FACULTAD DE INGENIERIA Y CIENCIAS APLICADAS

ESTUDIO COMPARATIVO DE ALGORITMOS DE RECONOCIMIENTO DE GESTOS

Trabajo de Titulación presentado en conformidad con los requisitos establecidos para optar por el título de Ingeniera en Sistemas de Computación e Informática

Profesor guía

PhD. Jorge Luis Pérez Medina

Autora

Ana Belén Erazo Baroja

AÑO

2019

DECLARACIÓN DEL PROFESOR GUÍA

"Declaro haber dirigido el trabajo, Estudio comparativo de algoritmos de reconocimiento de gestos, a través de reuniones periódicas con el estudiante Ana Belén Erazo Baroja en el semestre 201920, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación".

_____

Jorge Luis Pérez Medina
Doctor Especialidad Informática
C.C. 1758993123

DECLARACIÓN DEL PROFESOR CORRECTOR

"Declaro haber revisado este trabajo, Estudio comparativo de algoritmos de reconocimiento de gestos, de la estudiante Ana Belén Erazo Baroja, en el semestre 201920, dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación".

_____

Mario Salvador Gonzáles

Doctor en Ingeniería Informática y Telecomunicación

C.C. 0958376345

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes.”

_____

Ana Belén Erazo Baroja

C.C. 1725437618

AGRADECIMIENTOS

Agradezco a mi familia por acompañarme en todo este proceso, al PhD. Jorge Luis Pérez Medina, por su apoyo, guía y asesoramiento continuo.

DEDICATORIA

Este proyecto está dedicado a mis padres, Tomás y Diana, por su confianza y apoyo brindado, tanto en la vida profesional como personal; a mis hermanas María Paula y Aura Cristina, por ser mis cómplices, a mis abuelitas, tías, tíos y primos, y mi Papico, que son parte fundamental de mi historia.

RESUMEN

El presente proyecto de titulación tiene como objetivo realizar un estudio comparativo entre los algoritmos de reconocimiento de gestos de !FTL, $1, $P, $Q y Penny Pincher, con el uso de los *datasets* de NicIcon, Hhreco y Alfabeto Utopiano. El resultado de este trabajo es poder demostrar, con cuadros estadísticos, ¿cuál de los algoritmos resulta ser más eficiente, para facilitar el proceso de selección, al momento de requerirse incorporar interacciones naturales mediante gestos en las aplicaciones de *software*?

Para la experimentación se realizó un ambiente de pruebas, desarrollado en JavaScript. Se definieron los *datasets* candidatos para cada uno de los *datasets* a comparar, y todas las pruebas se realizaron en el mismo computador. Con ello, se pudieron llevar a cabo todos los algoritmos a un mismo nivel, con el fin de asegurar que los resultados de las comparaciones no puedan ser alterados por variables externas.

Antes de iniciar la experimentación, se establecieron hipótesis, con la intención de predecir los posibles resultados según el grado de dificultad de cada *database*. Se evaluaron la cantidad de gestos similares antes y después del *resampling* de las representaciones del *dataset.* En consecuencia, se pudieron definir rangos de error que podrían llegar a tener los resultados finales de cada *dataset*.

Con los resultados finales, se buscó obtener el tiempo de reconocimiento, la tasa de reconocimiento y la tasa de error de un algoritmo según el gesto evaluado. Con estos resultados, se analizaron y compararon las hipótesis, para definir el algoritmo más eficiente. Gracias al uso de la herramienta estadística de Excel, estos resultados pueden ser mejor observados y analizados para mostrar en gráficos las deducciones obtenidas de la experimentación del proyecto, y así definir los mejores algoritmos de reconocimiento.

ABSTRACT

The objective of the present project is to carry out a comparative study between the gesture recognition algorithms of the FTL, $ 1, $ P, $ Q and Penny Pincher, with the use of the datasets of NicIcon, Hhreco and Utopiano Alphabet. The result of this work is to be able to demonstrate, with statistical tables, which are the most efficient algorithms to facilitate the selection process of these algorithms when it is required to incorporate natural interactions through gestures in software applications.

For the experimentation of the project, a testing environment was developed, which was developed in JavaScript, the candidate datasets were defined for each of the datasets to be compared, and all the tests were performed on the same computer. With this, all the algorithms could be taken to the same level, in order to ensure that the results of the comparisons will not be altered by external variables.

Prior to the realization of the experimentation, the datasets were analyzed to carry out hypotheses of what could be the possible results according to the difficulty of this. They were evaluated according to the number of similar gestures before and after the resampling of the representations of the dataset. As a consequence, it was possible to define certain error ranges that could have the final results of each dataset.

With the results, what was sought is to obtain the recognition time, the recognition rate and the error rate of an algorithm according to a gesture. With these results, we could analyze and compare between them and the hypotheses to define which is the most efficient algorithm. Thanks to the use of the statistical tool of Excel, these results can be better observed and analyzed to show in graphs the deductions obtained from the experimentation of the project, and thus define the best recognition algorithms.

ÍNDICE

1. Capítulo I. Introducción

Las interfaces de usuario que han sido creadas con base al reconocimiento de gestos son utilizadas para poder interactuar de una manera más natural con un sistema. Estas aplicaciones han sido utilizadas para apoyar los procesos de diseño de diagramas, creación de prototipos de interfaz de usuario, el pedido de alimentos en línea, transcribir un escrito a mano desde un dispositivo externo, o para crear gestos en un dispositivo móvil.

En la actualidad, una amplia gama de sistemas operativos soporta el uso de interfaces basadas en gestos. Para poder recopilar estos gestos, se utilizan varias técnicas como: el aprendizaje automático, el emparejamiento basado en plantillas y el reconocimiento de patrones.

Para el procesamiento de los gestos, se utilizan algoritmos de reconocimiento que tratan de distinguir el gesto que se trazó, comparándolo con gestos de referencia previamente conocidos por el sistema y retornado el gesto más cercano, que se haya encontrado.

1.1 Justificación del Trabajo

Actualmente, el principal problema para el reconocimiento de gestos es la dificultad para reconocer el algoritmo más rápido y eficiente. Esto se debe a que todos los algoritmos han sido desarrollados en ambientes diferentes, es decir, distintos lenguajes de programación, en donde las pruebas han sido realizadas con diferentes conjuntos de gestos, comúnmente llamados “*datasets*”.

Mediante la revisión de la literatura, se observa que no existe un mecanismo de comparación, donde se puedan someter a los algoritmos de reconocimiento de gestos a un proceso donde las condiciones sean similares. Esto trae como consecuencias que se generen solo hipótesis y/o supuestos acerca de los algoritmos con los mayores niveles de rendimiento.

Una posible solución a esta inquietud, es realizar un estudio comparativo entre los reconocedores de gestos más destacados como lo son: $1 (Wobbrock, Wilson, & Li, 2007), $P (Vatavu, Anthony, & Wobbrock, 2012), Penny Pincher (Eugene M. Taranta & Joseph J. LaViola, 2015), $Q (Vatavu, Anthony, & Wobbrock, 2018), !FTL (Vanderdonckt, Roselli, & Pérez, 2018), utilizando como parámetros de evaluación su tasa de reconocimiento y tiempo de ejecución.

Para cumplir con ello, se levantará un caso de estudio, que tome como referencia un conjunto de gestos, mismos que serán usados de manera individual, en el proceso de evaluación de los algoritmos, con la finalidad de comparar los algoritmos, anteriormente indicados.

La evaluación comparativa se realiza en las mismas condiciones de ejecución y con los mismos conjuntos de gestos de referencia. Se hace uso del lenguaje de programación JavaScript para garantizar el mismo ambiente de ejecución. El estudio se limita en usar los *datasets* Hhreco (Hse & Newton, 2004), NicIcon (Niels, Willems, & Vuurpijl, 2008) y Alfabeto Utopiano (Ager, 2019).

Como resultado del estudio se espera obtener una serie de cuadros estadísticos con los datos obtenidos de la experimentación los cuales serán presentados y analizados, para identificar la plataforma más apropiada para el reconocimiento del gesto propuesto.

El propósito de esta investigación, por tanto, es aportar con los avances en el reconocimiento de gestos al poder dar a conocer cuáles son los algoritmos más eficientes, haciendo el proceso de selección más fácil, cuando se requiera incorporar interacciones naturales con gestos en aplicaciones de *software*. Esto ayudará en la creación de nuevos sistemas que permitan una interacción humano-computadora mucho más sofisticados y simples para los usuarios finales.

Con una correcta implementación de un algoritmo de reconocimiento, se puede hacer mucho más eficiente un sistema y se ahorrará tiempo y dinero en el desarrollo de este. Un enfoque especial, hacia la implementación de estos reconocedores, se evidencia en sistemas que ayuden al prototipo, en la fase de análisis y recopilación de requerimientos, ya que mostrarán en tiempo real cómo serán las interfaces y vistas que se desplegarán en la aplicación que se está desarrollando, haciendo que estas etapas del desarrollo sean mucho más precisas y respondan al gusto del usuario.

1.1.1 Objetivo General del Trabajo

Realizar un estudio comparativo entre los algoritmos !FTL, Penny Pincher y $Q, con el fin de verificar la eficiencia y el rendimiento con diferentes *datasets.*

1.1.2 Objetivos Específicos del Trabajo

- Crear un ambiente de comparación estandarizado para cada uno de los algoritmos.
- Definir los *datasets* que se van a utilizar para la implementación.
- Definir los criterios de similitud de los elementos de los *datasets*, para poder comparar con los resultados de la investigación.
- Capturar datos para cada uno de los *datasets*, de manera colectiva como individual.
- Someter a la experimentación los conjuntos de *datasets* y cada uno de sus elementos.
- Calcular la tasa, el tiempo de reconocimiento y el rango de error de los algoritmos con cada elemento del experimento.
- Utilizar herramientas estadísticas para presentar los resultados de los datos obtenidos de la comparación.

2. Capítulo II. Marco Teórico

2.1 Reconocimiento de los gestos

2.1.1 Gesto

Un gesto, dentro del contexto de reconocedores de gestos en 2D, es una secuencia de puntos equidistantes que tienen un inicio y un fin (Vanderdonckt, Magrofuoco, Burny, Pérez, & Roselli, 2017), como se muestra en la siguiente figura:

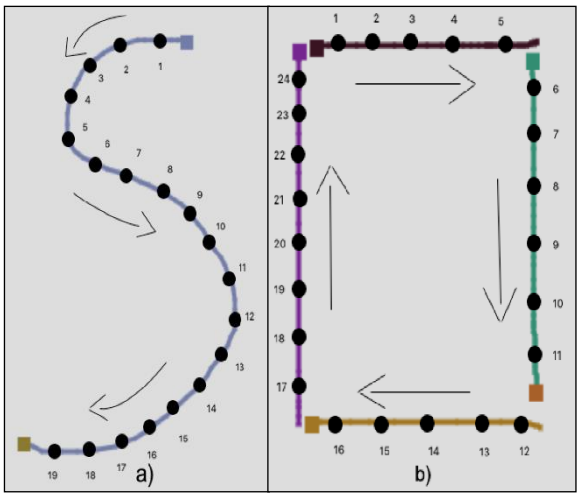


Figura 1. Ejemplos de gestos

a) Uni-stroke
b) Multi-stroke.

Existen dos tipos de gestos: Uni-stroke y Multi-stroke. Estos tipos de gestos se detallan a continuación:

- **Uni-stroke:** son gestos que no tienen ninguna interrupción en su trazo (Wobbrock, Wilson, & Li, 2007). Así como se muestra en la Figura 1, panel a).
- **Multi-stroke:** son gestos que tienen interrupción en su trazo. Estas interrupciones pueden ser asociadas con el levantamiento del dispositivo de entrada con el que se está haciendo el trazo (Vatavu, Anthony, &

Wobbrock, 2012). Un ejemplo de este tipo de trazo se presenta en el panel b) de la Figura 1. Nótese que en esta figura el gesto está compuesto por 4 *strokes*.

Un gesto también puede tener un remuestreo (*resampling*), que es transformar la figura con “n” cantidad de puntos, a una cantidad específica de puntos. Como en la Figura 1. Panel a), se puede observar que tiene una cantidad de 19 puntos, pero después del *resampling* se convierte en un trazo de 16 puntos. Por tanto, después del uso del algoritmo, la figura estaría formada por solo 16 puntos en lugar de 19.

Los gestos tienen varias propiedades como lo son:

- Traslación,
- Rotación,
- Escala y
- Dinámica del gesto

2.1.1.1 Traslación

Es el movimiento que un gesto puede tener al cambiar de posición, sin modificar su rotación (Vanderdonckt, Roselli, & Pérez, 2018). Como se ve en la Figura 2, es el mismo gesto del literal a) de la Figura 1, solo que con un desplazamiento.

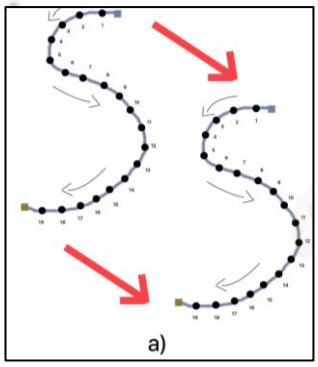


Figura 2. Ejemplo de traslación de un gesto.

2.1.1.2 Rotación

Es el movimiento que un gesto puede tener al cambiar de orientación, tomando en cuenta un eje de rotación (Vanderdonckt, Roselli, & Pérez, 2018). En la siguiente se puede observar que el panel a) de la Figura 1, ha rotado 90° respecto a su posición original.

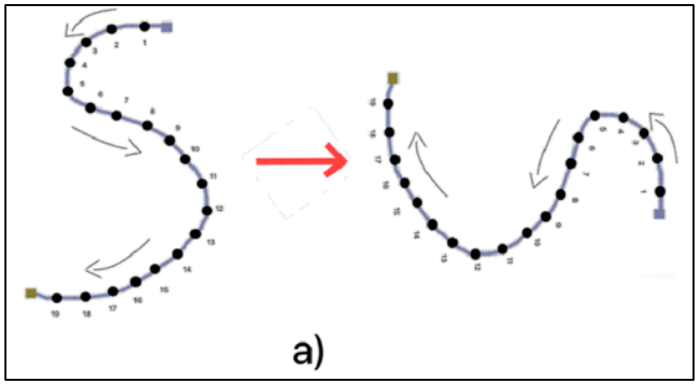


*Figura 3.*Ejemplo de rotación de un gesto.

2.1.1.3 Escala

Es el tamaño o proporción de un gesto. La escala puede variar, haciendo que el gesto sea más grande o más pequeño (Vanderdonckt, Roselli, & Pérez, 2018). En la Figura 4, se representa el mismo gesto observado anteriormente en la Figura 1, panel a) pero con una escala diferente, es decir, haciendo el gesto más pequeño.

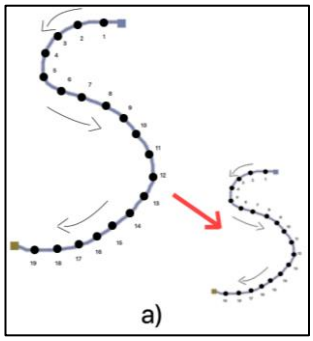


*Figura 4.*Ejemplo de escala de un gesto.

2.1.1.4 Dinámica del gesto

Es el cambio de dirección y/o de forma que realiza el usuario al hacer un gesto. Por ejemplo, en la Figura 5, podemos observar una representación del gesto de la Figura 1, panel b). Pero, su dinámica de construcción es completamente diferente. Nótese que el orden de las flechas cambia, dado que la secuencia con la que se construye el gesto es diferente. Asimismo, el número de trazos con el que se realizó el gesto es menor al número de trazos de la Figura 1, panel b). Sin embargo, siguen siendo la representación del mismo gesto, al ser ambos un rectángulo.

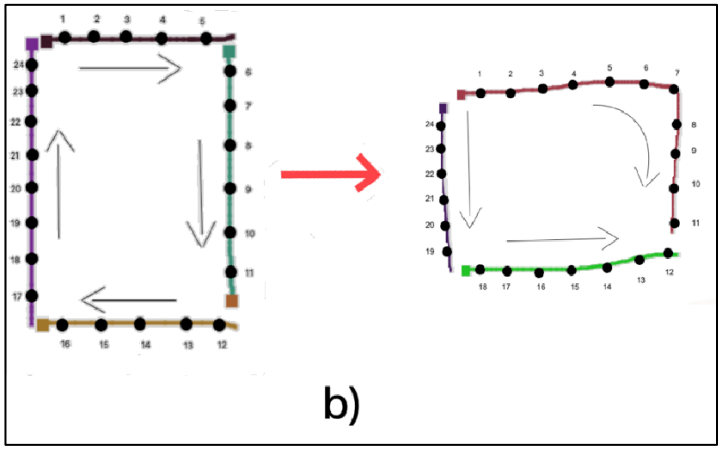


*Figura 5.*Ejemplo de dinámica de gesto.

2.2 Estado actual de los algoritmos de reconocimiento de gestos

Los reconocedores de gestos más conocidos tanto a nivel comercial, como académico son los siguientes: Rubine (Rubine, 1991), $1 (Wobbrock, Wilson, & Li, 2007), $3 (Kratz & Michael, 2010), $N (Anthony & Jacob O, 2010), ProTractor (Yang, 2010), $P (Vatavu, Anthony, & Wobbrock, 2012), $P+ (Vatavu, 2017), $V (Daiki, 2016), Penny Pincher (Eugene M. Taranta & Joseph J. LaViola, 2015), SHARK (Cao & Shumin, 2007), $Q (Vatavu, Anthony, & Wobbrock, 2018), !FTL (Vanderdonckt, Roselli, & Pérez, 2018).

