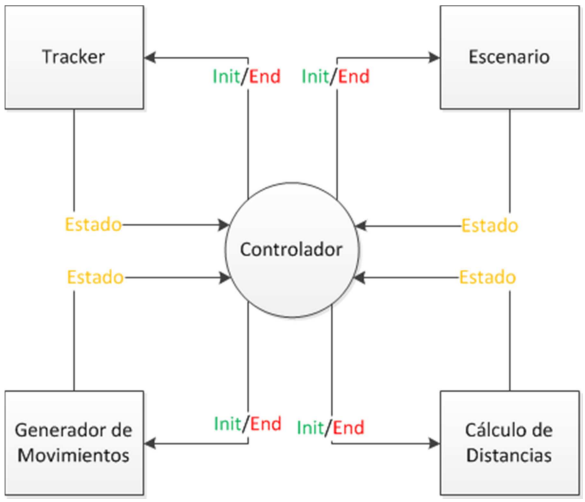

Figura 2.21: Diagrama de interacción entre el controlador y el resto de módulos para preservar el correcto funcionamiento del sistema.

2.8. El Interfaz de Usuario

Se tuvo que desarrollar un interfaz gráfico para dos módulos de este proyecto: para el controlador y para el cálculo de distancias. El resto de módulos disponían ya de uno (tracker y escenarios) o no necesitaban (generador de movimientos). A continuación se describirá brevemente en que consiste cada uno de estos dos interfaces, desarrollados ambos con la plataforma *Qt* (un *SDK* de *Nokia* para el desarrollo de interfaces de usuario) junto con *C++*.

2.8.1. Interfaz para el Cálculo de Distancias

Este modulo muestra la información en dos etapas: durante la realización del ejercicio (similitud LCS online y velocidad relativa de movimiento) y al finalizar el ejercicio (toda la información offline). Además, en esta segunda etapa, el estudiante puede guardar todos sus resultados.

En la primera etapa lo más importante es la similitud online, por lo que es el elemento principal de la ventana. Se busca que el estudiante conozca en todo momento si está mejorando o empeorando su destreza para que actué en consecuencia. Por ejemplo, la **Figura 2.22** el usuario no lo está haciendo demasiado bien por ahora pero, al menos, está mejorando respecto de su anterior movimiento.

En la etapa offline el usuario dispone de todo el tiempo del mundo para observar la información que se le muestra, por lo que simplemente se le muestra toda la información calculada tras el ejercicio. Por ejemplo, en la **Figura 2.23** se muestran los resultados de un usuario que realizó un ejercicio regular (40'91% similitud LCS), que otras medidas lo corroboran y que fue más rápido que el experto (21.7 segundos).

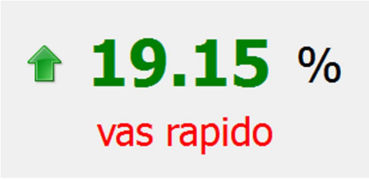

Figura 2.22: Captura del interfaz del módulo de cálculo de distancias durante la realización de un ejercicio.

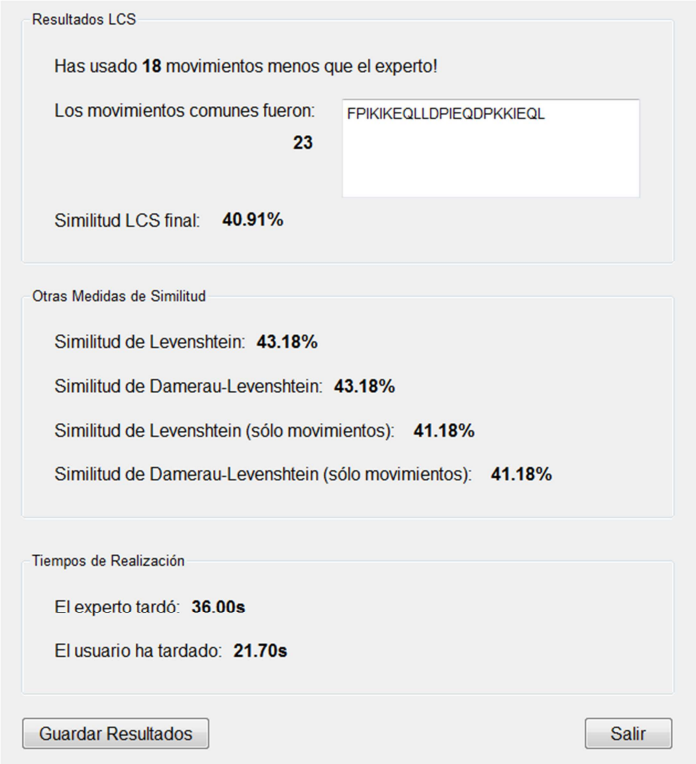

Figura 2.23: Captura del interfaz del módulo de cálculo de distancias tras la realización de un ejercicio.

2.8.2. Interfaz del Controlador

La aplicación de evaluación de destreza interactiva tiene dos parámetros: qué escenario se va a realizar y el grado de exigencia temporal con el que van a ser evaluados los movimientos (el peso para los algoritmos LCS, Levenshtein y Damerau-Levenshtein). Dado que Controlador es el punto de entrada a la aplicación y el encargado de iniciar el resto de módulos, es en su interfaz donde se ha de proporcionar valor a estos parámetros para que éste los comunique al módulo correspondiente

En la **Figura 2.24** puede verse como el usuario desea realizar el escenario 2 y que el evaluador de destreza sea estricto con el tiempo al obtener la similitud (peso 0).

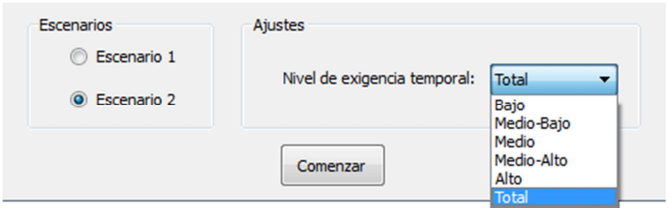

Figura 2.24: Captura del interfaz del módulo Controlador.

3. Validación y Resultados

3.1. Introducción

El objetivo de este proyecto es estudiar si mediante el uso de distancias entre cadenas se puede evaluar correctamente la destreza demostrada por un usuario mientras realiza un ejercicio (en este caso una laparoscopia). Por esta razón, lo que se va a validar es el cálculo de distancias. El resto de componentes descritos en este documento son herramientas necesarias para poder obtener la cadena de movimientos que realiza el usuario durante un escenario. Sin embargo, se ha de tener en cuenta que, pese a no ser evaluados directamente, sí que influyen en el cálculo de distancias, puesto que una mala captura y/o generación de los movimientos, difícilmente representará correctamente la destreza del estudiante.

3.2. Los Movimientos Expertos

Dado que la destreza del estudiante se evalúa según la distancia entre su cadena de movimientos y la del experto, si esta última cadena no es representativa de una buena destreza laparoscópica la evaluación será incorrecta. Por esta razón, se ha tomado un cuidado especial en la generación de la cadena del experto, para que sea una buena referencia de cómo realizar el ejercicio demostrando una gran destreza.

Como no se dispone de cirujanos expertos para crear la cadena experta, se decidió que el alumno podría hacer de experto, puesto que lleva más tiempo manejando el instrumental (tiene mayor experiencia que cualquiera de los usuarios que realicen la prueba durante la validación).

Se realizó un mismo escenario muchas veces, tratando de hacerlo perfecto en cada una de las repeticiones. Se descartaron aquellas pruebas en las que se cometió algún error, no se realizaron movimientos suficientemente fluidos, se movió demasiado rápido o lento,... para que la cadena fuese representativa de una buena destreza.

Al no disponer de una cadena de referencia, es imposible saber si una muestra es mejor que otra. Por esta razón, lo que se hizo fue utilizar cada una de las muestras como cadena experta y usar al resto como usuarios (*leave-one-out*), es decir, con cada una de las n muestras se obtienen $n-1$ similitudes. Después se calculó la similitud media de cada una de las muestras candidatas a experta, lo que permitió establecer un ranking. Por ejemplo, en la **Tabla 3.1** puede observarse como la muestra 8 tiene más en común con el resto de muestras que la número 5.

	M1	M2	M3	M4	M5	M6	M7	M8	Media
M1		64%	67'6%	64'9%	58%	65'5%	68'1%	65'8%	64'84%
M2			65%	58'8%	53'6%	71'7%	61'1%	60'3%	62'07%
M3				64'9%	58%	67'9%	64'6%	66'4%	62'58%
M4					58'6%	67'6%	62'8%	68'9%	63'64%
M5						57'5%	57'5%	61'9%	61'07%
M6							61'1%	67'1%	64'14%
M7								65'1%	63'18%
M8									65'07%

Tabla 3.1: Ejemplo de la obtención de representatividad de cada muestra como cadena experta.

Se volvió a repetir el proceso *leave-one-out* con las 10 muestras más representativas de este ranking para que, las menos representativas, no afectaran la búsqueda de la muestra experta. Sin embargo, en el nuevo ranking ninguna de estas 10 muestras tenía una similitud media superior a 75%, por lo que no eran suficientemente representativas.

Lo siguiente fue observar carácter a carácter estas 10 muestras candidatas a experta para ver qué tenían en común todas ellas. Se buscaban cuáles eran los movimientos comunes (aparecen en las 10 muestras) y los no tan comunes (aparecen en más de 6 muestras). Con esta información se creó una nueva cadena experta en la que aparecen todos los movimientos comunes y algunos de los menos comunes, en el orden correspondiente.

Por último, se efectuó de nuevo el proceso de *leave-one-out* con esta nueva muestra como experto y las otras 10 como usuarios, obteniendo una buena similitud en todos los casos. Por esta razón, se decidió que esta cadena era suficientemente representativa de un ejercicio bien realizado como para ser considerada experta.

3.3. Experimento

Lo que se desea validar en este proyecto es si, mediante el uso de distancias entre cadenas de movimientos, se puede evaluar la destreza de un usuario correctamente. Por lo tanto, para el experimento, se necesita que un conjunto de sujetos de experiencia conocida realicen un mismo escenario. El sistema será correcto si obtiene una similitud acorde a la experiencia que tienen.

El experimento que se plantea para la validación es el siguiente: que 15 sujetos pertenecientes a 3 grupos (según su experiencia previa) realicen 4 repeticiones de un mismo escenario, obteniendo así 20 muestras por grupo (60 en total). En concreto, estos tres grupos de experiencia previa son:

- **Usuarios Nada Expertos (NE):** nunca han manejado el instrumento laparoscópico y no tienen referencia de cómo lo hizo el experto.

- **Usuarios Algo Expertos (AE):** nunca han manejado el instrumento laparoscópico, pero si que saben cómo lo hizo el experto (se les muestra un video).

- **Usuarios Expertos (E):** se les permite manejar un poco el simulador antes del ejercicio (sin mostrarles el escenario), y se les enseña cómo lo hizo el experto.

Lo deseable es que el sistema obtenga mejores similitudes en los usuarios expertos que en los nada expertos. La diferencia de información previa entre un usuario algo experto y uno experto, o entre un usuario algo experto y uno nada experto, no es suficientemente grande como para que se le pueda pedir al sistema que los distinga entre si, por lo que esto será un objetivo secundario.

3.4. Análisis de Resultados

3.4.1. ¿Es capaz el sistema de distinguir los tres grupos de experiencia tras el ejercicio?

Una primera aproximación a validar el sistema offline es observando la similitud LCS media de las cuatro repeticiones obtenida por cada usuario al finalizar la prueba. Pero, como puede observarse en la **Figura 3.1**, esto no es concluyente. Por ejemplo, dos usuarios del grupo no experto demostraron mayor destreza que un usuario del grupo experto.

Si se obtiene la similitud media de cada grupo sí que se cumple el orden. Por ejemplo, en la **Figura 3.1** puede observarse como la similitud media de los NE es 28'70%, que es bastante menor de la de los AE (35'44%) y los E (40'28%). Sin embargo, la varianza dentro de cada grupo hace que las diferencias no sean significativas.

Por esta razón, se ha de comprobar si en cada una de las cuatro repeticiones sí que se cumple. En la **Figura 3.2** a simple vista se puede observar como se mantiene el orden entre las similitudes media de cada grupo a lo largo de los cuatro intentos (aunque en la cuarta repetición no lo parezca el grupo AE obtiene 48'35% y el grupo E un 48'42%). Aplicando el Test de Tukey se obtiene que la diferencia entre las medias del grupo NE y del grupo E son significativas ($p<0.05$) tanto en la primera como en la última repetición.