Algunos de estos los algoritmos basados en gestos utilizan la “Clasificación del Vecino Más Cercano” (NNC). Este funciona utilizando un gesto candidato

realizado por el usuario, y comparándolo con cada uno de los gestos de referencia que se tenga, y calculando la distancia entre ellos para poder escoger el que tenga una menor distancia por lo tanto, una mayor similitud (Duda, Hart, & Stork, 2012). Los algoritmos que más utilizan esta clasificación son los de la familia $. Entre ellos están $1 (Wobbrock, Wilson, & Li, 2007), $P (Vatavu, Anthony, & Wobbrock, 2012), $Q (Vatavu, Anthony, & Wobbrock, 2018). (Vanderdonckt, Roselli, & Pérez, 2018).

2.2.1 Dollar P

Dollar P Point-Cloud *Recognizer* es un reconocedor de gestos 2D. Este fue creado para la elaboración de prototipos de interfaces de usuarios que sean hechas con gestos. El código de $P es una mezcla de $1 y $N, por lo que permite usar Uni-stroke y Multi-stroke. Pero, su característica principal es que no usa un conjunto de puntos ordenados, sino una nube de puntos, ignorando el número de trazos, el orden y la dirección (Anexo 1). (Vatavu, Anthony, & Wobbrock, 2012)

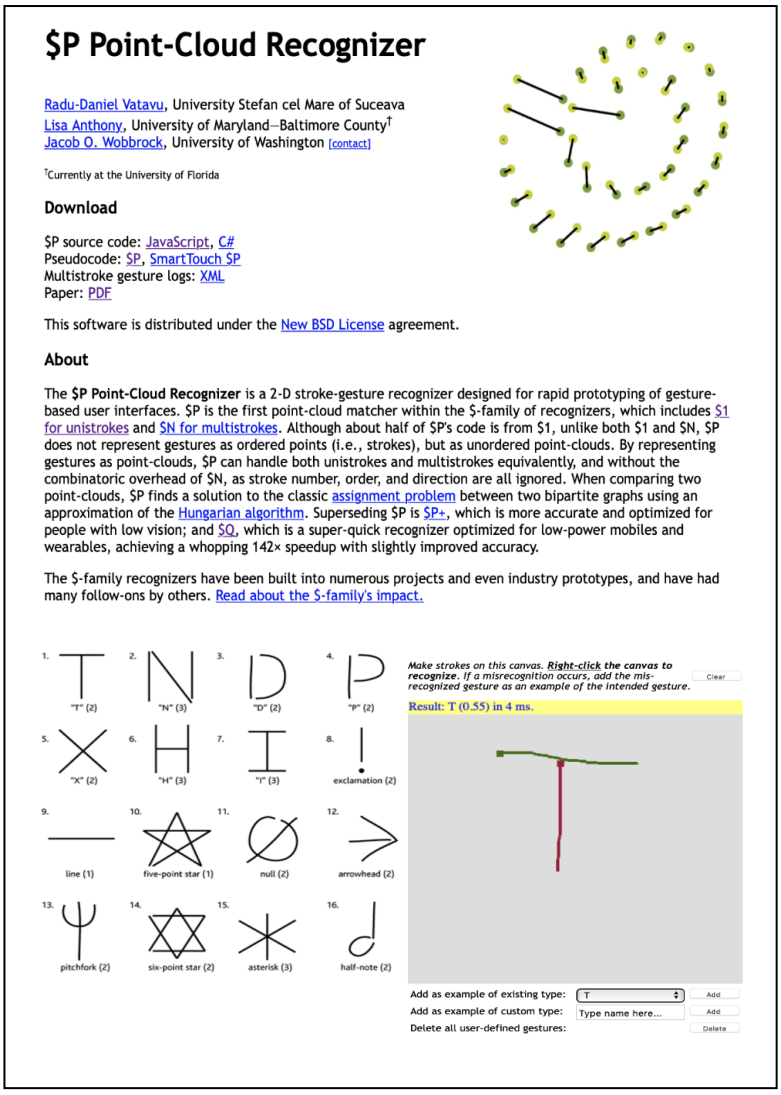


*Figura 6.*Entorno de pruebas de $P.

2.2.2 Dollar Q

$Q Super-Quick Recognizer es un reconocedor de gestos 2D, que fue creado para la elaboración de prototipos de interfaces de usuarios elaborados con gestos, especializándose en dispositivos móviles y dispositivos de bajo consumo. Este reconocedor se desarrolló en base a $P, solo que fue optimizado para tener una velocidad 142 veces mayor, con una mayor precisión. Dollar Q es el reconocedor más destacado de la familia $, actualmente (Anexo 2). (Vatavu, Anthony, & Wobbrock, 2018)

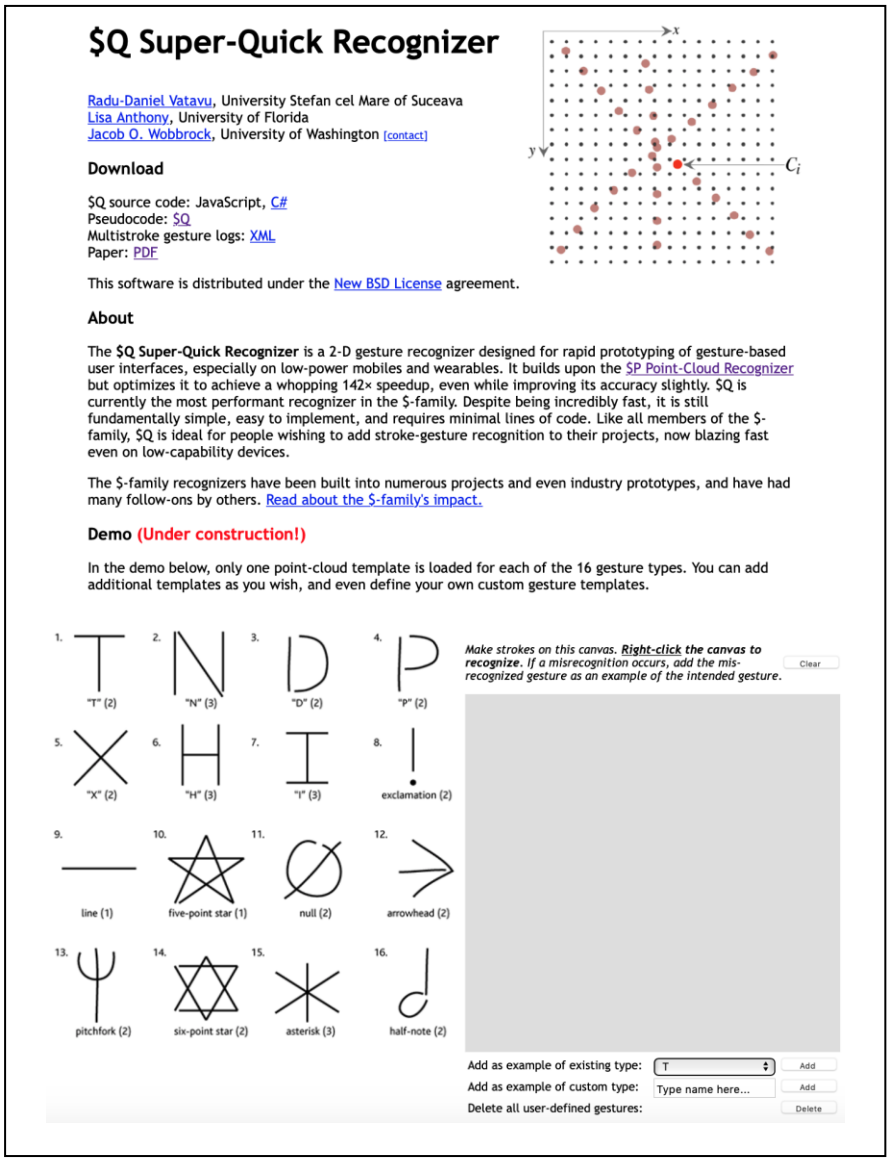


Figura 7.Entorno de pruebas de $Q.

2.2.3 Penny Pincher

El más reciente en salir fue Penny Pincher, el cual asegura que su exactitud en reconocer gestos está entre el 97.5%, 99.8%, y el 99.9%, en comparación a los otros de la familia $. Pero al centrándose más a los gestos de líneas, estos reconocedores tienen ciertas desventajas. Con ellos no se puede normalizar la cantidad de puntos con los que se puede comparar, tampoco permiten hacer un remuestreo de un gesto candidato con puntos equidistantes comparando así con

un gesto de referencia; y, por último, no toman en cuenta la escala, la rotación, ni la traslación de los gestos candidatos (Anexo 3). (Eugene M. Taranta & Joseph J. LaViola, 2015)

2.2.4 !FTL

Un reconocedor de gestos que ayuda a resolver las problemáticas anteriores es !FTL. Este gestor también utiliza NNC y utiliza algebra computacional y geométrica, creando triángulos entre los vectores de la figura para realizar el cálculo de la distancia de forma local por cada triángulo dibujado y sumando sus resultados para la distancia total.

La cantidad de código es reducida y su implementación es sencilla. Esto permite que el reconocimiento de gestos sea más rápido. Además, permite escoger sí el gesto candidato puede o no ser sensible a la orientación o al tamaño, haciéndolo mucho más aplicable en diferentes escenarios (Anexo 4). (Vanderdonckt, Roselli, & Pérez, 2018).

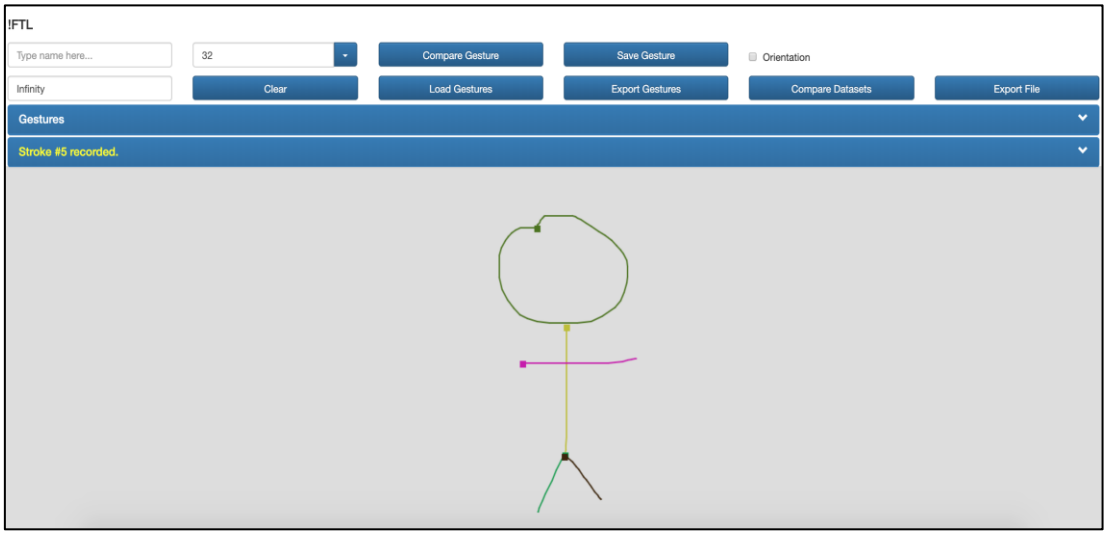


Figura 8.Entorno de pruebas de !FTL.

2.3 Ambientes de desarrollo y/o ejecución de los reconocedores de gestos

2.3.1 C#

C# es un lenguaje orientado a objetos de programación. Este lenguaje se origina en la familia de lenguajes C, es decir, C y C++. Con respecto a los programadores de JavaScript o de Java, resulta muy intuitivo, ya que se asemejan en sus sintaxis y estructura.

Actualmente C#, se enfoca mucho en tener componentes de *software* empaquetados de manera independiente y adicionalmente es fácil de entender su funcionalidad principal. Otra característica de este ambiente de desarrollo es que ayuda al control de errores con la colaboración de excepciones; la seguridad al acceso de datos según su tipo; y además, el lenguaje se encarga de recolectar elementos no utilizados para poder administrar la memoria del sistema.

C# usa un sistema de tipo unificado. Todos los tipos de datos provienen de una clase, llamada Object, de la cual heredan todos. Esto quiere decir que todos comparten operaciones y sus valores pueden ser almacenados, movilizados y utilizados coherentemente. Los tipos de datos pueden ser objetos definidos por el usuario o los tipos de datos estándares. (Microsoft, 2019). El algoritmo de $Q fue realizado con C#. Una traducción al lenguaje JavaScript fue realizada para incorporarlo en el estudio comparativo.

2.3.2 Swift

Swift es un lenguaje de programación creado por Apple, que está orientado al tanto para el desarrollo de aplicaciones de escritorio y móviles como para iOS y macOS, al igual que para la creación de Cloud Services. Por otro lado, Swift también permite que se integren bibliotecas programadas con Objective-C y usar funciones de C (Apple Inc., 2019).

Este lenguaje intenta que la sintaxis sea completamente intuitiva, basándose en el idioma natural de las personas para que el rendimiento sea igual o mejor que los lenguajes C. Asimismo, tiene herramientas para hacer que el diagnóstico de los datos tenga un nivel mayor de interacción con el usuario y el desarrollador. (Apple Inc., 2019). En este contexto, es importante mencionar que Penny Pincher usó como entorno de desarrollo Swift y una traducción al lenguaje JavaScript fue realizada para incorporarlo en el estudio comparativo.

2.3.3 JavaScript

JavaScript es un lenguaje de programación orientado a objetos, interpretado y liviano. Su mayor uso es en la programación web, ya que se ejecuta del lado del cliente, por lo que puede programar y diseñar la página web mediante eventos, pero se puede utilizar en diferentes entornos de desarrollo.

JavaScript ayuda a hacer una página web que tenga interacción con el usuario y un archivo HTML dinámico, y no solo una página estática de HTML. (Mozilla and individual contributors, 2019). Los algoritmos !FTL (incluyendo sus versiones LSD y NLSD) y $P están desarrollados en este lenguaje de programación. Por ello, se va a tomar como lenguaje estándar a JavaScript para convertir los otros algoritmos a este lenguaje.

2.4 Descripción del *dataset* y los gestos de referencia

2.4.1 Hhreco

Hhreco es una biblioteca de software que brinda un grupo de símbolos y un programa en Java para realizar el reconocimiento tanto de estos símbolos, como de gestos Multi-stroke. (Hse & Newton, 2004)

El contenido del *dataset* de prueba es recopilado en 19 archivos, por cada usuario participante. El archivo de datos de cada usuario contiene al menos 30 ejemplos por clase de símbolo. Actualmente, hay un total de 7791 ejemplos de

movimientos múltiples en general, y al menos 575 ejemplos por símbolo (Hse & Newton, 2004).

El conjunto de símbolos consta de las siguientes formas:

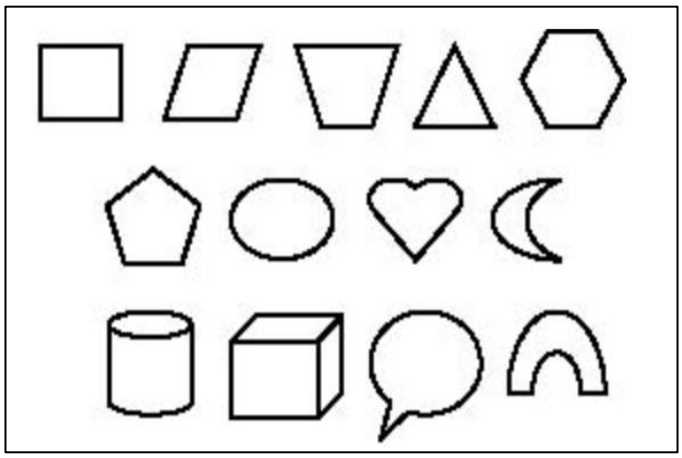


Figura 9.Dataset Hhreco.

La Tabla 1 muestra la caracterización de diferentes gestos del *dataset*, los cuales se estiman que tienen un porcentaje de semejanza alto. El porcentaje de semejanza se obtiene aplicando la siguiente ecuación:

$$Similarity\ Score = \frac{similar\ representations}{total\ representations}$$

Tabla 1.

Clasificación del dataset Hhreco.

	Characterization 1	Characterization 2	Characterization 3	Characterization 4	Characterization 5	Characterization 6
	  	 	 	 	 	 
Total Representations	3	2	2	2	2	2
Similarity Score	23.08%	15.38%	15.38%	15.38%	15.38%	15.38%

En la Tabla 1 se tienen seis tipos de caracterizaciones que abarcan a todos las figuras del *dataset*. En la primera columna, por ejemplo, se categorizaron todas aquellas representaciones que son cuadrados o cuadriláteros.

Este conjunto de figuras tiene un nivel de dificultad para ser reconocidas porque al realizar el *resampling* de la figura, es probable que todas sean consideradas como cualquiera de esta caracterización. Como se observa en la Figura 10, un paralelogramo después del *resampling* puede ser considerado un cuadrado, un hexágono puede ser un círculo, y un corazón puede ser un triángulo.

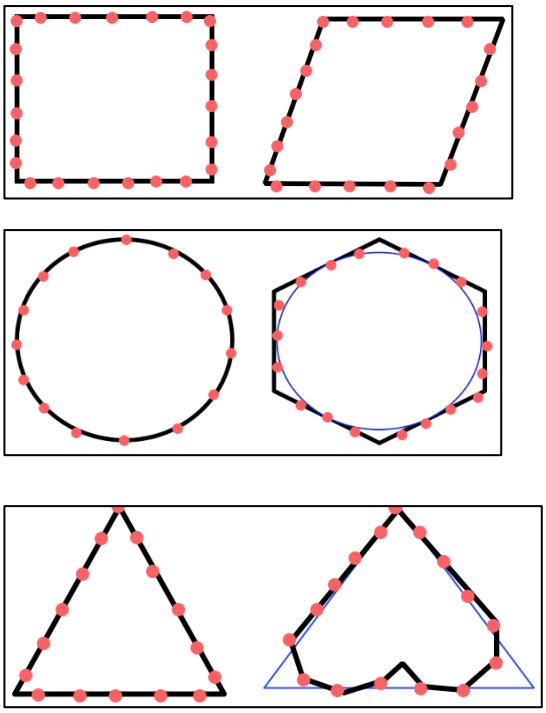


Figura 10.Resampling de gestos de Hhreco.