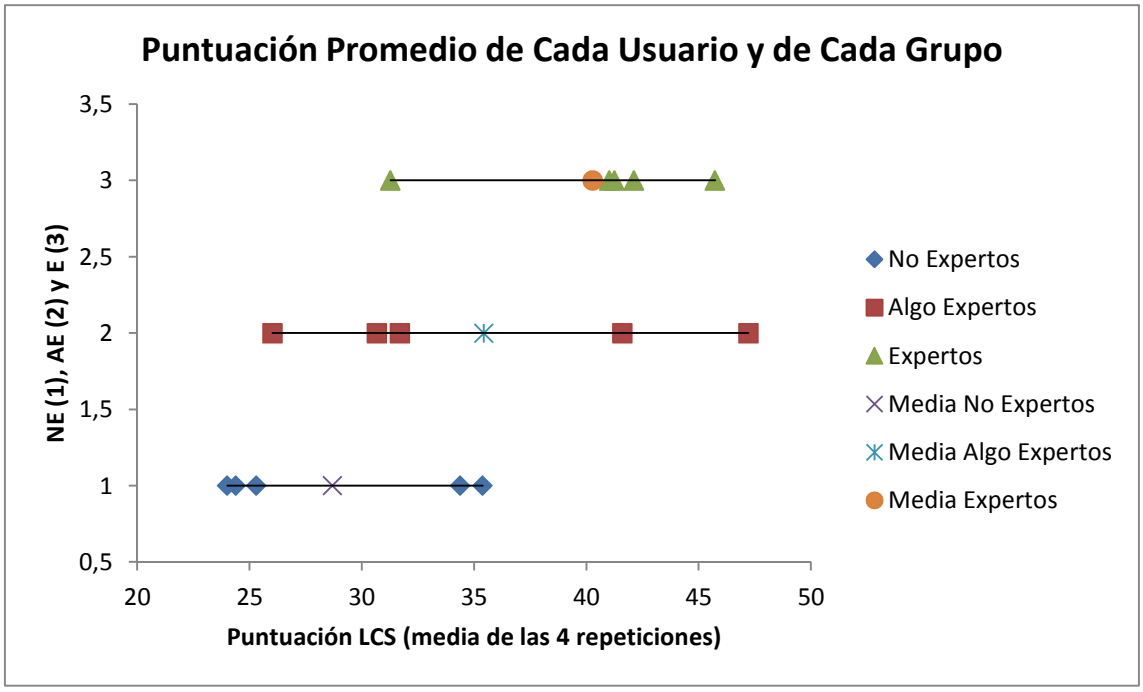

Figura 3.1: Similitud LCS promedio al terminar la prueba de cada uno de los 15 usuarios del experimento según su grupo de experiencia previa, y la similitud LCS promedio de cada grupo.

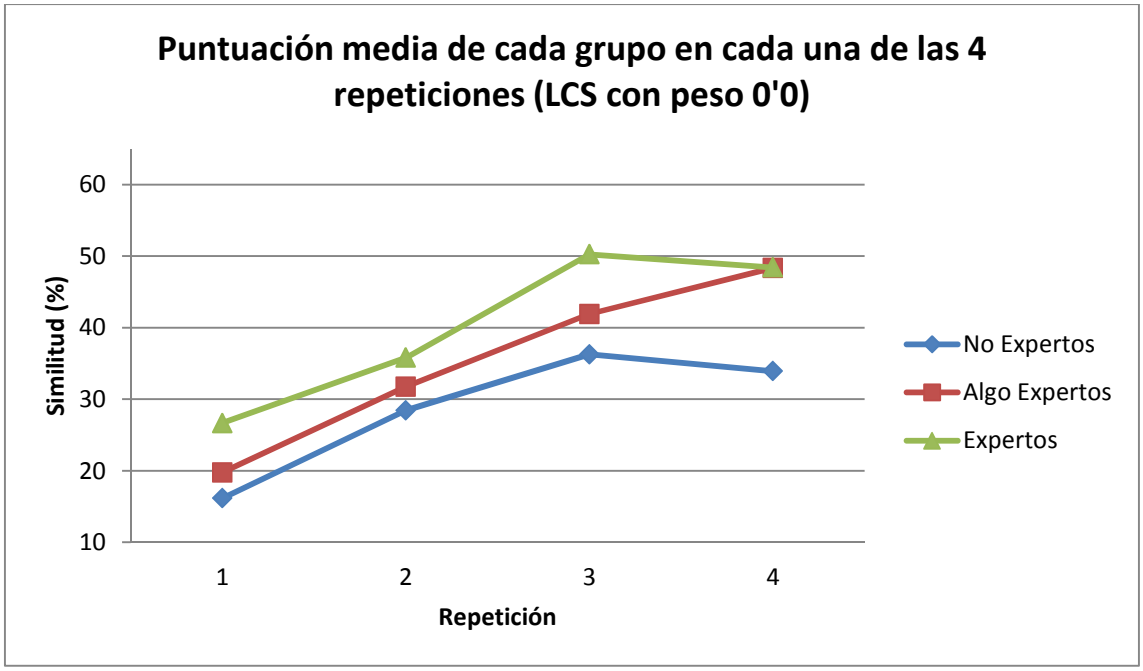

Figura 3.2: Similitud LCS (con peso 0) promedio de cada grupo de experiencia al terminar cada una de las cuatro repeticiones.

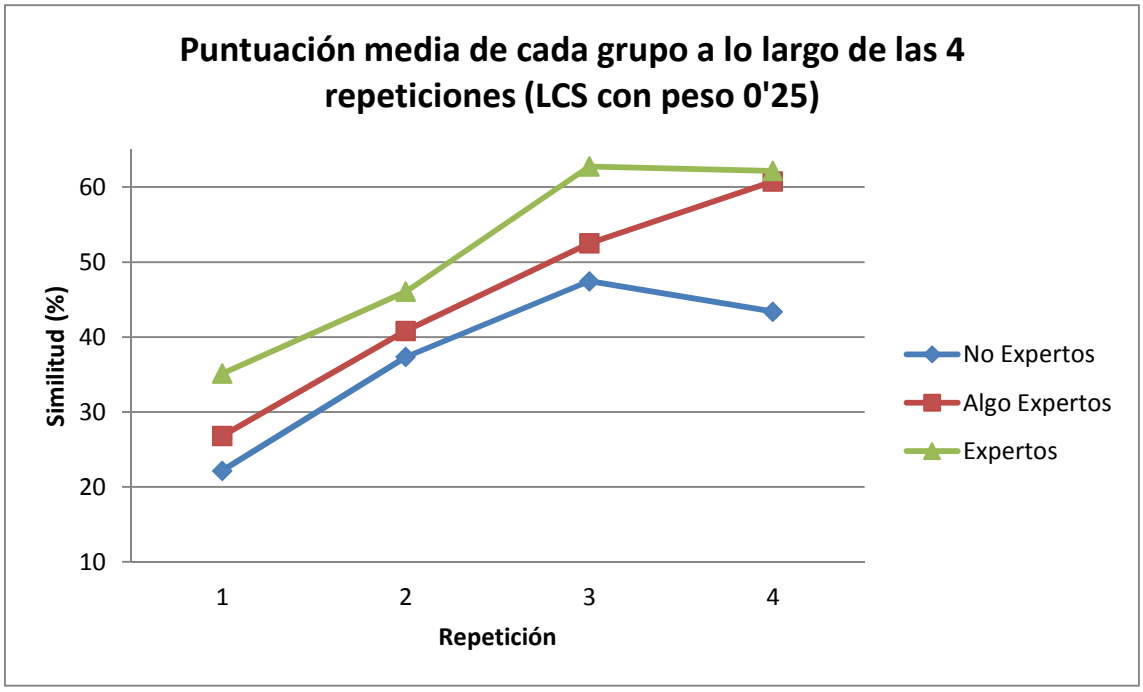

Figura 3.3: Similitud LCS (con peso 0'25) promedio de cada grupo de experiencia al terminar cada una de las cuatro repeticiones.

Si se aplica el Test de Mann-Whitney [**Anexo**] se obtiene que en un 68.75% (p=0.2429) de las comparaciones la curva AE está por encima de la NE, en otro 68.75% (p=0.2429) la curva E está por encima de la AE, y en un 75% (p=0.1714) de las comparaciones la curva E está por encima de la NE.

En la **Figura 3.3** se puede observar como con los pesos (en este caso el peso máximo 0'25) se sigue pudiendo diferenciar los distintos grupos de experiencia, y que las similitudes obtenidas son superiores a la versión sin pesos (ya que se penalizó menos los fallos temporales).

3.4.2. ¿Es capaz el sistema de distinguir los tres grupos de experiencia durante el ejercicio?

Dado que la evaluación de destreza no sólo se produce al finalizar el ejercicio sino que, de forma novedosa, también se produce durante el propio ejercicio, sería interesante ver si se sigue pudiendo diferenciar los niveles de destreza en todo momento durante la prueba.

En la **Figura 3.4** se muestra la curva LCS online de cada grupo de experiencia, que muestra la similitud promedio obtenida en cada segundo de ejercicio de sus 5 integrantes en sus 4 repeticiones. Como se puede observar en ella, a los pocos segundos de prueba ya comienza a ser diferenciable la similitud de los grupos E y AE de la del grupo NE.

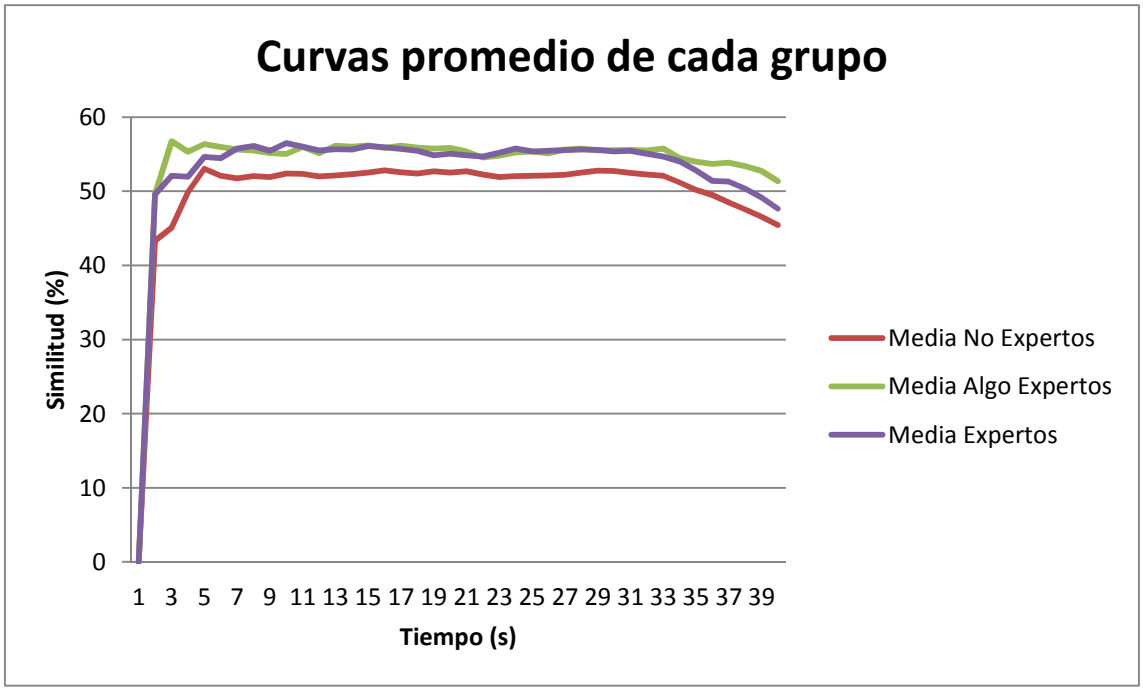

Figura 3.4: Curva de similitud online a lo largo de la prueba de cada grupo de experiencia (promedio de todos sus usuarios en sus cuatro repeticiones).

Las curvas del grupo E y AE se van entrelazando durante la realización de la prueba, lo que dificulta distinguir entre ambos grupos. Esto puede ser debido a que la diferencia que existe en la experiencia previa de ambos grupos es sólo haber practicado durante unos pocos segundos (en el caso del grupo E). Lo que realmente interesa es poder distinguir entre los grupos con alguna experiencia (AE y E) del grupo sin experiencia.

Una forma numérica de ver la diferencia que hay entre los grupos de experiencia según las curvas de similitud online es mediante el Test de Mann-Whitney [**Anexo**]. En este caso, se aplicará dicho test para conocer el porcentaje de tiempo en el que se cumple que, por ejemplo, la curva del grupo Experto está por encima de la del grupo No Experto. El test cruzará los valores de una curva con los de la otra para obtener el número de cruces en los que esto se cumple, que es el valor de *U*.

En la **Tabla 3.2** se muestran los resultados de dicho test, en los que se puede ver como en el 93'5% de los cruces (en total se dieron 1600) se cumplió que la curva NE estaba por debajo de la del grupo AE. Lo mismo ocurre en el 82% de los cruces entre la curva NE y la del grupo E. Sin embargo, sólo en el 40'2% de los casos se cumple que la curva AE esté por debajo de la del grupo E, lo que ya se pudo ver en la **Figura 3.4**. Además, en este último caso se da la hipótesis nula (no es <0'0001).