Teniendo en cuenta ello, en la Tabla 2 se puede ver el resultado de la categorización de los gestos después del *resampling*.

Tabla 2.

Clasificación del dataset Hhreco con resampling.

	Characterization 1	Characterization 2	Characterization 3	Characterization 4
				
				
				
				
				
Total Representations	5	4	2	2
Similarity Score	38.46%	30.77%	15.38%	15.38%

Como se puede observar en la columna denotada por la “Caracterización 1” de la Tabla 2, la estimación de similitud es de 38.46% para un número de 5 gestos. Dada la complejidad de las figuras y cantidad de figuras, se estimaría que los algoritmos de reconocimiento tendrían un porcentaje de error de hasta el 38.46% para este primer grupo de figuras, el 30.77% para la segunda columna, y así sucesivamente.

2.4.2 NicIcon

Este *dataset* es un conjunto de 14 íconos que son de gran utilidad en el ámbito de manejo de crisis e incidentes. Estos gestos fueron diseñados para que se tenga una relación entre el dibujo y lo que representa o lo correspondiente en símbolos. Con ello, se hace mucho más fácil para el usuario memorizar el *dataset* y que el sistema también pueda distinguirlo de manera sencilla. (Niels, Willems, & Vuurpijl, 2008)

El *dataset* fue realizado por 32 usuarios, los cuales dibujaron 770 gestos cada uno, pero algunos usuarios parecen que se saltaron algunos gestos. Esto da un

total aproximado de 24.640 gestos (Niels, Willems, & Vuurpijl, 2008). Los gestos que se diseñaron son los que se muestran en la Figura 11:

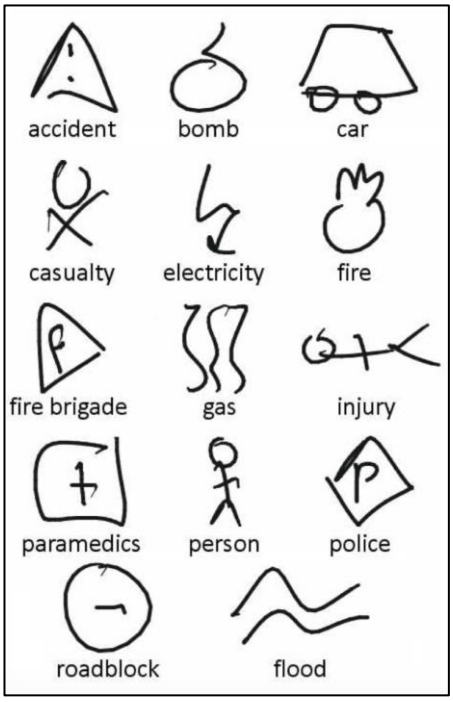


Figura 11.Dataset de NicIcon.

Para medir el nivel de dificultad del *dataset*, primero se organiza a los gestos en categorías que los abarquen a todos.

Tabla 3.

Clasificación del dataset NicIcon.

	Characterization 1	Characterization 2	Characterization 3	Characterization 4	Characterization 5	Characterization 6
						
						
						
Total Representations	2	3	3	3	2	1
Similarity Score	14.29%	21.43%	21.43%	21.43%	14.29%	7.14%

La Tabla 3 presenta la caracterización de los gestos similares para el *dataset* NicIcon. Estos símbolos serán los que posiblemente produzcan errores al

realizar la experimentación, ya que puede que sean confundidos con cualquiera de su caracterización, ya sea por la figura final de su *resampling* o porque sea la misa figura, pero rotada.

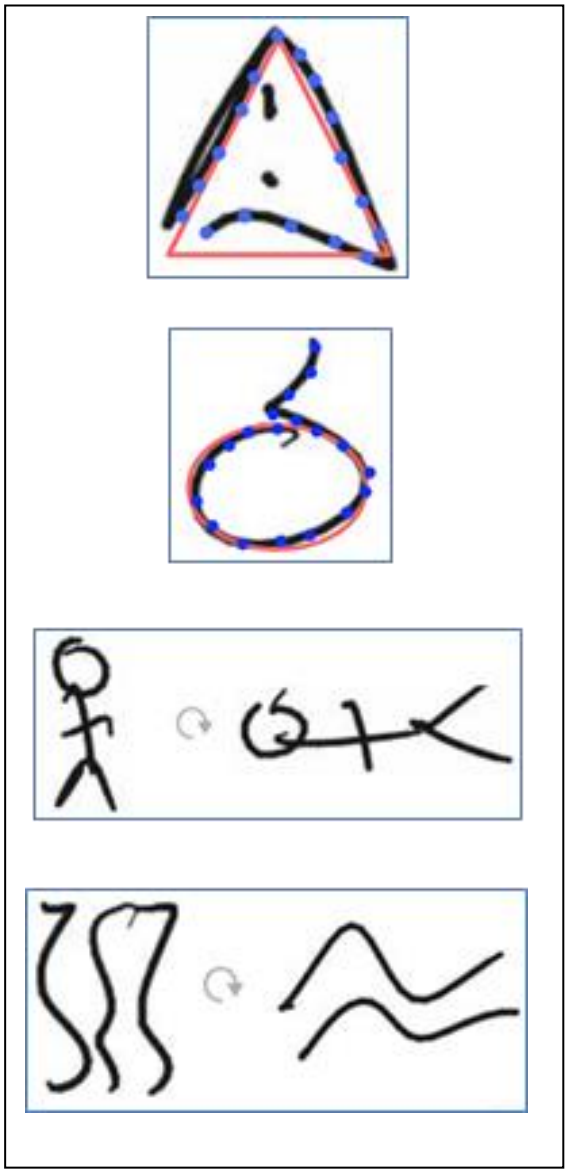


Figura 12.*Resampling* de gestos de NicIcon.

Tabla 4.

Clasificación del dataset Hhreco con resampling.

	Characterization 1	Characterization 2	Characterization 3	Characterization 4	Characterization 5
					
					
					
Total Representations	2	3	3	3	3
Similarity Score	14.29%	21.43%	21.43%	21.43%	21.43%

Se puede apreciar que en la Tabla 4, en la columna denotada por la "Caracterización 1", la estimación de similitud es de 14.29% para un número de 2 gestos. Debido a la complejidad de las figuras y cantidad de figuras, se estimaría que los algoritmos de reconocimiento están en la capacidad de acertar un 85.71% para el mencionado primer grupo de figuras y un 78.57% a partir de la segunda columna.

2.4.3 Vocabulario Utopiano

Este alfabeto aparece en el libro Utopía de St. Thomas More, que fue escrito en latín y publicado en 1516 (Ager, 2019). El *dataset* contiene 4400 gestos realizados por 20 usuarios. Los usuarios dibujaron cada letra del vocabulario 10 veces. Los gestos fueron realizados con uno o varios trazos. El *dataset* está clasificado por usuario, por lo que cada usuario tiene 220 gestos. Los gestos del *dataset* son los siguientes:

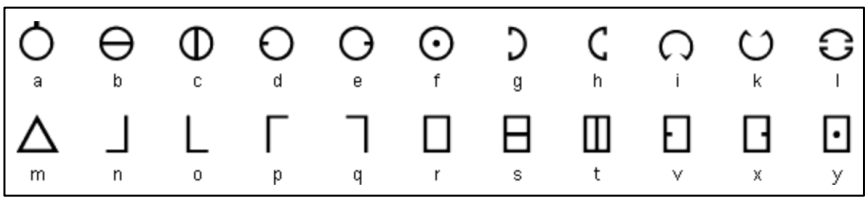


Figura 13.Dataset Vocabulario Utopiano.

La Tabla 5 presenta la caracterización de los gestos similares para el *dataset* Utopiano. Estas letras son las que se prevé tendrán mayores problemas para ser reconocidas por los algoritmos al realizar la experimentación, ya que puede que se confundan con cualquiera de su caracterización debido al resultado del *resampling*.

Tabla 5.

Clasificación del dataset Utopiano.

	Characterization 1	Characterization 2	Characterization 3	Characterization 4	Characterization 5
					
					
					
					
					
					
Total Representations	6	5	1	4	6
Similarity Score	27.27%	22.73%	4.55%	18.18%	27.27%

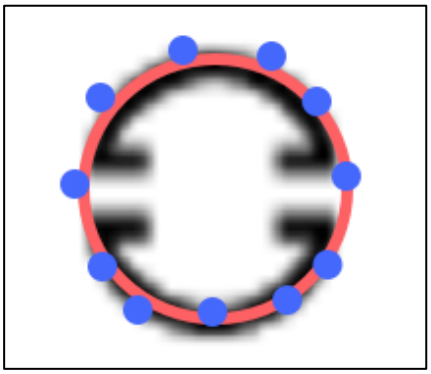


Figura 14.Resampling de gestos de Utopiano.

Tabla 6.

Clasificación del dataset Utopiano con resampling.

	Characterization 1	Characterization 2	Characterization 3	Characterization 4	Characterization 5
					
					
					
					
					
					
					
Total Representations	7	4	4	1	6
Similarity Score	31.82%	18.18%	18.18%	4.55%	27.27%

Como se observa en la Tabla 6, en columna denotada por la “Caracterización 1”, la estimación de similitud es de 27.27% para un número de 6 gestos. Por la complejidad de las figuras y cantidad de figuras, se estimaría que los algoritmos de reconocimiento se podrían equivocar hasta un 27.27% para la caracterización uno, un 30.77% para la segunda caracterización, y así sucesivamente.

3. Capítulo III. Métodos para evaluar el rendimiento de los “Reconocedores de Gestos”

3.1 Tasa de reconocimiento

La tasa de renacimiento es un indicador que permite calcular el porcentaje de efectividad de un algoritmo cuando es sometido a una evaluación de gestos de un *dataset* candidato con respecto a los gestos de referencia previamente almacenados en el sistema. Esto quiere decir que, en la comparación, cada gesto candidato será evaluado con los algoritmos de reconocimiento contra los gestos de referencia que estén previamente registrados en el sistema. El resultado se indicará mostrando al gesto más cercano en distancia y el tiempo

que se demoró en comparar. Después, se suman la cantidad de veces en que el algoritmo reconoció a los gestos candidatos de manera acertada. A esta suma se le divide sobre la cantidad de gestos comparados y con ello, se obtiene la tasa de reconocimiento.

3.2 Tasa de error

La tasa de error es un factor que permite calcular el porcentaje de deficiencia de un algoritmo específico cuando es sometido a la evaluación de los gestos de un *dataset* candidato con respecto a los gestos de referencia previamente almacenados. Esto quiere decir que, en la comparación, cada gesto candidato será evaluado con los algoritmos de reconocimiento contra los gestos que estén en el sistema y se indicará el resultado, mostrando al gesto más cercano en distancia y el tiempo que se demoró en comparar. Después, se suman la cantidad de veces que algoritmo reconoció a los gestos candidatos de manera incorrecta. Esta suma será dividida por la cantidad de gestos comparados. Con ello, se obtendrá la tasa de error, es decir, las respuestas incorrectas.

3.3 Tiempo de reconocimiento

El tiempo de reconocimiento es un indicador que captura el tiempo en el que se ha trabajado un algoritmo cuando este es sometido a una evaluación, en el cual se compara un *dataset* candidato, con los *datasets* almacenados en el sistema, usando el procedimiento del algoritmo para indicar el resultado. El tiempo de reconocimiento sirve para ver el rendimiento de cada algoritmo evaluado. Este factor es un valor promedio ya que el tiempo se mide desde que comienza la evaluación de un gesto del *dataset* candidato y finaliza con el resultado de ese gesto. Este tiempo se va sumando por cada gesto y al final se saca un promedio. El resultado indicará el tiempo medido en milisegundos.

3.4 Reconocimiento de acuerdo con la intervención del usuario

3.4.1 User independent

El reconocimiento que es independiente del usuario es aquel en donde el experimento se realiza con *datasets* previamente obtenidas y publicadas. Para la creación de estos *datasets*, se invita a un grupo de usuarios que tienen que dibujar gestos. Los gestos representados por cada usuario son parte del *dataset*.

Para el reconocimiento se usa un proceso semiautomatizado, con el que se comparan los gestos realizados por un usuario participante, contra el *dataset* previamente obtenido. Gracias a eso, se puede evaluar y comparar los resultados entre usuarios.

3.4.2 User dependent

El reconocimiento que depende del usuario es aquel que no tiene gestos previamente obtenidos. Es decir, los gestos que hace el usuario son sometidos a evaluación en ese mismo instante. Por lo tanto, este tipo de reconocimiento solo permite evaluar el rendimiento de un usuario y las características de los gestos, y no la comparación de varios usuarios.

3.4.3 Discusión

Para efectos de esta investigación, se centrará en el tipo de reconocimiento independiente del usuario. Esto se debe a que existen una gran cantidad de *datasets* que se pueden reutilizar, teniendo la ventaja de recuperar gestos públicos que ya se encuentren previamente desarrollados.

El autor Adrian Delaye en el año 2011,documentó una gran cantidad de *datasets* disponibles entre los cuales se encuentran el *dataset* de NicIcon y Hhreco. Una de las características que presentan estos *datasets*, es que poseen una cantidad significativa de gestos previamente realizados. Esto ayuda reducir el tiempo en

la captura de los gestos y de esta manera, disminuir el tiempo de evaluación de los algoritmos. Esto se debe a que no se necesita la intervención de varios usuarios voluntarios que realicen cada uno de los gestos. En el caso de este trabajo, el reúso de *datasets* existentes, permitió cumplir con los plazos establecidos.

A pesar de que se propone el *dataset* de preferencia Utopiano, la estrategia sigue siendo independiente del usuario, ya que no se compara el rendimiento de cada usuario que participó en el *dataset*, ni las características del gesto que se dibujaron. La evaluación tampoco es en tiempo real, ya que se hace en un posterior proceso semiautomatizado para determinar los resultados.

4. Capítulo IV. Propuesta para la evaluación de los "Reconocedores de Gestos"

4.1 Descripción del experimento

4.1.1 Materiales

Para realizar el experimento se hizo uso de un conjunto de *datasets* y un ambiente de ejecución en común. La tabla 7, muestra los datasets utilizados en el experimento. Nótese por cada dataset se visualiza una cantidad significativa de gestos.

Tabla 7.
Datasets.

Dataset	Representaciones	Usuarios	Repeticiones	Total Gestos
NicIcon	14	32	55	14x32x55= 24.640
Hhreco	13	19	30	13x19x30= 7.410
Alfabeto Utopiano	22	20	10	22x20x10= 4.400

Para el ambiente de ejecución, se hizo uso del lenguaje de programación JavaScript con el entorno de desarrollo de !FTL.

En el experimento se utilizaron los siguientes algoritmos de reconocimiento de gestos:

- !FTL (LSD y NLSD)
- $1
- $P
- $Q
- Penny Pincher

Las evaluaciones se realizaron haciendo uso de un computador MacBook con las siguientes características:

- **Hardware**
  - **Procesador:** 1.1 GHz Intel Core M
  - **Memoria RAM:** 8 GB
  - **SSD:** 251 GB
  - **Tarjeta Gráfica:** Intel HD Graphics 5300 1536 MB
- **Software**
  - **Versión del sistema operativo:** macOS 10.14.4 (18E226)
  - **Versión del kernel:** Darwin 18.5.0

4.1.2 Métodos

Se definieron condiciones similares para todos los algoritmos, a ser ejecutados bajo las mismas características.

Para los algoritmos estudiados, el método consintió en evaluar la tasa de error y tasa de rendimiento, así como el tiempo de ejecución de cada uno de estos algoritmos.

4.1.3 Procedimiento

Primero, se escogió un ambiente de ejecución similar para el proceso de comparación. Por tales motivos, se implementaron en el lenguaje JavaScript los algoritmos Penny Pincher y $Q. Luego, estos algoritmos fueron integrados en el ambiente de prueba de !FTL. En paralelo, se definió el *dataset* Utopiano. Luego de ello, se creó el *dataset* Utopiano, con la participación voluntaria de 20 personas. Una vez creada las condiciones similares para cada uno de los algoritmos se realizó la experimentación con cada *dataset*.