	U	z	P(1)	P(2)
NE vs AE	1496 (93'5%)	-6'69	<0'0001	<0'0001
NE vs E	1322 (82'6%)	-5'02	<0'0001	<0'0001
AE vs E	642'5 (40'2%)	1'51	0'0655	0'131

Tabla 3.2: Resultados del Test de Mann-Whitney para las curvas de similitud LCS online de cada par de grupos.

3.5. Conclusiones

A la vista de los resultados se puede concluir que mediante algoritmos de cálculo de distancias entre cadenas de movimientos se puede distinguir correctamente entre un usuario sin experiencia alguna (grupo NE) de un usuario con algo de experiencia (grupos E y AE). La evaluación de destreza será válida tanto durante la realización del ejercicio (online) como tras su finalización (offline).

Debido a la poca diferencia en la experiencia inicial que había entre los grupos Experto y Algo Experto en este experimento, no se puede saber si un usuario pertenece a uno de estos grupos o al otro. Quizás se tendría que haber marcado más la diferencia entre estos dos grupos de experiencia.

Pese a que no se busca evaluar si un sistema de aprendizaje online es mejor que uno offline, los resultados obtenidos dan indicios de que podría ser así, puesto que todos los usuarios mejoran a lo largo de sus primeras tres repeticiones (**Figura 3.2**). En la cuarta repetición no siempre se mejora respecto de la tercera, lo que puede ser debido a que la presión de querer mejorar en su último intento les hace cometer más fallos (imprecisión, ir demasiado rápido,...).

4. Conclusiones y Trabajo Futuro

En este proyecto se ha desarrollado un método de evaluación interactiva y automática de destrezas mediante el uso de distancias entre cadenas. Este método es novedoso en dos aspectos: es la primera vez que se ha utilizado la similitud entre cadenas en el ámbito de la evaluación de destrezas y es la primera vez que se desarrolla un algoritmo de cálculo de distancia que trabaje de forma interactiva

Los resultados obtenidos en la validación han sido satisfactorios, puesto que no sólo verifican que con distancias entre cadenas de movimientos se puede distinguir correctamente entre niveles de destreza experta y no experta sino que, además, se puede hacer tanto durante el ejercicio como tras su finalización.

Queda por validar si este método de evaluación interactiva proporciona una curva de aprendizaje mejor que los sistemas de evaluación actuales (no interactivos). Los resultados de la validación dan indicios de que podría ser así, pero habría que realizar un estudio más exhaustivo (con expertos cirujanos y estudiantes reales).

Gracias a utilizar la versión *timeless* (sin caracteres temporales) de los algoritmos Levenshtein y Damerau-Levenshtein en la fase offline se ha detectado que, sin el tiempo, las similitudes suelen empeorar (entre 3-12%). Esto quiere decir que, teniendo en cuenta únicamente los movimientos explícitos, los usuarios lo hacen peor de lo que las puntuaciones indican, es decir, que el tiempo les está sobrevalorando. Habría que consultar con un especialista para ver el papel que juega el tiempo en laparoscopia, para saber si le hemos dado demasiada importancia (o demasiado poca) en la evaluación de destreza.

Otro aspecto mejorable ocurre cuando el experto termina antes que el usuario (y al revés, aunque ocurre menos). Como se comentó en la sección **2.6.1.2,** se decidió que cuando uno de los dos terminase antes que el otro, el resto de movimientos fuesen fallos. Esto se traduce en que, cuando un usuario tarda más movimientos que el experto, haga lo que haga después, no podrá mejorar su similitud. Como se puede observar en la **Figura 3.4**, las tres curvas empiezan a descender a partir del segundo 36 (lo que tardó el experto).

Una posible solución a esto sería tener en cuenta también los movimientos en exceso para el cálculo de similitud online, para que, aunque el estudiante tarde más que el experto, sus movimientos puedan ser correctos. Igual que en el caso anterior, lo ideal sería comentárselo a un especialista para que dijese si se ha de penalizar a los estudiantes “tardones” o no.

Anexo

Linear Discriminant Analysis (LDA)

Se utiliza para hallar la combinación lineal de una serie de variables que caracterizan o separan una serie de grupos/clases. Es decir, trata de encontrar un modelo de la diferencia entre los diferentes grupos/clases y los umbrales que las separan.

La combinación lineal resultante del proceso puede ser utilizada como clasificador lineal, para predecir la pertenencia a uno de los grupos/clases (**Figura I**) o para reducir la dimensión del número de variables a tener en cuenta por posteriores clasificadores.

Principal Components Analysis (PCA)

Convierte un conjunto de observaciones de variables posiblemente correlacionadas en un conjunto de valores de variables no-correlacionadas llamadas *Principal Components (PC)*.

El número de *PC*s es menor o igual al número original de variables, lo que hace a *PCA* un método útil para reducir la dimensión del número de variables en un problema de clasificación (**Figura II** y **Figura III**).

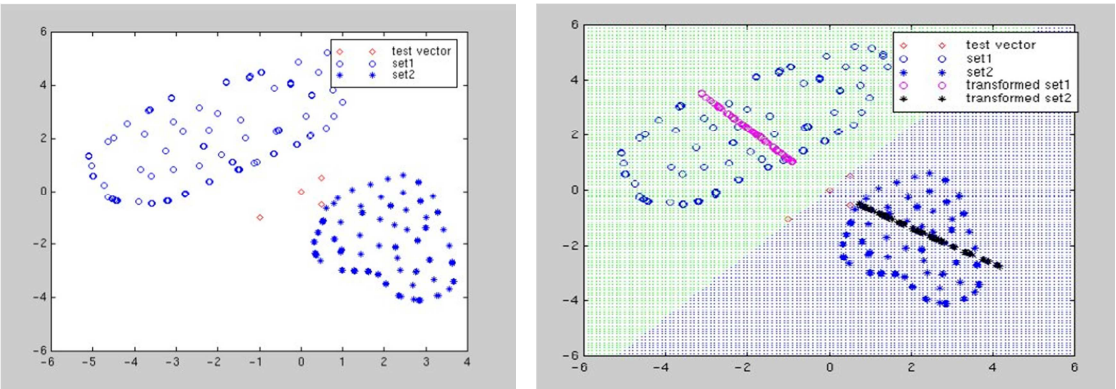

Figura I: En la gráfica de la izquierda se pueden ver los datos originales (muestras de dos clases en azul y las muestras en rojo de los datos a clasificar). En la gráfica de la derecha se pueden ver la *partición que se hizo* del espacio con el análisis discriminante, y como los datos a clasificar parece que caen dentro de la clase de la derecha (fondo azul).