Por cada *dataset* el procedimiento de experimentación efectuado fue el siguiente:

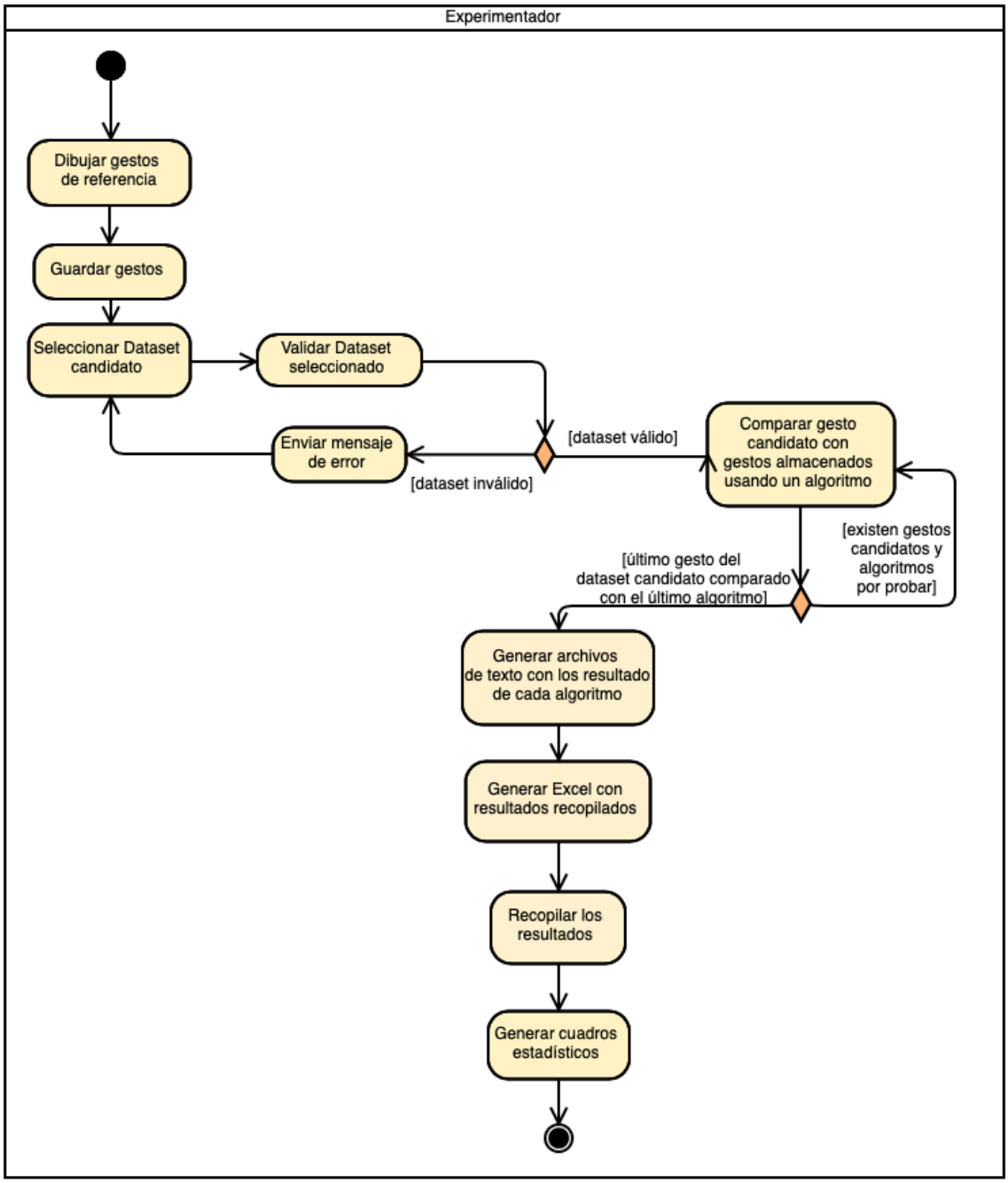


Figura 15. Diagrama de actividades.

Como se puede notar en la Figura 15, el procedimiento comienza con el registro del *dataset* de referencia. Para cada *dataset* el número de gestos de referencia fue de diez, por cada representación.

Posteriormente, por cada uno de usuarios del *dataset*, se evalúa uno a uno los gestos candidatos con respecto a los gestos de referencia previamente

almacenados. La Figura 16, muestra un ejemplo de la estructura resultante del procedimiento de evaluación de un usuario con una representación específica, en este caso, accident, dentro del *dataset* de NicIcon, y usando un algoritmo de reconocimiento.

```
Date: accident_5/6/2019, 8:43:23 PM
!FTL(LSD)
Records: 38
(NicIcon) File name=accident.xml, -, -, -

Iteration: undefined Gesture evaluated=accident, Nro. Gestures=30, Number of Points:32, Threshold: Infinity, -

Recognized gesture name, Nro. points, distance/score, time(ms)
bomb, 32, 284.5015720508556, 10.224999998230487
roadblock, 32, 178.18836256893823, 0.4100000004982576
bomb, 32, 513633199397455.2, 5.644999997457489
accident, 32, 353.3769309208858, 0.630000002332963
electricity, 32, 508470925671045.94, 0.6999999968102202
electricity, 32, 216.01870621207553, 2.7849999968893826
accident, 32, 181.43678044138935, 0.6800000047078356
bomb, 32, 149.8849359597491, 0.30999999574851245
injury, 32, 472151573837416.06, 0.5850000015925616
accident, 32, 224.3029136826871, 0.6500000017695129
injury, 32, 53.096252574212954, 0.5450000027194619
roadblock, 32, 76.71075671833779, 0.6450000000186265
casualty, 32, 55.200126065585664, 0.3050000013317913
roadblock, 32, 147.02317242480123, 0.7550000010523945
fire, 32, 113.80746321713063, 6.045000001904555
casualty, 32, 113.37104722706194, 0.6899999934248626
casualty, 32, 55.20012606558607, 0.5149999997811392
casualty, 32, 76.2558781239167, 0.5799999998416752
bomb, 32, 114.33573849117975, 0.3199999992502853
bomb, 32, 112.89799668221558, 0.6599999979371205
bomb, 32, 196.91199385007113, 0.455000001238659
roadblock, 32, 267.94330483940195, 0.4950000001117587
roadblock, 32, 508470925670927.44, 0.8050000034272671
electricity, 32, 122.11623422915763, 5.664999996894039
roadblock, 32, 182.45067112445577, 0.7699999988544732
bomb, 32, 80.52783731008265, 0.3849999993108213
bomb, 32, 161.00356223989968, 0.330000002519227 56
bomb, 32, 176.44771577904868, 0.6050000010291114
bomb, 32, 173.6770991521764, 0.350000002072192 73
roadblock, 32, 164.56657158739165, 0.32500000088475645

Distance, 253.0388750149874, -, -, -
Time, 0.6533333362701038, -, -, -
Samples, 30, -, -, -
Recognize samples, 3, -, -, -
% Recognition, 10, -, -, -
```

Figura 16.Resultados Usuario0 de accident con !FTL (LSD).

En este caso, se evalúa la representación accident con !FTL (LSD). En el resultado se observa que al inicio se presentan datos como la fecha, el algoritmo que se utiliza para la experimentación y el archivo que se seleccionó para comparar. Luego, se especifica la representación a evaluar, el número de repeticiones que realizó el usuario, el número de puntos con el que se hará el *resampling* y el *threshold*.

El listado contiene cuatro columnas que resumen los siguientes resultados: el nombre del gesto reconocido, la cantidad de puntos evaluados, el porcentaje de evaluación, y el tiempo en reconocimiento. En el caso de !FTL usando la

distancia LSD y LNSD, indica que, entre más se acerque el valor de evaluación al infinito, el algoritmo tiene más precisión (Luzzi & Roselli, 2018).

Al pie del archivo se encuentran los totales de la evaluación de los 30 gestos del usuario. Podemos ver que dentro de los totales se tiene un promedio del tiempo del reconocimiento de 0.6533 ms, la tasa de reconocimiento de 3 y el porcentaje de reconocimiento de 10%.

accident_5_6_2019, 8_43_23 PM_TOTALES

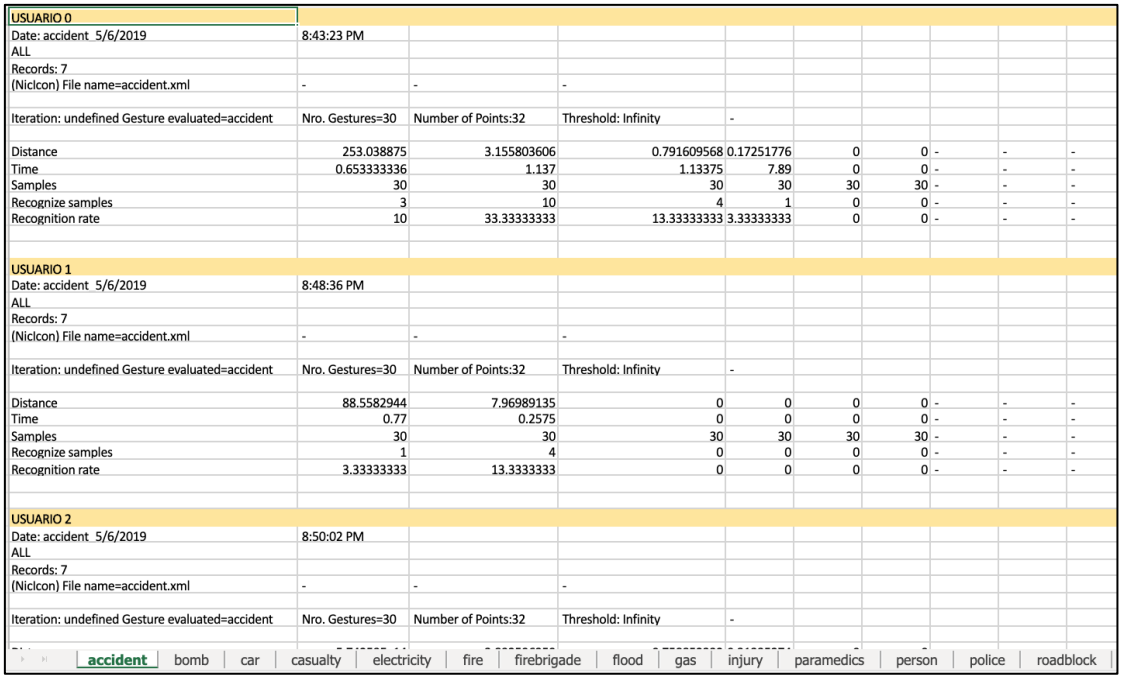
Date: accident_5/6/2019	**8:43:23 PM**					
ALL						
Records: 7						
(NicIcon) File name=accident.xml	-	-	-			
Iteration: undefined Gesture evaluated=accident	Nro. Gestures=30	Number of Points:32	Threshold: Infinity	-		
Distance	253.0388750149874	3.1558036055701346	0.7916095676960685	0.1725177562351916	0	0
Time	0.6533333362701038	1.1370000000228173	1.133750000008149	7.890000000828877	0	0
Samples	30	30	30	30	30	30
Recognize samples	3	10	4	1	0	0
Recognition rate	10	33.333333333333336	13.333333333333334	3.3333333333333335	0	0

*Figura 17.*Resultados de la representación accident del usuario.

En la figura 17, se muestra el resultado de una representación realizada por un usuario, comparando los datos obtenidos por cada algoritmo evaluado. En la primera columna se encuentran el nombre del archivo usado, el gesto evaluado, y los parámetros de comparación para cada algoritmo: distancia, tiempo, cantidad de gestos evaluados, cantidad de gestos reconocidos correctamente y porcentaje de reconocimiento. En la siguiente columna, lo primero que muestra es el número de repeticiones del gesto, y los resultados de !FTL(LSD). En este caso, se tiene una distancia de 253.038, un tiempo promedio de 0.653 segundos, 30 gestos evaluados, 3 gestos reconocidos y un porcentaje de reconocimiento de 10%. En la tercera columna se muestra el número de puntos y los resultados de !FTL(NLSD). La siguiente columna indica el *threshold*, que es infinito, y los resultados de usar el algoritmo $1. Los resultados de evaluar con $P se muestran

en la cuarta columna. En la siguiente se encuentra $Q y en la última columna Penny Pincher.

Cada resultado fue almacenado en un archivo con extensión .csv.



USUARIO 0									
Date: accident 5/6/2019	8:43:23 PM								
ALL									
Records: 7									
(NicIcon) File name=accident.xml	-	-	-						
Iteration: undefined Gesture evaluated=accident	Nro. Gestures=30	Number of Points:32	Threshold: Infinity	-					
Distance	253.038875	3.155803606	0.791609568	0.17251776	0	0	-	-	-
Time	0.653333336	1.137	1.13375	7.89	0	0	-	-	-
Samples	30	30	30	30	30	30	-	-	-
Recognize samples	3	10	4	1	0	0	-	-	-
Recognition rate	10	33.33333333	13.33333333	3.33333333	0	0	-	-	-
USUARIO 1									
Date: accident 5/6/2019	8:48:36 PM								
ALL									
Records: 7									
(NicIcon) File name=accident.xml	-	-	-						
Iteration: undefined Gesture evaluated=accident	Nro. Gestures=30	Number of Points:32	Threshold: Infinity	-					
Distance	88.5582944	7.96989135	0	0	0	0	-	-	-
Time	0.77	0.2575	0	0	0	0	-	-	-
Samples	30	30	30	30	30	30	-	-	-
Recognize samples	1	4	0	0	0	0	-	-	-
Recognition rate	3.33333333	13.3333333	0	0	0	0	-	-	-
USUARIO 2									
Date: accident 5/6/2019	8:50:02 PM								
ALL									
Records: 7									
(NicIcon) File name=accident.xml	-	-	-						
Iteration: undefined Gesture evaluated=accident	Nro. Gestures=30	Number of Points:32	Threshold: Infinity	-					

accident | bomb | car | casualty | electricity | fire | firebrigade | flood | gas | injury | paramedics | person | police | roadblock

*Figura 18.*Archivo CSV con los resultados.

Una vez obtenidos los resultados por cada *dataset*, se procedió a contabilizar los resultados con el fin de hacer los cuadros estadísticos.

4.2 Descripción del ambiente de ejecución

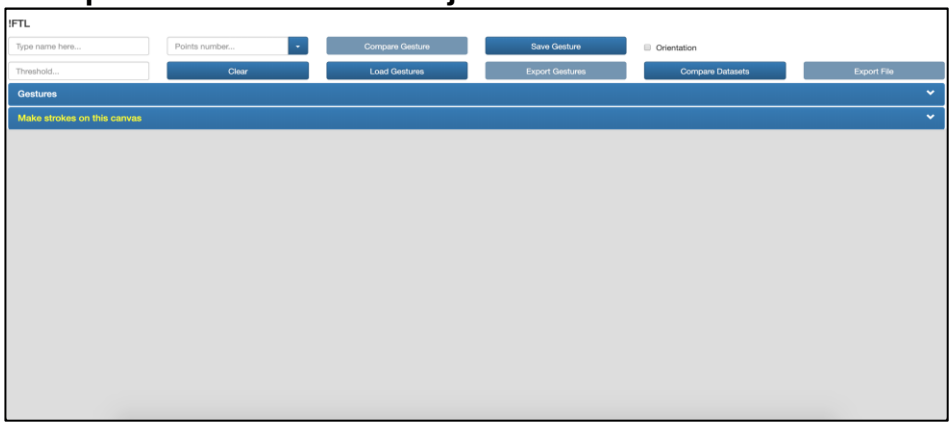


Figura 19.Pantalla inicial del ambiente de ejecución de !FTL.

En la Figura 19 se muestra la pantalla principal del ambiente de ejecución de !FTL. Este ambiente fue desarrollado en JavaScript.

En la pantalla se ve que se pueden realizar las siguientes funcionalidades: comparar gesto, guardar gesto, limpiar área de trabajo, cargar gestos, exportar gestos, comparar *datasets*, exportar archivos con los resultados.

Para poder guardar un gesto candidato, lo primero que se hace es dibujar el gesto dentro del área de trabajo, colocar un nombre, escoger el número de puntos para el *resampling*, seleccionar el *threshold* y dar clic en el botón para guardar el gesto. Lo siguiente es comparar el gesto, para ello se realiza otro gesto con el cual se va a someter a la experimentación, y dar clic sobre el comando *Compare Gestures*. Con ello se desplegarán los resultados por cada algoritmo, mostrando el nombre del gesto resultante, el tiempo utilizado para obtener el resultado y la distancia. Como se observa en la siguiente figura:

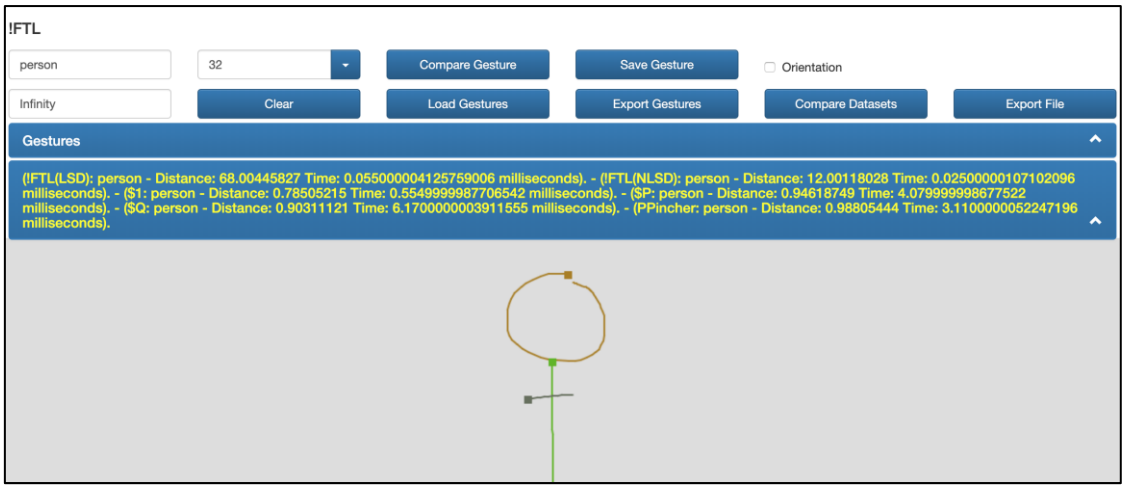


Figura 20.Ejemplo de la funcionalidad de guardar un gesto, compararlo y mostrar resultados.

En la figura 21 se puede ver que el proceso anterior produce el *resampling* de la figura candidata y de la figura comparada, lo cual se puede observar seleccionando la opción de *Gestures* en el área de trabajo.

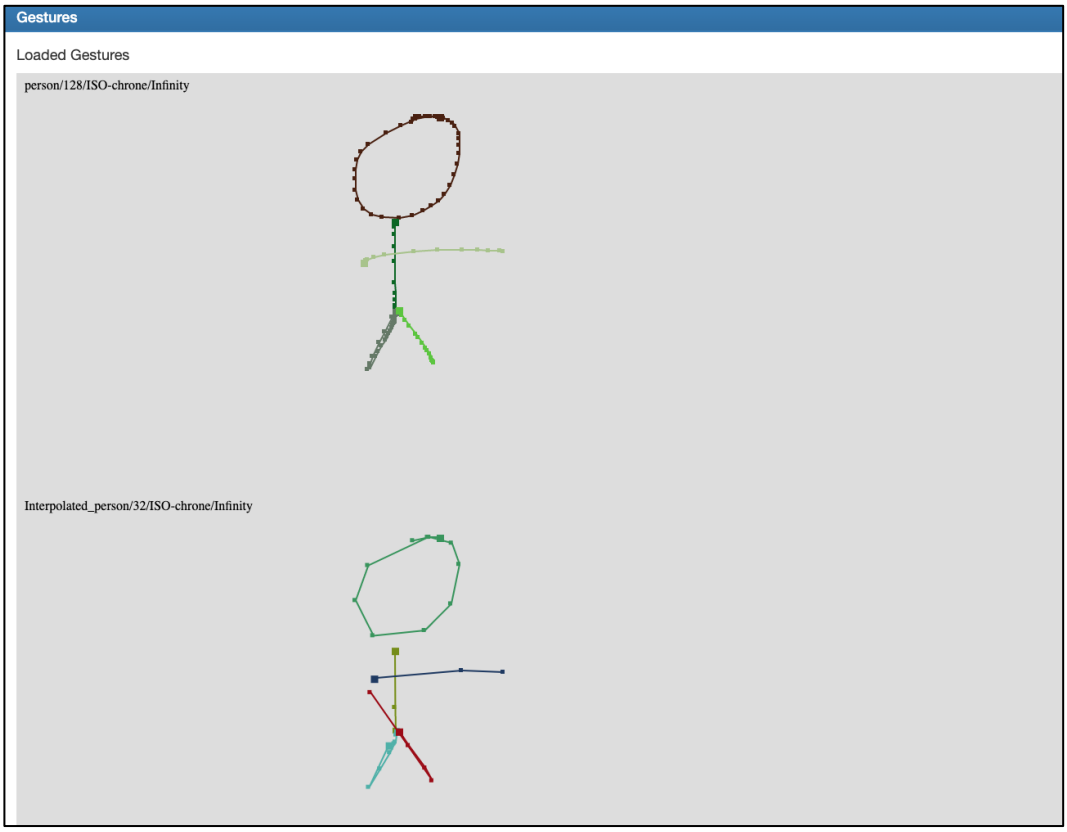
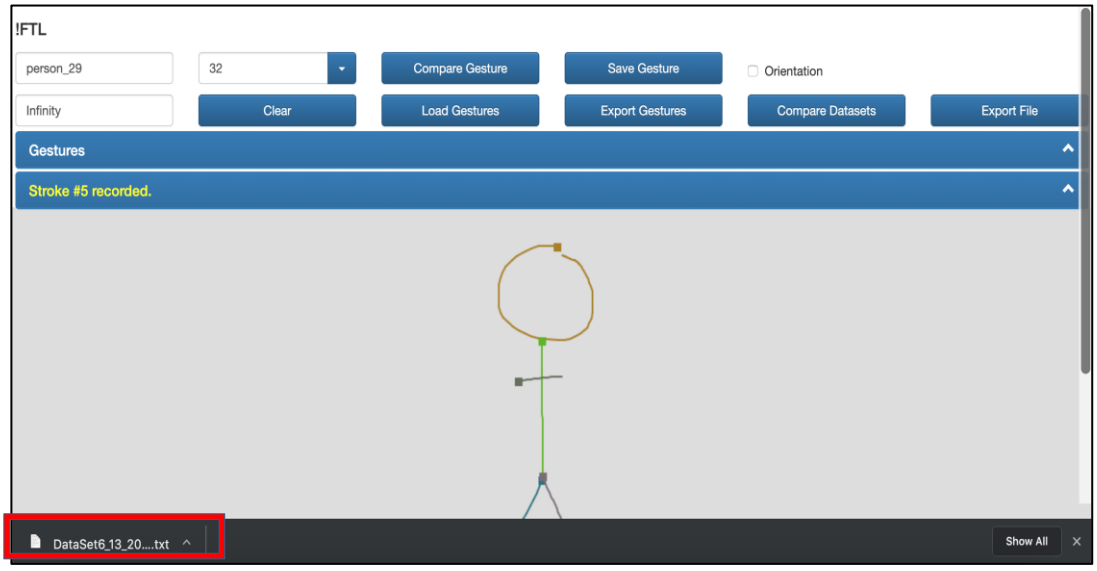


Figura 21.Resampling de una figura candidata.

Además, las figuras que fueron guardadas se pueden exportar e importar con los botones de *Export Gestures* y *Load Gestures.*



[[{"Name":"person","Points":[{"X":518,"Y":53,"ID":1},{"X":503.51612903225805,"Y":53.096774193548384,"ID":1},{"X":430.4193548387097,"Y":87.5806451612903,"ID":1},{"X":415,"Y":130.3548387096774,"ID":1},{"X":436.6774193548387,"Y":173.38709677419357,"ID":1},{"X":499.61290322580646,"Y":167.03225806451613,"ID":1},{"X":531.9677419354838,"Y":134.48387096774192,"ID":1},{"X":542,"Y":86.19354838709677,"ID":1},{"X":532.7096774193549,"Y":60.225806451612904,"ID":1},{"X":505.258064516129,"Y":53.00000000000001,"ID":1},{"X":485.3548387096774,"Y":57.29032258064515,"ID":1},{"X":463,"Y":191.19354838709677,"ID":2},{"X":463.38709677419354,"Y":261.0322580645161,"ID":2},{"X":464,"Y":288.8387096774194,"ID":2},{"X":464,"Y":291,"ID":2},{"X":455.64516129032256,"Y":306.7096774193549,"ID":3},{"X":431.45161290322585,"Y":358.61290322580646,"ID":3},{"X":445.1935483870967,"Y":336.1612903225806,"ID":3},{"X":457.1612903225806,"Y":316.258064516129,"ID":3},{"X":462.61290322580646,"Y":305.38709677419354,"ID":3},{"X":464,"Y":303.35483870967744,"ID":3},{"X":465,"Y":294.64516129032256,"ID":3},{"X":466.00000000000006,"Y":291,"ID":3},{"X":467.741935483871,"Y":289.258064516129,"ID":3},{"X":468,"Y":289.7741935483871,"ID":4},{"X":480.0322580645161,"Y":307.64516129032256,"ID":4},{"X":500.35483870967744,"Y":335.03225806451616,"ID":4},{"X":508.8709677419355,"Y":350.6129032258064,"ID":4},{"X":433.3225806451613,"Y":242.9032258064516,"ID":4},{"X":437.4838709677419,"Y":225.19354838709677,"ID":5},{"X":545.0322580645161,"Y":216,"ID":5},{"X":596,"Y":218,"ID":5}],"Format":"ISO-chrone","Threshold":"Infinity"}]]

Figura 22. Ejemplo de resultado de exportar gestos guardados.

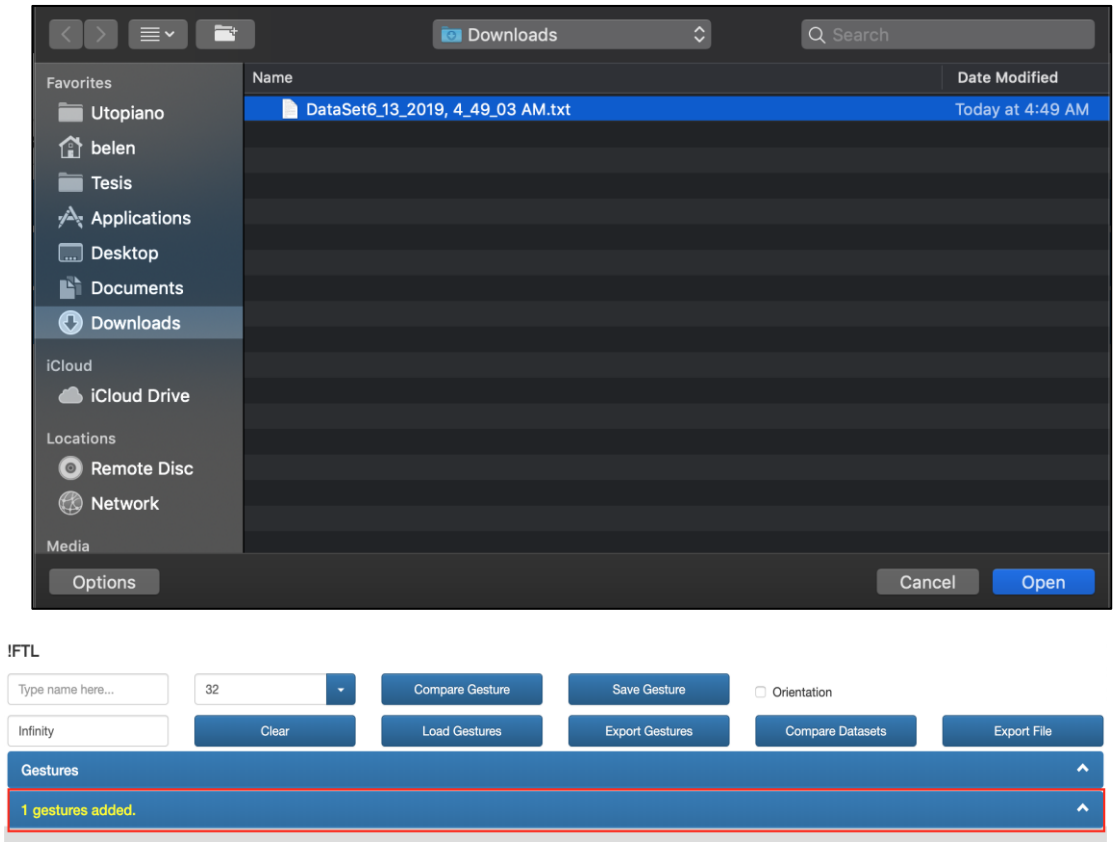


Figura 23. Ejemplo de resultado de cargar gestos guardados.

Por último, se pueden comparar los gestos guardados con el *dataset* de Hhreco, NicIcon y el Utopiano. Para ello, se da escoge *Compare Datasets*, se selecciona el *dataset* con el que quiere comparar los gestos candidatos que están almacenados en el sistema, se escoge el archivo donde está el *dataset* y se compara. Esto da como resultado los archivos mencionados en 4.1.3 del documento.

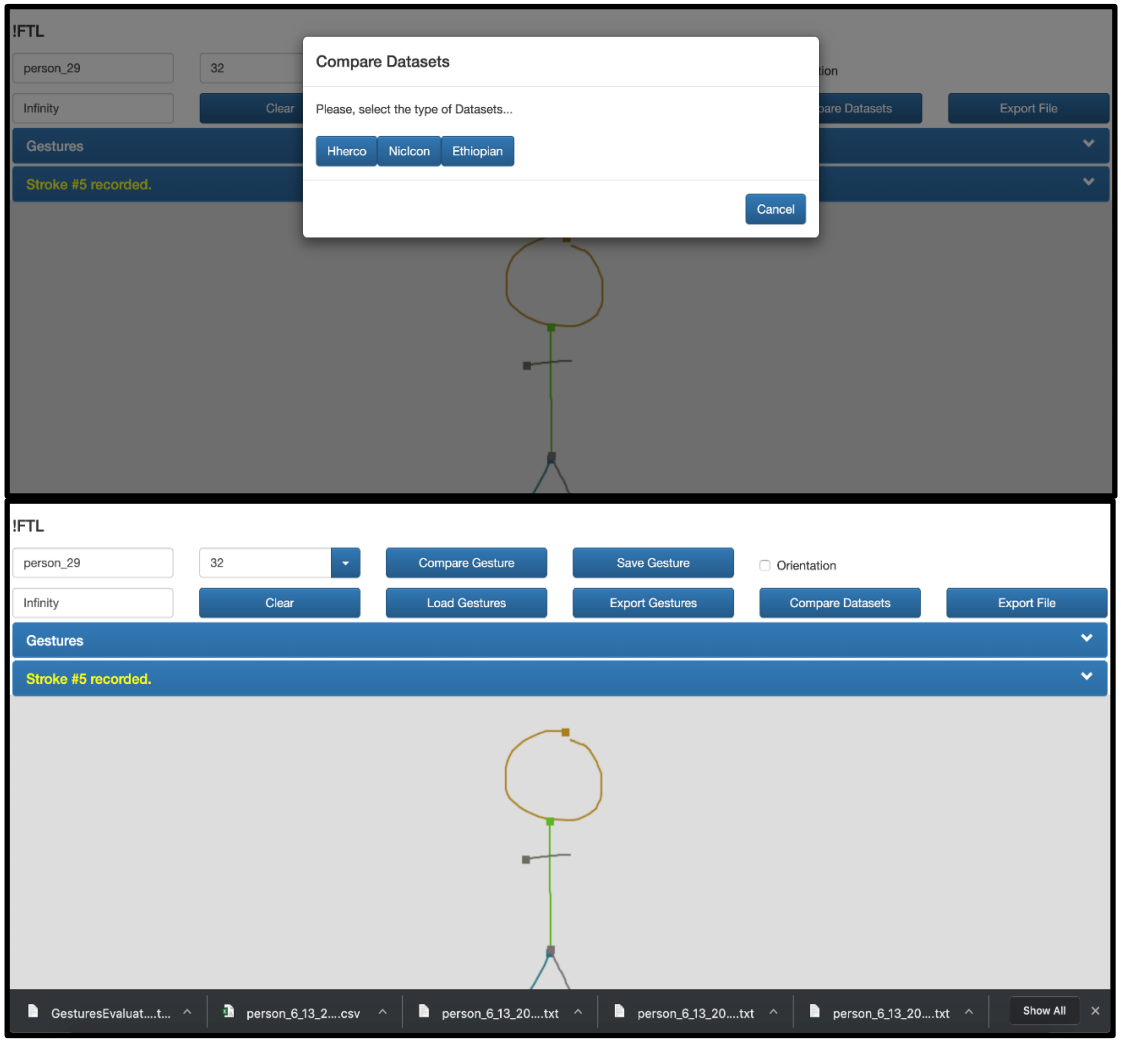


Figura 24. Ejemplo de la funcionalidad de comparar un *dataset* y sus resultados.

5. Capítulo V. Resultados

Esta sección presenta los resultados obtenidos de la evaluación comparativa de los algoritmos !FTL, $1, $P, $Q y Penny Pincher, realizada con los *datasets* Hhreco, NicIcon y Utopiano. Para cada uno de los *datasets* se utilizó la herramienta estadística de Excel para calcular la tasa de reconocimiento, el tiempo de reconocimiento y el rango de error de los algoritmos, anteriormente mencionados.

Por cada uno de estos valores se presentarán y discutirán los gráficos resultantes.

5.1 Tasa de reconocimiento

5.1.1 Análisis de la tasa de reconocimiento de NicIcon.

Tabla 8.

Tabla de la tasa de reconocimiento de NicIcon.

Average Recognition Rate							
	FTL (LSD)	FTL (NLSD)	$1	$P	$Q	Ppincher	Total
Gas	0.20%	0.00%	0.00%	0.91%	0.51%	2.42%	0.67%
Casualty	1.01%	0.00%	1.92%	3.03%	0.30%	1.31%	1.26%
Police	2.93%	1.22%	0.91%	2.83%	0.71%	0.10%	1.45%
Firebrigade	6.79%	0.30%	6.19%	2.98%	0.71%	1.52%	3.08%
Paramedics	4.34%	5.56%	8.18%	1.82%	1.41%	11.92%	5.54%
Accident	6.05%	18.63%	2.43%	5.34%	0.85%	2.46%	5.96%
Car	6.05%	18.63%	2.43%	5.34%	0.85%	2.46%	5.96%
Person	4.24%	0.20%	3.03%	10.00%	18.99%	9.70%	7.69%
Fire	8.91%	8.91%	19.21%	5.23%	0.20%	8.69%	8.53%
Injury	9.80%	0.10%	13.43%	11.82%	16.57%	11.21%	10.49%
Flood	12.73%	7.88%	8.28%	0.81%	37.98%	0.00%	11.28%
Bomb	31.59%	35.20%	7.10%	5.05%	0.91%	4.35%	14.03%
Roadblock	14.06%	14.67%	7.38%	26.12%	16.42%	7.08%	14.29%
Electricity	28.33%	21.22%	21.32%	15.80%	1.52%	19.83%	18.00%
Total	9.79%	9.47%	7.27%	6.93%	6.99%	5.93%	

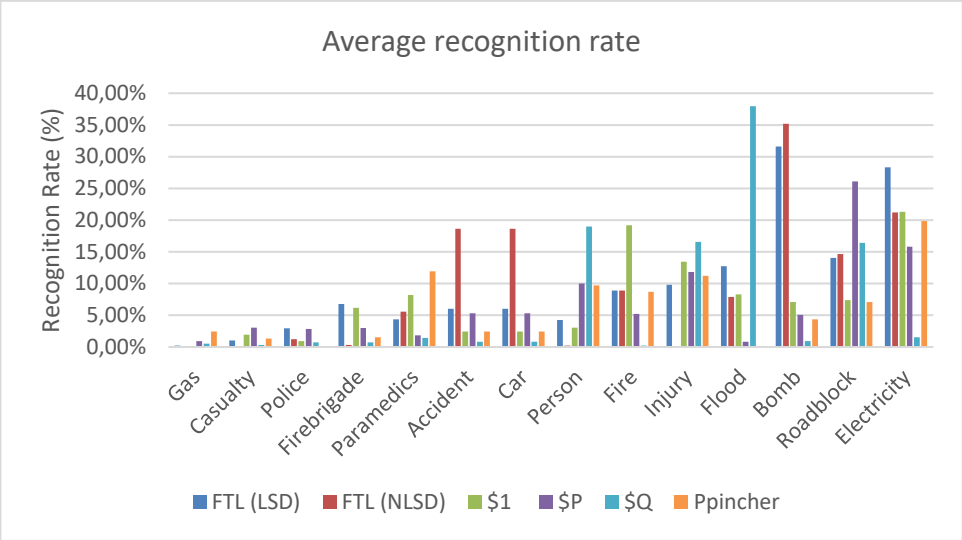


*Figura 25.*Gráfico de la tasa de reconocimiento de NicIcon.