La transformación está definida de forma que la primera componente muestra la mayor varianza que sea posible, es decir, contiene la mayor variabilidad en los datos posible. Cada componente sucesiva tendrá la mayor varianza restante posible.

Usando la varianza acumulada (*CVE*) de las *PCs* podemos establecer un criterio para seleccionar el número de éstas a usar en posteriores etapas de clasificación. A mayor correlación entre las variables originales menos *PCs* podremos usar.

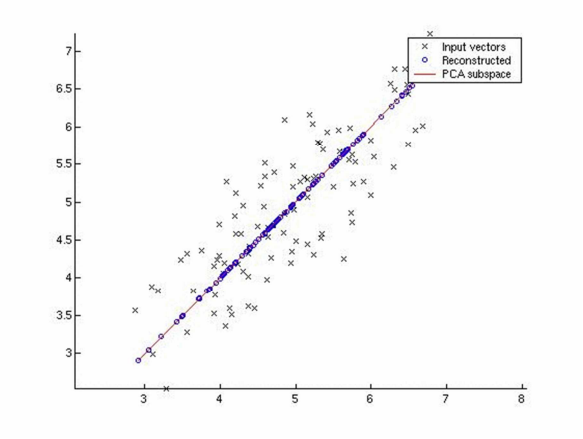

Figura II: En esta gráfica se puede ver como un conjunto de datos en 2 dimensiones queda reducido a un conjunto menor de datos en 1 dimensión.

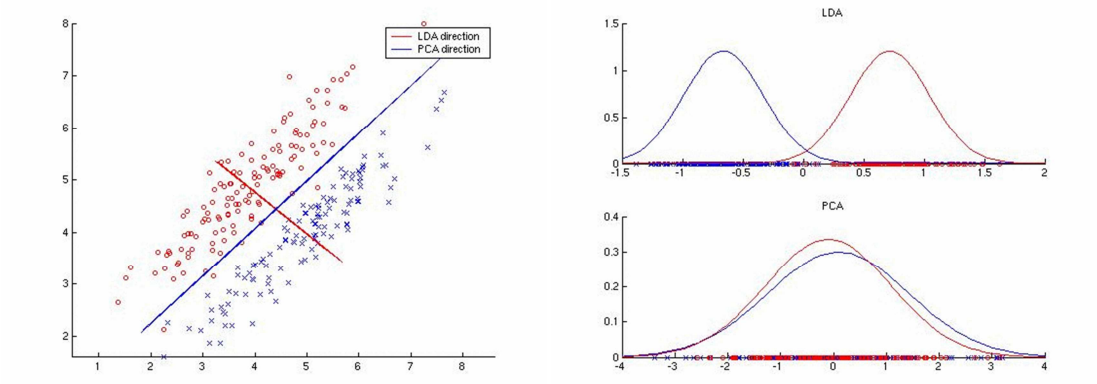

Figura III: En las siguientes gráficas podemos ver la diferencia entre los métodos LDA y PCA para reducir las dimensiones de los datos

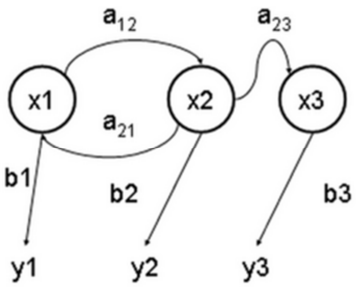

Figura IV: Ejemplo de transición de estados en un HMM. Las X son los estados ocultos, las Y son las salidas observables, las A son las probabilidades de transición, y las B son las probabilidades de salida.

Hidden Markov Model (HMM)

Un Modelo de Markov es un modelo estadístico en el que se asume que el sistema a modelar es un proceso de Markov, es decir, un proceso estocástico con la propiedad de que cualquier predicción sobre el siguiente valor de la secuencia, conociendo los estados anteriores, puede estar basada sólo en el estado precedente.

Lo que lo distingue de un Modelo de Markov común es que hay una serie de parámetros desconocidos. El objetivo es determinar estos parámetros ocultos a partir de parámetros observables.

Los estados de un *HMM* no son visibles directamente, sino que sólo las variables a las que influye lo son. Además, cada estado tiene una distribución de probabilidad sobre los posibles símbolos de salida.

Para definir un *HMM* se necesita (**Figura IV**): $\lambda = (A, B, \pi)$

- el número de estados del modelo (N),
- la matriz de distribución de probabilidad de transición entre estados (A),
- la matriz de distribución de probabilidad de observación de símbolos (B),
- y el vector de distribución de estados inicial (π).

Una vez definida la arquitectura del *HMM* hay tres problemas de interés:

- *El problema de la evaluación*: obtener la probabilidad (P) de la secuencia de observación dado un modelo (λ) y dicha secuencia (O)

$$P(O|\lambda)$$

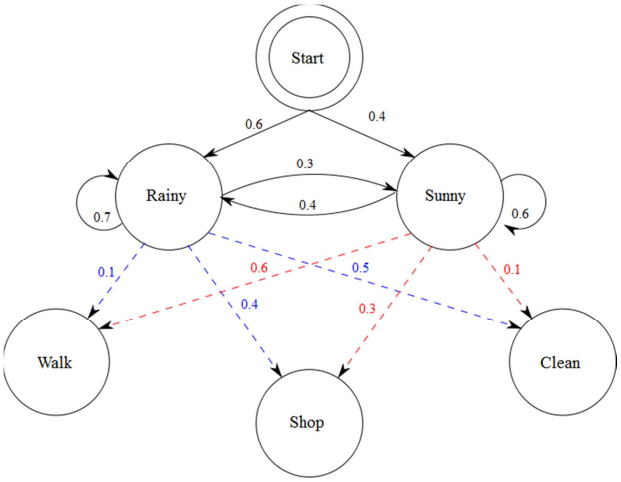

Figura V: HMM de una persona realiza una de las 3 actividades (caminar, comprar o limpiar) según el tiempo que haga ese día. No se conocen las probabilidades de que sea un día lluvioso o soleado (aunque se conoce la tendencia general del tiempo). Basándose en lo que hizo la persona se puede saber el tiempo que hizo ese día.