La Figura 25 y la Tabla 8 muestran los resultados de promedio de la tasa de reconocimiento para el *dataset* de NicIcon. Se puede ver que el algoritmo que más reconoció fue !FTL(LSD), seguido por !FTL (NLSD). Por otro lado, los gestos que tuvieron más resultados certeros fueron *Electricity* y *Roadblock*.

Hay que tener en cuenta que algunos algoritmos no reconocieron en su totalidad ciertos gestos, como son el caso de !FTL (NLSD) y $1 con el gesto de *Gas*, y Penny Pincher con el gesto de *Flood*.

5.1.2 Análisis de la tasa de reconocimiento de Hhreco.

Tabla 9.

Tabla de la tasa de reconocimiento de Hhreco.

Average Recognition Rate							
	FTL (LSD)	FTL (NLSD)	$1	$P	$Q	Ppincher	Total
Parallelogram	1.96%	15.29%	4.71%	4.51%	5.49%	3.14%	5.85%
Hexagon	2.94%	18.63%	10.39%	0.78%	1.18%	1.37%	5.88%
Moon	12.35%	16.27%	4.90%	4.12%	4.12%	4.31%	7.68%
Pentagon	6.47%	14.12%	23.14%	0.59%	1.57%	0.39%	7.71%
Cylinder	0.98%	16.47%	19.80%	6.86%	4.12%	8.04%	9.38%
Trapezoid	0.78%	12.75%	13.14%	9.80%	11.57%	9.41%	9.58%
Square	13.92%	32.16%	8.63%	4.31%	4.31%	4.71%	11.34%
Triangle	12.16%	34.90%	27.45%	7.25%	5.88%	8.04%	15.95%
Cube	3.33%	15.29%	20.00%	22.75%	11.96%	24.12%	16.24%
Heart	13.14%	16.47%	34.12%	19.41%	23.92%	16.08%	20.52%
Callout	28.43%	34.31%	36.27%	12.35%	11.76%	12.94%	22.68%
Ellipse	64.12%	95.29%	15.10%	10.00%	8.82%	4.12%	32.91%
Arch	6.27%	5.10%	28.63%	50.20%	54.51%	54.90%	33.27%
Total	12.84%	25.16%	18.94%	11.76%	11.48%	11.66%	

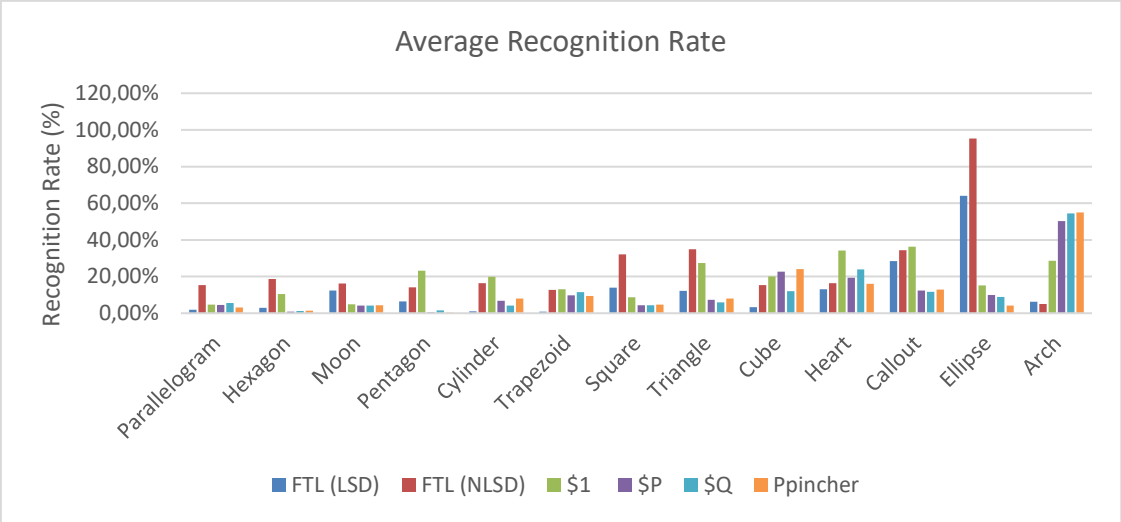


Figura 26. Gráfico de la tasa de reconocimiento de Hhreco.

La Tabla 9 y la Figura 26 indican los resultados obtenidos después de la experimentación con el *dataset* Hhreco. Se puede ver que el algoritmo que más reconoció fue !FTL(NLSD), seguido por $1. Los gestos más reconocidos fueron *Ellipse* y *Arch*.

5.1.3 **Análisis de la tasa de reconocimiento de Utopiano.**

Tabla 10.

Tabla de la tasa de reconocimiento de Utopiano.

Average Recognition Rate

	FTL (LSD)	FTL (NLSD)	1	$P	$Q	Ppincher	Total
B	1.60	3.85	3.90	1.10	0.85	0.95	2.042
E	16.50	23.00	19.00	7.00	7.00	5.50	13.000
S	2.50	6.50	44.00	10.00	13.00	10.00	14.333
Y	2.00	17.33	32.67	12.67	10.67	11.33	14.444
V	5.50	13.50	30.50	14.50	15.00	15.00	15.667
M	5.00	7.00	43.00	17.50	11.00	17.50	16.833
X	7.50	16.50	39.00	13.00	13.50	13.50	17.167
K	23.16	12.11	54.21	7.89	11.05	7.37	19.298
D	30.50	38.00	18.50	15.00	11.50	17.00	21.750
T	4.50	15.00	44.50	23.50	18.50	25.00	21.833
F	43.50	46.50	26.00	9.50	9.50	12.50	24.583
R	9.00	6.50	62.00	25.00	22.00	24.00	24.750
C	16.50	47.00	44.00	14.00	18.00	15.50	25.833
A	23.50	32.00	65.50	13.50	10.50	13.00	26.333
I	24.50	26.50	37.50	25.50	24.00	25.50	27.250
O	33.50	39.50	62.50	14.50	11.50	14.50	29.333
L	31.00	68.00	56.00	12.50	5.50	10.50	30.583
H	20.00	32.50	67.00	20.50	22.00	24.50	31.083
G	17.83	35.17	78.67	20.17	20.83	22.17	32.472
N	20.00	35.00	67.00	28.50	17.00	28.50	32.667
P	33.50	19.50	74.00	35.00	28.50	35.00	37.583
Q	40.50	37.00	70.50	32.00	34.00	30.50	40.750
Total	18.731	26.271	47.270	16.947	15.246	17.242	

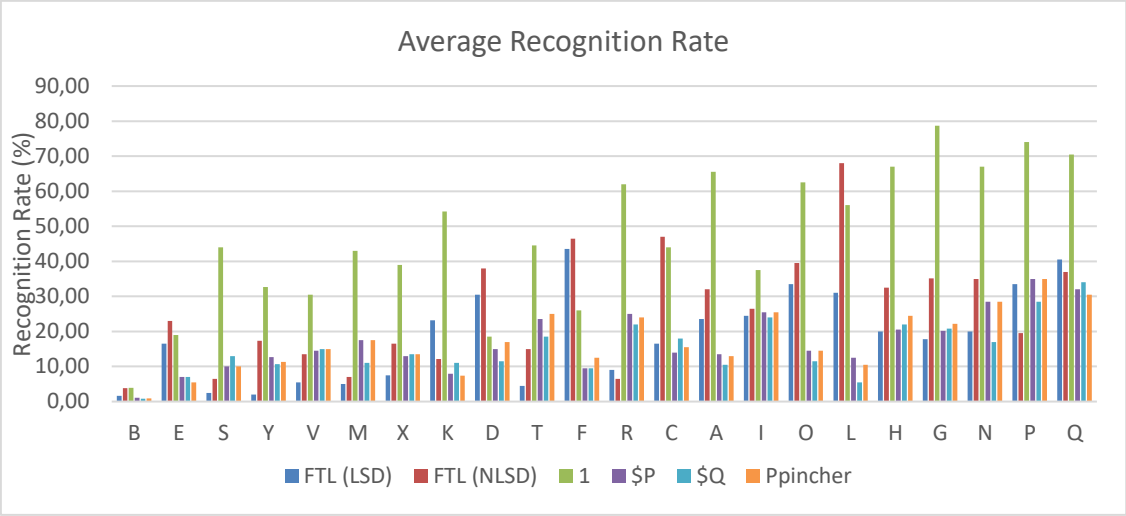


*Figura 27.*Gráfico de la tasa de reconocimiento de Utopiano.

La Tabla 10 y la Figura 27 indican los resultados obtenidos después de la experimentación con el *dataset* Utopiano. Se puede ver que el algoritmo que más reconoció fue $1 y !FTL(NLSD). Los gestos más reconocidos fueron la letra N y P.

5.2 Tiempo de ejecución

5.2.1 Análisis del tiempo de ejecución de NicIcon.

Tabla 11.

Tiempo de reconocimiento de NicIcon.

	Average time						
	FTL (LSD)	FTL (NLSD)	$1	$P	$Q	Ppincher	Total
Gas	0.56	0.00	0.00	1.36	0.46	0.35	0.45
Police	0.48	0.17	0.16	5.92	0.52	0.00	1.21
Flood	3.79	0.24	1.55	1.47	3.41	0.00	1.74
Paramedics	2.77	0.28	2.38	6.62	0.70	0.15	2.15
Roadblock	3.24	0.48	1.34	6.67	1.89	0.07	2.28
Fire	3.47	0.36	1.87	8.69	0.14	0.18	2.45
Firebrigade	1.28	0.02	1.02	12.40	0.41	0.02	2.53
Accident	0.48	0.32	0.52	13.78	1.45	0.06	2.77
Car	0.48	0.32	0.52	13.78	1.45	0.06	2.77
Casualty	0.54	0.00	0.81	18.50	0.69	0.08	3.44
Injury	3.82	0.01	2.04	8.57	8.67	0.39	3.92
Bomb	2.07	1.95	2.39	14.31	2.58	1.05	4.06
Person	4.50	0.04	0.68	14.98	4.83	0.26	4.22
Electricity	3.83	1.02	2.13	28.84	1.95	0.35	6.35
Total	2.24	0.37	1.24	11.14	2.08	0.22	

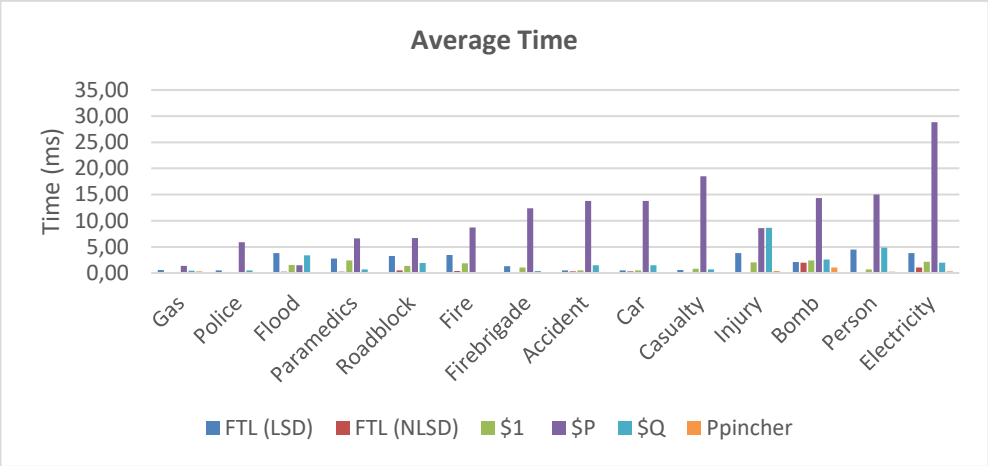


*Figura 28.*Gráfico del tiempo de reconocimiento de NicIcon.

La Figura 28 y la Tabla 11 muestran los resultados de promedio del tiempo de reconocimiento del *dataset* NicIcon. Técnicamente se observa que los algoritmos Penny Pincher, seguido de !FTL (NLSD), son los más rápidos en reconocer los gestos de NicIcon. *Gas* y *Police* fueron los gestos con menor tiempo de reconocimiento.

A pesar de la rapidez de estos algoritmos, hay que tener en cuenta que algunos de los casos, estos algoritmos no reconocen en su totalidad los gestos, como es el caso de Penny Pincher con el gesto *Flood* y *Police*. Mientras que en el caso de! FTL(NLSD), los gestos sin reconocer son *Gas* y *Casualty*.

5.2.2 Análisis del tiempo de ejecución de Hhreco.

Tabla 12.

Tiempo de reconocimiento de Hhreco.

Average Time

	FTL (LSD)	FTL (NLSD)	$1	$P	$Q	Ppincher	Total
Hexagon	0.13	0.68	0.26	0.23	0.09	0.12	0.25
Pentagon	0.16	0.25	0.44	0.45	0.39	0.06	0.29
Cylinder	0.07	0.84	0.45	0.60	0.57	0.17	0.45
Parallelogram	0.06	0.77	0.20	0.94	1.19	0.23	0.56
Moon	0.29	1.28	0.31	1.51	0.64	0.33	0.73
Trapezoid	0.06	0.65	0.30	2.20	1.13	0.34	0.78
Triangle	0.98	0.90	0.37	1.46	0.91	0.29	0.82
Ellipse	0.78	0.73	1.18	1.66	0.71	0.16	0.87
Square	0.29	1.12	0.12	2.01	1.52	0.35	0.90
Callout	0.35	1.67	0.44	2.02	1.26	0.44	1.03
Cube	0.48	0.19	0.28	3.28	1.60	0.61	1.07
Heart	0.82	0.37	0.50	3.58	2.26	0.57	1.35
Arch	0.21	0.22	0.48	4.69	2.87	0.76	1.54
Total	0.36	0.74	0.41	1.89	1.16	0.34	

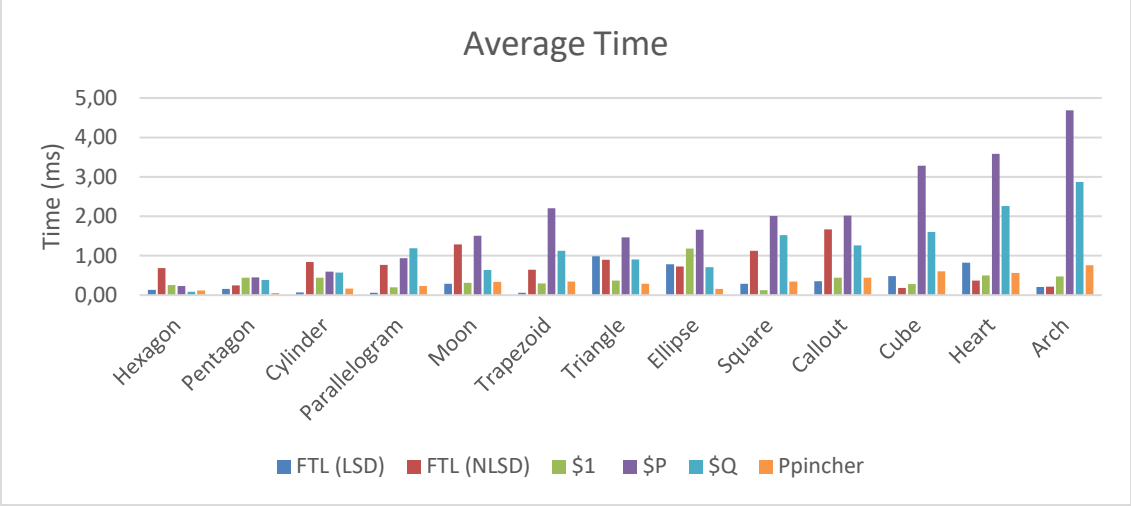


Figura 29. Gráfico del tiempo de reconocimiento de Hhreco.

La Figura 29 y la Tabla 12 expresan los resultados de promedio del tiempo de reconocimiento del *dataset* Hhreco.

Se indica que los algoritmos Penny Pincher, seguido de !FTL (LSD), son los más rápidos en reconocer los gestos de Hhreco. Y las representaciones más rápidas en ser reconocidas fueron *Pentagon* y *Hexagon*.

A pesar del resultado obtenido por estos algoritmos, su tasa de reconocimiento no fue muy alta, por lo tanto, estos algoritmos no pueden ser considerados muy eficientes.

Por el contrario, esto muestra que estos tuvieron la menor capacidad de reconocer a los gestos.

5.2.3 Análisis del tiempo de ejecución de Utopiano.

Tabla 13.

Tiempo de reconocimiento de Utopiano.

	Average Time						
	FTL (LSD)	FTL (NLSD)	1	$P	$Q	Ppincher	Total
K	1.66	0.72	1.39	5.50	2.12	0.31	1.951
F	0.87	0.37	1.25	8.18	3.34	0.55	2.428
S	0.21	0.28	1.45	8.18	4.21	0.64	2.496
E	1.08	0.29	0.90	7.68	4.90	0.25	2.519
A	1.11	1.00	1.92	7.88	2.86	0.49	2.545
D	1.01	0.34	0.85	11.48	1.96	0.59	2.704
L	0.54	0.49	1.31	10.35	3.45	0.54	2.779
O	1.02	0.34	1.35	8.75	4.66	0.58	2.783
M	0.30	0.55	1.40	9.97	4.32	0.64	2.862
B	0.71	0.31	1.90	9.35	4.77	0.35	2.897
C	0.56	0.27	1.37	9.80	5.25	0.69	2.990
X	0.33	0.52	1.93	11.11	3.40	0.69	2.997
H	0.89	0.33	1.55	11.91	3.01	0.69	3.063
G	1.18	0.77	1.43	10.72	4.05	0.79	3.156
T	0.28	0.32	1.42	12.36	4.67	0.83	3.313
V	0.36	0.24	1.11	14.46	2.96	0.79	3.321
I	1.05	0.32	1.22	12.78	4.44	0.74	3.424
R	0.71	0.09	1.40	13.41	5.05	0.84	3.584
Q	1.12	0.48	1.38	13.93	5.10	0.93	3.822
N	1.48	0.30	1.41	14.62	5.58	0.78	4.030
P	0.86	0.32	1.20	15.37	5.64	0.94	4.054
Y	2.55	2.75	3.48	13.78	5.47	3.00	5.170
Total	0.904	0.518	1.483	10.980	4.146	0.757	

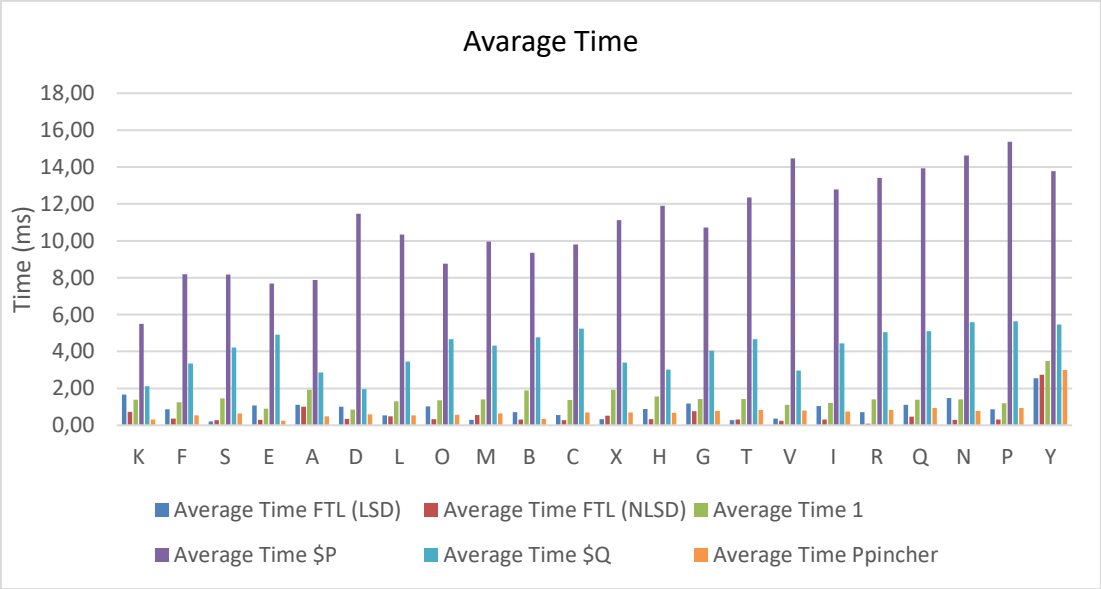


Figura 30. Gráfico del tiempo de reconocimiento de Utopiano.

La Figura 30 y la Tabla 14 muestran los resultados del promedio del tiempo de reconocimiento del *dataset* Utopiano. Se indica que los algoritmos !FTL (NLSD), seguido de Penny Pincher, son los más rápidos en reconocer los gestos de NicIcon. Y las representaciones más rápidas en ser reconocidas fueron la letra F y K.

5.3 Discusión

Tabla 14.

Recopilación de resultados.

Datasets	NicIcon			Hhreco			Utopian		
Recognition Rate	1st	!FTL (LSD)	9.79%	1st	**!FTL (NLSD)**	**25.26%**	1st	$1	47.27%
	2nd	**!FTL (NLSD)**	**9.47%**	2nd	$1	18.94%	2nd	**!FTL (NLSD)**	**26.27%**
Time	1st	**Penny Pincher**	**0.216**	1st	**Penny Pincher**	**0.34**	1st	**!FTL (NLSD)**	**0.518**
	2nd	**!FTL (NLSD)**	**0.372**	2nd	!FTL (LSD)	0.361	2nd	**Penny Pincher**	**0.757**

En la tabla 14 se muestra una recopilación de los resultados obtenidos de la evaluación comparativa presentada en la sección 5.2. La Tabla indica los dos mejores algoritmos, según la tasa de reconocimiento y el tiempo de reconocimiento para cada uno de los *datasets*.

En el primer *dataset*, NicIcon, se distingue a !FTL(LSD) como el algoritmo con mayor tasa de reconocimiento con un 9.79%. El siguiente algoritmo es !FTL(NLSD) con un 9.47% de tasa de reconocimiento, es decir, solo 0.32% menor que !FTL(LSD). Tomando en cuenta las hipótesis de similitud que se definieron en la sección 2.4.1 del Capítulo 2, muestra que, en el caso de *Electricity* y *Roadblock*, los dos gestos con mayor número aciertos, los algoritmos no cumplieron con las expectativas, ya que se propuso que tenga solo una tasa de error del 21.43%, en ambos gestos, y se obtuvo un 90.21% y 90.53% de error con !FTL(NLSD), mientras que !FTL(LSD) mostró 71.67% y 85.94% de reconocimientos fallidos. Asimismo, el algoritmo de Penny Pincher es el más rápido para el *dataset* de NicIcon con 0.22 ms, pero su tasa de reconocimiento es solo del 5.93%. El siguiente más rápido es !FTL(NLSD) con 0.37 ms, es decir

0.15 ms más. Por su tasa de reconocimiento y su rapidez, hace que el algoritmo !FTL(NLSD) sea el más eficiente para el *dataset* NicIcon.

En Hhreco, se puede ver qué !FTL(NLSD) como el algoritmo con la mejor tasa de reconocimiento con un 25.26%. El algoritmo que obtuvo el segundo lugar fue $1 con un 18.94% de tasa de reconocimiento, es decir que es 6.32% menor que !FTL(NLSD). Según los supuestos de similitud que se definieron en la sección 2.4.2 del Capítulo 2, en el caso de *Ellipse* y *Arch*, que son los dos gestos más acertados, se cumplió medianamente con lo propuesto, ya que *Ellipse* con !FTL(NLSD) tuvo 95.29% de asertividad, mientras que con *Arch* obtuvo 5.10% de aciertos sobre el 15.34% de la tasa de error propuesta. Por otro lado, con $1 y *Ellipse* dio como resultado 15.10% de asertividad sobre el 70% que se estimaba, y con *Arch* consiguió 28.63%, superando la tasa de error indicada de 15.34%. Desde el punto de vista del tiempo de reconocimiento, el algoritmo de Penny Pincher fue el más rápido para el *dataset* de Hhreco con 0.34 ms, solo que su tasa de reconocimiento solo llegó al 11.66%. El siguiente algoritmo más rápido fue !FTL(NLSD) con 0.36 ms, es decir 0.02 ms más que el tiempo de Penny Pincher. Esto demuestra nuevamente su eficiencia.

Por último, en el *dataset* del Alfabeto Utopiano, se muestra que el algoritmo $1 tiene la mejor tasa de reconocimiento con un 47.27%. El segundo algoritmo con mayor número de aciertos fue !FTL(NLSD) con un 26.27%, es decir que es 21% menor que $1. Siguiendo la hipótesis de similitud que se propuso en la sección 2.4.3 del Capítulo 2, en el caso de la letra P y N, que son los dos gestos con mayor tasa de reconocimiento, se cumplió a totalidad con lo estimado debido a que la letra P con $1 tuvo 74% de asertividad, mientras que con la letra N consiguió 67% de aciertos sobre el 18.18% de la tasa de error que se definió en ambas letras. Con !FTL(NLSD) y la letra P obtuvo 19% sobre el 18.18% que se estimaba para acertar, y con la letra N consiguió 35% de asertividad, superando la tasa de error indicada anteriormente. En lo que se refiere al tiempo de reconocimiento, el algoritmo !FTL(NLSD) fue el más rápido en el *dataset* Utopiano con 0.52 ms, seguido por Penny Pincher con 0.77 ms, es decir solo

una diferencia de 0.25 ms. Y por tercera vez, !FTL(NLSD) volvió a ser el algoritmo más rápido y uno de los más certeros.

6. Conclusiones y Recomendaciones

6.1 Conclusiones

Este trabajo presentó un estudio comparativo de diferentes algoritmos con el uso *datasets* existentes y un *dataset* generado para este proyecto. Después de mostrar los resultados finales y, a pesar de que los algoritmos fueron evaluados en las mismas condiciones de ejecución, con los mismos conjuntos de gestos de referencia y en el mismo computador, las tendencias que se mostraron no son cien por ciento certeras. No obstante, los algoritmos !FTL, $1 y Penny Pincher son los algoritmos que el estudio sugiere y concluye para ser utilizados. Por ello, se recomienda que sean implementados ampliamente en las aplicaciones que busquen el uso de reconocimiento de gestos.

Como se mostraron en los resultados, y si bien la experimentación se realizó en el mismo computador, existieron picos de tiempos que variaron debido a los eventos que el *browser* decidió asignar, más tiempo para ser realizados. Por este motivo, se recomienda que se busque una manera para que la asignación de tiempo a cada proceso sea más uniforme. Una propuesta es implementar un servicio con lenguaje de bajo nivel como C, para hacer la evaluación de los algoritmos dentro de una microcontroladora.

El proceso de evaluación fue semiautomático ya que se tenía que hacer los gestos candidatos manualmente por cada *dataset* y compararlos con los gestos de los *datasets*. Por ello, se recomienda automatizar este proceso mediante una evaluación que se pueda tomar como referencia un gesto del *datasets* de forma aleatoria y evaluar los gestos restantes contra este gesto seleccionado. Este proceso se repetiría por cada uno de los gestos del *dataset*, es decir, mientras se exista un gesto sin evaluar, este sería seleccionado y evaluado contra el grupo de gestos restantes. Con esta forma de evaluación se evitaría realizar gestos candidatos por parte del usuario

6.2 Recomendaciones

Uno de los principales inconvenientes encontrados al hacer las comparaciones tiene que ver con los diferentes formatos para los datos que existen por *dataset*. Cada *dataset* tiene una forma diferente de guardar las repeticiones de los gestos, dicho de otro modo, la forma de almacenar el nombre de la repetición que se realizó, la manera de desplegar las coordenadas de los puntos y como se guarda cada archivo con todas las repeticiones según usuario o representación. Por consiguiente, se sugiere que se realice un protocolo para que los algoritmos se ejecuten en entornos similares.

El uso de *datasets* ayudó a ahorrar tiempo a la diversificación de gestos para el proceso evaluativo, y como se mencionó anteriormente, existen varios *datasets* que no pudieron ser utilizados en este proyecto. Por esta razón, se recomienda recopilar y utilizar otros *datasets* en un futuro. Existen *datasets* que tienen características que desafían a los algoritmos de reconocimiento, como es el caso de *datasets* con figuras en 3D. Esto abre la oportunidad de evaluar más algoritmos que se especialicen en estos gestos, como el de $3 o el mismo !FTL, que pueden reconocer figuras en tercera dimensión.

REFERENCIAS

Ager, S. (2019). *Omniglot*. Recuperado el 10 de enero de 2019, de http://omniglot.com/conscripts/utopian.htm

Anthony, L., & Jacob O, W. (2010). A lightweight multistroke recognizer for user interface prototypes. *Proceedings of Graphics Interface 2010. Canadian Information Processing Society*, 245-252. Recuperado el 21 de mayo de 2019

Apple Inc. (2019). *Swift*. Recuperado el 28 de abril de 2019, de About Swift: https://swift.org/about/#swiftorg-and-open-source

Cao, X., & Shumin, Z. (2007). Modeling human performance of pen stroke gestures. *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1495-1504. Recuperado el 23 de marzo de 2019

Daiki, Y. (2016). A Fast and Lightweight Unistroke Recognizer for Large User-defined vocabulary. *University of Tsukuba*. Recuperado el 23 de abril de 2019

Delaye, A. (2011). Recuperado el 19 de marzo de 2019, de sites.google.com: https://sites.google.com/site/adriendelaye/home/pen-and-touch-datasets

Duda, R. O., Hart, P. E., & Stork, D. G. (2012). Pattern classification. *John Wiley & Sons*. Recuperado el 09 de febrero de 2019

Eugene M. Taranta, I., & Joseph J. LaViola, J. (2015). Penny Pincher: A Blazing Fast, Highly Accurate $-family Recognizer. *Proceedings of the 41st Graphics Interface Conference*, 195-202. Recuperado el 03 de marzo de 2019

Hse, H., & Newton, A. R. (2004). Sketched symbol recognition using zernike moments. *Proceedings of the 17th International Conference on Pattern Recognition*, 367-370. Recuperado el 15 de Junio de 2019

Kratz, S., & Michael, R. (2010). A $3 gesture recognizer: simple gesture recognition for devices equipped with 3D acceleration sensors. *Proceedings of the 15th international conference on Intelligent user interfaces*, 341-344. Recuperado el 19 de Enero de 2019

Luzzi, L., & Roselli, P. (2018). The convergence of a gesture recognizer and the shape of a plane gesture. Recuperado el 23 de mayo de 2019

Microsoft. (2019). *Microsoft*. Recuperado el 10 de febrero de 2019, de Un paseo por el lenguaje C#: https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/

Mozilla and individual contributors. (2019). *MDN web docs*. Recuperado el 14 de enero de 2019, de About JavaScript: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript?redirectlocale=en-US&redirectslug=JavaScript%2FAbout_JavaScript

Niels, R., Willems, D., & Vuurpijl, L. (2008). The NicIcon Database of Handwritten Icons for Crisis Management. *Nijmegen Institute for Cognition and Information Radboud University Nijmegen*. Recuperado el 13 de marzo de 2019

Renau-Ferrer, N., Li, P., Delaye, A., & Anquetil, E. (2012). The ILGDB database of realistic pen-based gestural commands. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, 3741-3744. Recuperado el 19 de febrero de 2019

Rubine, D. (1991). Specifying gestures by example. *ACM*, Vol. 25, No. 4, pp. 329-337. Recuperado el 22 de junio de 2019

Vanderdonckt, J., Magrofuoco, N., Burny, N., Pérez, J. L., & Roselli, P. (2017). A Survey of 2D Gesture Recognition Techniques. *ACM International Conference on Intelligent User Interfaces*. Recuperado el 19 de abril de 2019

Vanderdonckt, J., Roselli, P., & Pérez, J. L. (2018). !FTL, an Articulation-Invariant Stroke Gesture Recognizer with Controllable Position, Scale, and

Rotation Invariances. *Proceedings of the 2018 on International Conference on Multimodal Interaction*, 125-134. Recuperado el 12 de abril de 2019

Vatavu, R.-D. (2017). Improving gesture recognition accuracy on touch screens for users with low vision. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 4667-4679. Recuperado el 12 de mayo de 2019

Vatavu, R.-D., Anthony, L., & Wobbrock, a. J. (2018). $Q: A SuperQuick, Articulation-Invariant Stroke-Gesture Recognizer for Low-Resource Devices. *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 23. Recuperado el 19 de abril de 2019

Vatavu, R.-D., Anthony, L., & Wobbrock, J. O. (2012). Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes. *ICMI '12 Proceedings of the 14th ACM international conference on Multimodal interaction*, 273-280. Recuperado el 12 de febrero de 2019

Wobbrock, J., Wilson, A., & Li, Y. (2007). Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. *ICMI '07 Proceedings of the 20th annual ACM symposium on User interface software and technology. ACM.*, 159-168. Recuperado el 23 de abril de 2019

Yang, L. (2010). A Fast and Accurate Gesture Recognizer. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2169-2172. Recuperado el 11 de mayo de 2019

ANEXOS

Anexo No 1: Pseudocódigo de $P

Recognizer main function. Match *points* against a set of *templates* by employing the Nearest-Neighbor classification rule. Returns a normalized score in [0..1] with 1 denoting perfect match.

$P-Recognizer (Points *points*, Templates *templates*)

```
 1: n ← 32 // number of points
 2: Normalize(points, n)
 3: score ← ∞
 4: for each template in templates do
 5:     Normalize(template, n) // should be pre-processed
 6:     d ← Greedy-Cloud-Match(points, template, n)
 7:     if score > d then
 8:         score ← d
 9:         result ← template
10: score ← Max((2.0 − score)/2.0, 0.0) // normalize score in [0..1]
11: return ⟨result, score⟩
```

Cloud matching function. Match two clouds (*points* and *template*) by performing repeated alignments between their points (each new alignment starts with a different starting point index i). Parameter $\epsilon \in [0..1]$ controls the number of tested alignments ($n^{\epsilon} \in \{1, 2, ...n\}$). Returns the minimum alignment cost.

Greedy-Cloud-Match (Points *points*, Points *template*, int n)

```
1: ε ← .50
2: step ← ⌊n^(1−ε)⌋
3: min ← ∞
4: for i = 0 to n step step do
5:     d1 ← Cloud-Distance(points, template, n, i)
6:     d2 ← Cloud-Distance(template, points, n, i)
7:     min ← Min(min, d1, d2)
8: return min
```

Distance between two clouds. Compute the minimum-cost alignment between *points* and *tmpl* starting with point *start*. Assign decreasing confidence *weights* $\in [0..1]$ to point matchings.