- *Descubrir los estados ocultos*: obtener la secuencia de estados ocultos (Q) dada la secuencia de observación y el modelo.

$$Q = q_1, q_2, \ldots, q_T$$

- *El problema del entrenamiento*: ajustar los parámetros del modelo para maximizar la probabilidad de observación de secuencias.

En la **Figura V** se puede ver un ejemplo real de uso de Modelos de Markov.

Fuzzy Classifier

Utilizan conocimiento informal y previo sobre el dominio del problema para la clasificación. Por ejemplo: el salmón es flaco y de color claro, mientras que la lubina es corpulenta y de color oscuro.

El objetivo de este tipo de clasificadores es crear unas funciones difusas de pertenencia a unas determinadas categorías para que, a partir de unos datos/parámetros de entrada, se obtenga la pertenencia de estos a una de dichas categorías. Pero no hay que confundir estas categorías con las clases finales, ya que hacen referencia a rangos de valores de los parámetros que se solapan.

Por ejemplo, se quiere realizar un clasificador *Fuzzy* para distinguir entre varios tipos de peces. Uno de los datos de entrada es el color de la piel, por lo que se decide que habrán cuatro categorías de color de piel según su reflectividad (**Figura VI**): muy oscura, oscura, clara y muy clara.

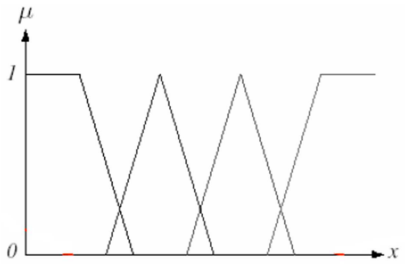

Figura VI: El parámetro x representa la reflectividad de la piel del pez. El diseñador cree que hay cuatro categorías: muy oscura, oscura, clara y muy clara. La primera va desde 0 hasta la primera intersección, la segunda de esta intersección a la siguiente,…

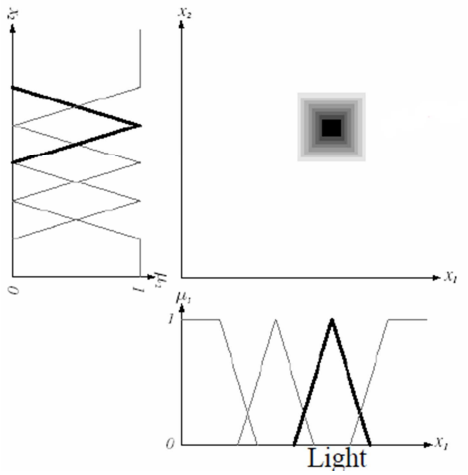

Figura VII: El parámetro x_1 es la reflectividad de la piel, mientras que x_2 es el tipo del cuerpo del pez. Si queremos clasificar un pez como salmón se ha de cumplir ese rango de valores en negrita, cuanto más cerca del centro más seguro estaremos de que es un salmón.

Estas funciones de pertenencia a categorías, derivadas del conocimiento previo del problema por parte del diseñador del clasificador, junto con unas reglas de conjunción llevarán a los discriminantes.

Si juntamos varias funciones de categoría de diferentes parámetros, obtenemos una regla de conjunción, que dará un número que ayude a tomar la decisión final. En la **Figura VII** se puede ver el clasificador resultante para el problema de distinción entre peces mencionado anteriormente.

Test de Mann-Whitney

También conocida como *Prueba U de Whitney, Prueba de Mann-Whitney-Wilcoxon* y *Prueba de la Suma de Rangos Wilcoxon*, se trata de una prueba no paramétrica con la que se identifican diferencias entre dos poblaciones basadas en el análisis de dos muestras independientes, cuyos datos han sido medidos en una escala de nivel ordinal.

La prueba calcula el *estadístico U* cuya distribución para muestras con más de 20 observaciones se aproxima bastante bien a la distribución normal *Z*.

Para calcular este valor se asigna a cada uno de los valores de las muestras un rango para construir las ecuaciones U_1 y U_2 (donde n_1 y n_2 son los tamaños de cada muestra, y R_1 y R_2 la suma de los rangos de las observaciones de cada muestra):

$$U_1 = R_1 - \frac{n_1(n_1+1)}{2} \quad \textbf{(I)}$$

$$U_2 = R_2 - \frac{n_2(n_2+1)}{2} \quad \textbf{(II)}$$

El estadístico U se define como el mínimo entre U_1 y U_2.

Una forma simple de ver este test es la siguiente. En una carrera entre 19 tortugas (T) y 19 liebres (L) se obtiene la siguiente secuencia de llegada a meta: primero llegan 9 liebres, después 10 tortugas, luego 10 liebres, y por último 9 tortugas. La media nos dice que las tortugas (posición media 19ª) son más rápidas que las liebres (posición media 20ª). Sin embargo, el valor de la *U* dice todo lo contrario:

- U(liebres) = (9 liebres perdieron ante 0 tortugas) + (10 liebres perdieron ante 10 tortugas) = 0 + 10*10 = 100.
- U(tortugas) = (10 tortugas perdieron ante 9 liebres) + (9 tortugas perdieron ante 19 liebres) = 10*9 + 9*19 = 261.

Los valores *U* evidencian que las liebres suelen hacerlo mejor que las tortugas, ya que hay un mayor número de casos en el que las tortugas pierden ante liebres (261 frente a 100). Visto de otra forma, si tenemos en cuenta que el número de comparaciones totales es $19 * 19 = 361$, entonces un 72'3% de las veces las tortugas perdieron ante las liebres.

Bibliografía

[1] L. Allison and T.I. Dix, "A Bit-String Longest-Common-Subsequence Algorithm," *Information Processing Letters*, vol. 23, no. 5, pp. 305-310, 1986.

[2] A. Apostolico and C. Guerra, "The longest common subsequence problem revisited," *Algorithmica*, vol. 2, no. 1-4, pp. 315-336, 1987.

[3] L. Bergroth, H. Hakonen, and T. Raita, "A Survey of Longest Common Subsequence Algorithms," in *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, A Coruña (Spain), 2000, p. 39.

[4] M.K. Chmarra, C.A. Grimbergen, and J. Dankelman, "Systems for tracking minimally invasive surgical instruments," *Minimally Invasive Therapy & Allied Technologies*, vol. 16, no. 6, pp. 328-340, 2007.

[5] M.K. Chmarra, S. Klein, J.C.F. Winter, F.W. Jansen, and J. Dankelman, "Objective Classification of Residents Based on their Psychomotor Laparoscopic Skills," *Surg Endoscopy*, vol. 24, no. 5, pp. 1031–1039, 2010.