Cloud-Distance (Points *points*, Points *tmpl*, int n, int *start*)

```
 1: matched ← new bool[n]
 2: sum ← 0
 3: i ← start // start matching with points_i
 4: do
 5:     min ← ∞
 6:     for each j such that not matched[j] do
 7:         d ← Euclidean-Distance(points_i, tmpl_j)
 8:         if d < min then
 9:             min ← d
10:             index ← j
11:     matched[index] ← true
12:     weight ← 1 − ((i − start + n) MOD n)/n
13:     sum ← sum + weight · min
14:     i ← (i + 1) MOD n
15: until i == start // all points are processed
16: return sum
```

Gesture normalization. Gesture points are resampled, scaled with shape preservation, and translated to origin.

NORMALIZE (POINTS *points*, int n)

```
1: points ← RESAMPLE(points, n)
2: SCALE(points)
3: TRANSLATE-TO-ORIGIN(points, n)
```

Points resampling. Resample a *points* path into n evenly spaced points. We use $n = 32$.

RESAMPLE (POINTS *points*, int n)

```
1: I ← PATH-LENGTH(points) / (n − 1)
2: D ← 0
3: newPoints ← points_0
4: for each p_i in points such that i ≥ 1 do
5:     if p_i.strokeId == p_{i−1}.strokeId then
6:         d ← EUCLIDEAN-DISTANCE(p_{i−1}, p_i)
7:         if (D + d) ≥ I then
8:             q.x ← p_{i−1}.x +((I − D)/d) · (p_i.x - p_{i−1}.x)
9:             q.y ← p_{i−1}.y +((I − D)/d) · (p_i.y - p_{i−1}.y)
10:            q.strokeId ← p_i.strokeId
11:            APPEND(newPoints, q)
12:            INSERT(points, i, q) // q will be the next p_i
13:            D ← 0
14:        else D ← D + d
15: return newPoints
```

PATH-LENGTH (POINTS *points*)

```
1: d ← 0
2: for each p_i in points such that i ≥ 1 do
3:     if p_i.strokeId == p_{i−1}.strokeId then
4:         d ← d + EUCLIDEAN-DISTANCE(p_{i−1}, p_i)
5: return d
```

Points rescale. Rescale *points* with shape preservation so that the resulting bounding box will be $\subseteq [0..1] \times [0..1]$.

SCALE (POINTS *points*)

1: $x_{min} \leftarrow \infty$, $x_{max} \leftarrow 0$, $y_{min} \leftarrow \infty$, $y_{max} \leftarrow 0$
2: **for each** p in *points* **do**
3: $\quad x_{min} \leftarrow \text{MIN}(x_{min}, p.\text{x})$
4: $\quad y_{min} \leftarrow \text{MIN}(y_{min}, p.\text{y})$
5: $\quad x_{max} \leftarrow \text{MAX}(x_{max}, p.\text{x})$
6: $\quad y_{max} \leftarrow \text{MAX}(y_{max}, p.\text{y})$
7: $scale \leftarrow \text{MAX}(x_{max} - x_{min}, y_{max} - y_{min})$
8: **for each** p in *points* **do**
9: $\quad p \leftarrow ((p.\text{x} - x_{min})/scale, (p.\text{y} - y_{min})/scale,$ p.strokeId)

Points translate. Translate *points* to the origin $(0, 0)$.

TRANSLATE-TO-ORIGIN (POINTS *points*, int n)

1: $c \leftarrow (0, 0)$ // will contain centroid
2: **for each** p in *points* **do**
3: $\quad c \leftarrow (c.\text{x} + p.\text{x}, c.\text{y} + p.\text{y})$
4: $c \leftarrow (c.\text{x}/n, c.\text{y}/n)$
5: **for each** p in *points* **do**
6: $\quad p \leftarrow (p.\text{x} - c.\text{x}, p.\text{y} - c.\text{y},$ p.strokeId)

Anexo No 2: Pseudocódigo de $Q

$Q-RECOGNIZER (POINTS *points*, TEMPLATES *templates*)

```
1: n ← 32, m ← 64   // defaults for cloud size (n) and size of the look-up table (m)
2: NORMALIZE(points, n, m)   // templates have already been normalized
3: score ← ∞
4: for each template in templates do
5:     d ← CLOUD-MATCH(points, template, n, score)
6:     if d < score then
7:         score ← d
8:         result ← template
9: return ⟨result, score⟩   // the closest template from the set and the smallest score
```

CLOUD-MATCH (POINTS *points*, POINTS *template*, int *n*, int *min*)

```
1: step ← ⌊n^0.5⌋
2: //compute lower bounds for both matching directions between points and template
3: LB_1 ← COMPUTE-LOWER-BOUND(points, template, step, template.LUT)
4: LB_2 ← COMPUTE-LOWER-BOUND(template, points, step, points.LUT)
5: for i ← 0 to n − 1 step step do
6:     if LB_1[i/step] < min then
7:         min ← MIN(min, CLOUD-DISTANCE(points, template, n, i, min))
8:     if LB_2[i/step] < min then
9:         min ← MIN(min, CLOUD-DISTANCE(template, points, n, i, min))
10: return min
```

CLOUD-DISTANCE (POINTS *points*, POINTS *template*, int *n*, int *start*, float *minSoFa*

```
1: unmatched ← {0,1,2,...,n−1}   // indices of unmatched points from template
2: i ← start      // start the matching from this index in the points cloud
3: weight ← n   // weights decrease from n to 1
4: sum ← 0      // computes the cloud distance between points and template
5: do
6:     min ← ∞
7:     for each j in unmatched do
8:         d ← SQR-EUCLIDEAN-DISTANCE(points_[i], template_[j])
9:         if d < min then
10:            min ← d
11:            index ← j
12:    REMOVE(unmatched, index)   // implementable in O(1)
13:    sum ← sum + weight · min
14:    if sum ≥ minSoFar then
15:        return sum          // early abandoning of computations
16:    weight ← weight − 1   // weights decrease from n to 1
17:    i ← (i + 1) MOD n     // advance to the next point in points
18: until i == start
19: return sum
```

COMPUTE-LOWER-BOUND (POINTS *points*, **POINTS** *template*, **int** *step*, **int[,]** *LUT*)

```
1:  LB ← new float[n/step + 1]   // multiple lower bounds, one for each starting point
2:  SAT ← new float[n]   // summed area table for fast computations (see text)
3:  // first, compute the lower bound for starting point index 0
4:  LB[0] ← 0
5:  for i ← 0 to n − 1 do
6:      index ← LUT[points[i].x, points[i].y]
7:      d ← SQR-EUCLIDEAN-DISTANCE(points[i], template[index])
8:      SAT[i] ← (i == 0) ? d : SAT[i−1] + d
9:      LB[0] ← LB[0] + (n − i) · d
10: // compute the lower bound for the other starting points (see formula in the text)
11: for i ← step to n − 1 step step do
12:     LB[i/step] ← LB[0] + i · SAT[n−1] − n · SAT[i−1]
13: return LB
```

NORMALIZE (POINTS *points*, int *n*, int *m*)

```
1: points ← RESAMPLE(points, n)
2: TRANSLATE-TO-ORIGIN(points, n)
3: SCALE(points, m)
4: LUT ← COMPUTE-LUT(m, points, n)
```

RESAMPLE (POINTS *points*, int *n*)

```
1: I ← PATH-LENGTH(points) / (n − 1)
2: D ← 0
3: newPoints ← { points[0] }
4: for i ← 1 to n − 1 do
5:     if points[i].strokeId == points[i−1].strokeId then
6:         d ← EUCLIDEAN-DISTANCE(points[i−1], points[i])
7:         if (D + d) ≥ I then
8:             q.x ← points[i−1].x + (I − D)/d · (points[i].x - points[i−1].x)
9:             q.y ← points[i−1].y + (I − D)/d · (points[i].y - points[i−1].y)
10:            APPEND(newPoints, q)
11:            INSERT(points, i, q) // q will be the next points[i]
12:            D ← 0
13:        else D ← D + d
14: return newPoints
```

TRANSLATE-TO-ORIGIN (POINTS *points*, int *n*)

```
1: c ← (0,0)   // will compute the centroid of the points cloud
2: for each p in points do
3:     c ← (c.x + p.x, c.y + p.y)
4: c ← (c.x/n, c.y/n)
5: for each p in points do
6:     p ← (p.x − c.x, p.y − c.y)
```

SCALE (POINTS *points*, int *m*)

```
1: x_min ← ∞, x_max ← −∞, y_min ← ∞, y_max ← −∞
2: for each p in points do
3:     x_min ← MIN(x_min, p.x)
4:     y_min ← MIN(y_min, p.y)
5:     x_max ← MAX(x_max, p.x)
6:     y_max ← MAX(y_max, p.y)
7: s ← MAX(x_max − x_min, y_max − y_min)/(m − 1)   // scale factor
8: for each p in points do
9:     p ← ((p.x − x_min)/s, (p.y − y_min)/s)   // p.x and p.y are now integers in 0...m − 1
```

COMPUTE-LUT (POINTS *points*, int *n*, int *m*)

```
1: LUT ← new int[m, m]
2: for x ← 0 to m − 1 do
3:     for y ← 0 to m − 1 do
4:         min ← ∞
5:         for i ← 0 to n − 1 do
6:             d ← SQR-EUCLIDEAN-DISTANCE(points[i], new POINT(x, y))
7:             if d < min then
8:                 min ← d
9:                 index ← i
10:        LUT[x,y] ← index
11: return LUT
```

PATH-LENGTH (POINTS *points*)

```
1: d ← 0  // will compute the path length
2: for i ← 1 to n − 1 do
3:     if points[i].strokeId == points[i−1].strokeId then
4:         d ← d + EUCLIDEAN-DISTANCE(points[i−1], points[i])
5: return d
```

SQR-EUCLIDEAN-DISTANCE (POINT *a*, POINT *b*)

```
1: return (a.x − b.x)^2 + (a.y − b.y)^2  // much faster to compute without the sqrt()
```

EUCLIDEAN-DISTANCE (POINT *a*, POINT *b*)

```
1: return sqrt(SQR-EUCLIDEAN-DISTANCE(a, b))
```

Anexo No 3: Pseudocódigo de Penny Pincher

RESAMPLE-BETWEEN-POINTS (POINTS *points*)

$I \leftarrow$ PATH-LENGTH(A) / $(n-1)$
$D \leftarrow 0$, $v = \{\}$
$prev \leftarrow points_0$
foreach p_i **in** *points* such that $i \geq 1$ **do**
    $d \leftarrow$ DISTANCE(p_i, p_{i-1})
    if $D + d \geq I$ **then**
        $q \leftarrow p_{i-1} + (p_i - p_{i-1}) * (I - D)/d$
        $r \leftarrow q - prev$ ▷ Equation 2
        $r \leftarrow r$ / DISTANCE($(0,0), r$) ▷ Only if template
        $D \leftarrow 0$, $prev \leftarrow q$
        APPEND(v, r)
        INSERT($points, i, q$)
    else $D \leftarrow D + d$
return v

PATH-LENGTH (POINTS *points*)

$d \leftarrow 0$
for $i \leftarrow 1$ **to** $|points| - 1$ **do**
    $d \leftarrow d +$ DISTANCE($points_{i-1}$, $points_i$)
return d

DISTANCE (POINT a, POINT b)

return $\sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$

RECOGNIZE (POINTS g, TEMPLATES $\mathscr{T}$)

$c \leftarrow$ RESAMPLE-BETWEEN-POINTS(g)
$similarity \leftarrow -\infty$
foreach t **in** $\mathscr{T}$ **do**
    $d \leftarrow 0$
    for $i \leftarrow 0$ **to** $n - 2$ **do**
        $d \leftarrow d + t_{i_x} c_{i_x} + t_{i_y} c_{i_y}$ ▷ Equation 6
    if $d > similarity$ **then**
        $similarity \leftarrow d$
        $T \leftarrow t$
return $\langle T, similarity \rangle$

Anexo No 4: Pseudocódigo de !FTL

```
$Compare-Gestures (Gesture gesture, NumberOfPoints, Threshold, Format,
Orientation, method, Gestures-Database RPs)
 1: t0 ← Time
 2: P ← NumberOfPoints
 3: if length(gesture) = P then
 4:     SP ← gesture
 5: else
 6:     SP ← Interpolate-Gesture(gesture)
 7:     L ← length(SP)
 8: pos ← −1
 9: LA ← length(RPs) − 1
10: for g = 0 to LA do
11:     RPg ← RPs[g]
12:     if length(RPg) = P then
13:         RP ← RPg.Points
14:     else
15:         RP ← Interpolate-Gesture(RPg.Points)
16:         L ← length(RP)
17:     d ← 0
18:     dPlus ← 0
19:     dLess ← 0
20:     for i = 1 to L − 2 do
21:         if method = "LSD" then
22:             lSd ← Local-Shape-Distance(
23:                     newPoint( SP(i) − SP(i − 1) ),
24:                     newPoint( SP(i + 1) − SP(i) ),
25:                     newPoint( RP(i) − RP(i − 1) ),
26:                     newPoint( RP(i + 1) − RP(i) ) )
27:         else
28:             lSd ← Normalized-Local-Shape-Distance(
29:                     newPoint( SP(i) − SP(i − 1) ),
30:                     newPoint( SP(i + 1) − SP(i) ),
31:                     newPoint( RP(i) − RP(i − 1) ),
32:                     newPoint( RP(i + 1) − RP(i) ) )
33:         dPlus ← dPlus + lSd
34:         if dPlus > Threshold then
35:             break
36:     if Orientation = False then
37:         for i = 1 to L − 2 do
38:             if method = "LSD" then
39:                 lSd ← Local-Shape-Distance(
40:                         newPoint( SP(i) − SP(i − 1) ),
41:                         newPoint( SP(i + 1) − SP(i) ),
42:                         newPoint( RP(L − (i + 1)) − RP(L − i) ),
43:                         newPoint( RP(L − (i + 2)) − RP(L − (i + 1)) ) )
44:             else
45:                 lSd ← Local-Shape-Distance(
46:                         newPoint( SP(i) − SP(i − 1) ),
47:                         newPoint( SP(i + 1) − SP(i) ),
48:                         newPoint( RP(L − (i + 1)) − RP(L − i) ),
49:                         newPoint( RP(L − (i + 2)) − RP(L − (i + 1)) ) )
50:             dLess ← dLess + lSd
51:             if dPlus > Threshold then
52:                 break
53:         d ← Math.min(dPlus, dLess)
54:     else
55:         d ← dPlus
56:     if d < Threshold then
57:         Threshold ← d
58:         pos ← g
59: t1 ← Time
60: if pos = −1 then
61:     result ← new Result("No match.", 0.0, (t1 − t0), [], " − ")
62: else
63:     result ← new Result(RPs[pos].Name, Threshold, (t1 −
        t0), RPs[pos].RP, " − ")
64: return ⟨result⟩
```

INTERPOLATE-GESTURE (g)

```
newG ← []
L ← length(g)
Pless1 ← P − 1
for i = 0 to Pless1 do
    factor1 ← (P − (i + 1))/Pless1
    factor2 ← i/Pless1
    factorForI ← [(L − 1) ∗ i]/Pless1
    newG[i] ← new Point( factor1 ∗ g[Math.floor(factorForI)].X+
                         factor2 ∗ g[Math.ceil(factorForI)].X,
                         factor1 ∗ g[Math.floor(factorForI)].Y+
                         factor2 ∗ g[Math.ceil(factorForI)].Y,
                         g[Math.floor(factorForI)].ID) )
return ⟨newG⟩
```

SCALAR-PRODUCT (a, u)

```
return ⟨(a.X ∗ u.X) + (a.Y ∗ u.Y)⟩
```

LOCAL-SHAPE-DISTANCE (a, b, u, v)

```
alpha ← SCALAR-PRODUCT(a, a)
beta ← SCALAR-PRODUCT(b, b)
gamma ← SCALAR-PRODUCT(u, u)
delta ← SCALAR-PRODUCT(v, v)
numerator ← 0
val ← 0
if (beta ∗ delta ≠ 0) then
    numerator ← alpha ∗ delta + beta ∗ gamma −
                2 [ SCALAR-PRODUCT(a, b) ∗ SCALAR-PRODUCT(u, v) −
                SCALAR-PRODUCT(a, v) ∗ SCALAR-PRODUCT(b, u) +
                SCALAR-PRODUCT(a, u) ∗ SCALAR-PRODUCT(b, v) ]
    val ← Math.sqrt( numerator/(beta ∗ delta) )
return ⟨val⟩
```

NORMALIZED-LOCAL-SHAPE-DISTANCE (a, b, u, v)

```
alpha ← SCALAR-PRODUCT(a, a)
beta ← SCALAR-PRODUCT(b, b)
gamma ← SCALAR-PRODUCT(u, u)
delta ← SCALAR-PRODUCT(v, v)
numerator ← 0
val ← 0
if (alpha * beta * gamma * delta ≠ 0) then
    val ← Math.sqrt( Math.max(0, (1 - ( (SCALAR-PRODUCT(a, b) *
    SCALAR-PRODUCT(u, v) + SCALAR-PRODUCT(a, u) *
    SCALAR-PRODUCT(u, v) * SCALAR-PRODUCT(b, v) - SCALAR-PRODUCT(a, v) *
    SCALAR-PRODUCT(b, u) )/( Math.sqrt(alpha) * Math.sqrt(beta) *
    Math.sqrt(gamma) * Math.sqrt(delta) ) ) ) ) )
return ⟨val⟩
```

REFERENCE-GESTURE-VECTOR STRUCTURE $(name, points, format, threshold)$

```
this.Name ← name
this.Points ← points
this.Format ← format
this.Threshold ← threshold
return
```

RESULT $(name, score, time, data, nroOfPoints, image)$

```
this.Name ← name
this.Score ← score
this.Time ← time
this.Data ← data
this.NumberOfPoints ← nroOfPoints
this.Image ← image
return
```