[6] F.J. Damerau, "A Technique for Computer Detection and Correction of Spelling Errors," *Communications of the ACM*, vol. 7, no. 3, pp. 171-176, 1964.

[7] I Hajshirmohammadi and S. Payandeh, "Fuzzy Set Theory for Performance Evaluation in a Surgical Simulator," *Presence: Teleoperators and Virtual Environments*, vol. 16, no. 6, pp. 603-622, 2007.

[8] P.A.V. Hall and G.R. Dowling, "Approximate String Matching," *ACM Computing Surveys (CSUR)*, vol. 12, no. 4, pp. 381-402, 1980.

[9] D.S. Hirschberg, "A Linear Space Algorithm for Computing Maximal Common Subsequences," *Communications of the ACM (CACM)*, vol. 18, no. 6, pp. 341-343, 1975.

[10] D.S. Hirschberg, "Algorithms for the Longest Common Subsequence Problem," *Journal of the ACM (JACM)*, vol. 24, no. 4, pp. 664-675, 1977.

[11] J Huang, S. Payandeh, P Doris, and I. Hajshirmohammadi, "Fuzzy Classification: Towards Evaluating Performance on a Surgical Simulator," *Studies in Health Technology and Informatics*, vol. 111, pp. 194-200, 2005.

[12] J.W. Hunt and T.G. Szymanski, "A Fast Algorithm for Computing Longest Common Subsequences," *Communications of the ACM (CACM)*, vol. 20, no. 5, pp. 350-353, 1977.

[13] S.K. Kumar and C.P. Rangan, "A Linear Space Algorithm for the LCS Problem," *Acta Informatica*, vol. 24, no. 3, pp. 353-362, 1087.

[14] S. Kuo and G.R. Cross, "An Improved Algorithm to Find the Length of the Longest Common Subsequence of Two Strings," *ACM SIGIR Forum*, vol. 23, no. 3-4, pp. 89-99, 1989.

[15] V. Lahanas, C. Loukas, N. Nikiteas, D. Dimitroulis, and E. Georgiou, "Psychomotor Skills Assessment in Laparoscopic Surgery Using Augmented Reality Scenarios," in *17th International Conference on Digital Signal Processing (DSP)*, Corfu (Greece), 2011.

[16] J.J.H. Leong et al., "HMM Assessment of Quality of Movement Trajectory in Laparoscopic Surgery," in *International Conference on Medical Image Computing and Computer-Assisted Intervention.*, 2006, pp. 752-759.

[17] H. C. Lin, I. Shafran, D. Yuh, and G. D. Hager, "Towards Automatic Skill Evaluation: Detection and Segmentation of Robot-Assisted Surgical Motions," *Computer Aided Surgery*, vol. 11, no. 5, pp. 220-230, 2006.

[18] Z. Lin et al., "Objective Evaluation of Laparoscopic Surgical Skills Using Waseda Bioinstrumentation System WB-3," in *Proceedings of the 2010 IEEE International Conference on Robotics and Biomimetics*, Tianjin (China), 2010, pp. 247-252.

[19] W.J. Masek and M.S. Paterson, "A Faster Algorithm Computing String Edit Distances," *Journal of Computer and System Sciences*, vol. 20, no. 1, pp. 18-31, 1980.

[20] G. Megali, S. Sinigaglia, O. Tonet, and P. Dario, "Modelling and Evaluation of Surgical Performance Using Hidden Markov Models," *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 10, pp. 1911-1919, 2006.

[21] E.W. Myers, "An O(ND) Difference Algorithm and its Variations," *Algorithmica*, vol. 1, pp. 251-266, 1986.

[22] G. Navarro, "A Guided Tour to Approximate String Matching," *ACM Computing Surveys (CSUR)*, vol. 33, no. 1, pp. 31-88, 2001.

[23] J. Rosen et al., "The Blue DRAGON - A System for Monitoring the Kinematics and the Dynamics of Endoscopic Tools in Minimally Invasive Surgery for Objective Laparoscopic Skill Assessment," *Studies in Health Technology and Informatics*, vol. 85, pp. 412-418, 2002.

[24] J. Rosen, J. D. Brown, Chang L., M. N. Sinanan, and B. Hannaford, "Generalized Approach for Modeling Minimally Invasive Surgery as a Stochastic Process Using a Discrete Markov Model," *Biomedical Engineering, IEEE Transactions on*, vol. 53, no. 3, pp. 399-413, 2006.

[25] J. Rosen et al., "Minimally Invasive Surgery Task Decomposition - Etymology of Endoscopic Suturing," *Studies in Health Technology and Informatics*, vol. 94, pp. 295-301, 2003.

[26] J. Rosen, C. Richards, B. Hannaford, and M. Sinanan, "Hidden Markov Models of Minimally Invasive Surgery," in *Medicine Meets Virtual Reality 2000*.: IOS Press, 2000, pp. 279-285.

[27] J. Rosen, M. Solazzo, B. Hannaford, and M. Sinanan, "Objective Laparoscopic Skills Assessments of Surgical Residents Using Hidden Markov Models Based on Haptic Information and Tool/Tissue Interaction," *Studies in Health Technology and Informatics*, vol. 81, pp. 417-423, 2001.

[28] J. Solis et al., "Quantitative Assessment of the Surgical Training Methods with the Suture/Ligature Training System WKS-2RII," in *IEEE International Conference on Robotics and Automation*, Kobe (Japan), 2009.

[29] W. Soto and Y. Pinzón, "Sobre el Longest Common Subsequence: Extensiones y Algoritmos," *Revista Colombiana de Computación (RCC)*, vol. 8, no. 2, 2007.

[30] G.A. Stephen, *String Searching Algorithms*.: World Scientific, 1994.

[31] N. Stylopoulos et al., "CELTS: A clinically-based Computer Enhanced Laparoscopic Training System," *Surgical Endoscopy*, vol. 18, no. 5, pp. 782-789, 2004.

[32] E. Ukkonen, "Algorithms for Approximate String Matching," *Information and Control*, vol. 64, no. 1-3, pp. 100-118, 1985.

[33] R.A. Wagner and M.J. Fischer, "The String-to-String Correction Problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168-173, 1974.

[34] S. Wu, U Manber, G. Myers, and W. Miller, "An O(NP) Sequence Comparison Algorithm," *Information Processing Letters*, vol. 35, no. 6, pp. 317-323, 1990.